Research Article

Post-Quantum Secure Password-Authenticated Key Exchange Based on Ouroboros

Hao Wang,1,2 Yu Li,1,2 and Li-Ping Wang1,2

1State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100195, China
2School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Correspondence should be addressed to Li-Ping Wang; wangliping@iie.ac.cn

Received 30 November 2021; Accepted 13 May 2022; Published 14 July 2022

Academic Editor: Youwen Zhu

Copyright © 2022 Hao Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Password-authenticated key exchange (PAKE) protocols play an important role in cryptography. Most of PAKEs are based on the Diffie–Hellman key exchange protocols or RSA encryption schemes, but their security is threatened by quantum computers. In this study, we propose the first code-based PAKE protocol based on Ouroboros, which is a code-based key exchange protocol. Our scheme enjoys high efficiency and provides mutual explicit authentication, with a security reduction to decoding random quasi-cyclic codes in the random oracle model.

1. Introduction

Authenticated key exchange (AKE) allows two communicating entities to establish a common and high-entropy secret session key over an insecure communication network. In general, users need to store some prepared long-time secret keys with high entropy in devices such as smart cards and ID cards. However, the access to hardware devices makes AKE more inconvenient and complex.

To solve this shortcoming, password-authenticated key exchange (PAKE) was proposed since PAKEs only require human-memorable passwords with low entropy, such as six to eight characters. Nowadays, more and more people tend to use handheld devices, and so PAKEs have a wide range of practical applications. Nevertheless, it is difficult to design a secure key exchange protocol based on passwords, because the low entropy of a password makes it vulnerable to dictionary attacks if an adversary can get some password-dependent data. Generally speaking, there are two types of dictionary attacks. The first one is an online dictionary attack. In this attack, an adversary actively participates the execution of a protocol. For example, an adversary runs a protocol with a guessed password and observes whether the protocol succeeds. However, this type of attack is easy to avoid by just allowing an adversary to test at most a constant number of passwords per online interaction. What we need to consider is the another dictionary attack, i.e., offline dictionary attack. In this attack, an adversary can observe the execution of a protocol or interacts with the participants of the protocol. Next, the adversary tests the correctness of a guessed password offline. To avoid this attack, session keys and protocol messages must look computationally independent from passwords to the adversary.

The first PAKE protocol was proposed by Bellovin and Merritt in [1], which is called the encrypted key exchange (EKE) protocol, but they did not give a formal security proof in the protocol. Since then, a number of PAKE protocols were proposed [2–7]. However, these protocols still have no formal security proof. Until 2000, the formal security models began with the works of Bellare et al. [8] and Boyko et al. [9]. Canetti et al. introduced the universally composable notion to PAKE security model in 2005 [10].

With the security model, plenty of protocols have been designed and analyzed. On the one hand, most of PAKEs are designed using the Diffie–Hellman (DH) key exchange protocols [9, 11–15], in which the PAK [9] and PPK [15] protocols are two efficient and simple constructions of PAKEs based on the DH key exchange protocol. The PAK protocol is three-pass protocol and provides explicit authentication, and the PPK protocol is two-pass protocol and provides implicit authentication. On the other hand, some
PAKEs were designed using RSA encryption schemes [16–19]. In particular, Mackenzie et al. proposed the first RSA-based PAKE protocol with a formal security proof in 2000, which is called SNAP [17]. In 2004, Zhang showed that SNAP protocol was not practical and proposed two efficient RSA-based PAKEs [19].

Since the well-known Shor algorithm was proposed [20], the security of the DH key exchange protocols and RSA encryption schemes encounters great challenges. Fortunately, lattice-based cryptosystems and code-based cryptosystems are supposed to effectively resist attack on the quantum computers. Based on AKE protocols [21–26], several simple and efficient PAKE protocols have been designed [27, 28]. The protocols in [27] can be regarded as a parallel extension of the PAK and PPK. In the code-based cryptosystem, there are no DH-type key exchange protocols. Nevertheless, Deneuville et al. proposed a secure and efficient code-based key exchange protocol, which is called Ouroboros [29]. The Ouroboros scheme gathers the best properties of the MDPC-McEliece [30] and the HQC [31] and has a simple decoding algorithm. The security of the Ouroboros is reduced to decoding random quasi-cyclic codes in the random oracle model. As far as we know, there is no code-based provably secure PAKE scheme.

In this study, we propose the first code-based PAKE protocol based on Ouroboros with formal security proof. The protocol is constructed by using a weight-restricted hash function and enjoys several desired features, including high efficiency, mutual explicit authentication, and quantum resistance, with a security reduction in our scheme to decoding random quasi-cyclic codes in the random oracle model.

The rest of this study is organized as follows. Section 2 introduces notations used throughout the study and gives needed preliminary definitions and propositions. In Section 3, we review the security model. In Section 4, we provide a detailed description of our PAKE protocol. Section 5 gives the formal security analysis of our PAKE protocol. Section 6 provides the efficiency evaluation of our scheme. Finally, Section 7 concludes the study.

2. Preliminaries

In this section, we introduce notations and needed preliminary definitions and propositions throughout the study.

2.1. Notations. In this study, the ring of integers is denoted by ℤ, and a finite field is denoted by 𝔽ₚ with q elements, where q is a prime number. Additionally, we denote the Hamming weight of a vector by ω(·), i.e., the number of its nonzero coordinates. 𝕀ⁿ denotes a vector space of dimension n over 𝔽₂ for some positive n ∈ ℤ. Elements of 𝕀ⁿ can be considered as row vectors or polynomials in ℤ[x]/(xn − 1) and represented by lower-case bold letters. The product of two elements 𝑥, 𝑦 ∈ 𝕀ⁿ is defined similarly as in ℤ[x], i.e., 𝑥 ⋅ 𝑦 = 𝑐 ∈ 𝕀ⁿ with

\[ c_m = \sum_{i+j=m \mod n} x_i y_j \quad \text{for } m \in \{0, 1, \ldots, n-1\}. \]  \hspace{1cm} (1)

For any finite set T, x ~ T denotes x is a uniformly random element sampled from T. For an event E, Ê denotes the complementary event of E.

In particular, we also use the symbol \( \sigma^m_w(F_2^n) \) as follows:

\[ \sigma^m_w(F_2^n) = \{ x \in F_2^n : \omega(x) = w \}. \]  \hspace{1cm} (2)

2.2. Coding Theory. We now focus on relevant basic definitions and properties relating to coding theory.

Definition 1 (Linear Code). A linear \([n, k]_q\) code \( C \) with length n and dimension k over \( F_q \) is a subspace of \( F_q^n \).

Definition 2 (Generator Matrix). A generator matrix for a linear \([n, k]_q\) code \( C \) is a matrix \( G \in F_q^{k \times n} \) whose rows form a basis for \( C \), i.e.,

\[ C = \{ \mu G \mid \mu \in F_q^k \}. \]  \hspace{1cm} (3)

Definition 3 (Parity Check Matrix). A parity check matrix \( H \in F_{q}(n-k) \times n \) for a linear code \( C \) is a generator matrix for the dual code \( C^⊥ \), i.e.,

\[ C^⊥ = \{ x \in F_q^n : Hx^T = 0 \}. \]  \hspace{1cm} (4)

Definition 4 (Quasi-Cyclic Codes [30]). Given positive integers s, n, and k, a \([sn, k]_q\) linear code \( C \) is quasi-cyclic (QC) of order s if for any \( c = (c_1, c_2, \ldots, c_s) \in C \) it holds that \( (Xc_1, Xc_2, \ldots, Xc_s) \in C \).

Definition 5 (Systematic Quasi-Cyclic Codes of Rate 1/s [30]). A systematic quasi-cyclic \([sn, n]_q\) code of order s is a quasi-cyclic code with a parity check matrix of the form

\[
H = \begin{bmatrix}
I_n & 0 & \cdots & 0 & A_1 \\
0 & I_n & \cdots & 0 & A_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & I_n & A_{s-1}
\end{bmatrix},
\]  \hspace{1cm} (5)

where \( A_1, \ldots, A_{s-1} \) are \( n \times n \) circulant matrices.

Next, we define the syndrome decoding (SD) problem over \( F_2 \) in the Hamming metric.

Definition 6 (SD Distribution). Given positive integers \( n, k, w \), the SD distribution \( H \rightarrow \delta_w(F_2^n) \) and \( x \sim \delta_w(F_2^n) \) and outputs \( (H, \sigma(x) = Hx^T) \).

Definition 7 (Search SD Problem). Given \( (H, y^T) \in F_2^{n \times (n-k)} \times F_2^{n \times k} \) from the SD distribution, the search SD problem decides whether there exists \( x \in \delta_w(F_2^n) \), such that \( Hx^T = y^T \).

Definition 8. (Decisional SD Problem) On input \( (H, y^T) \in F_2^{n \times (n-k)} \times F_2^{n \times k} \), the decisional SD problem DSD \((n, k, w)\) needs to decide with nonnegligible advantage...
whether \((H, y)\) came from the SD distribution or the uniform distribution over \(F_2^{(n-k) \times n} \times F_2^{s\cdot n}\).

The search SD problem in the Hamming metric has been proved NP complete over binary field in the worst case [32], and the decisional SD problem has been proved to be polynomially equivalent to its search version [33].

Then, we describe relevant definitions of quasi-cyclic codes.

**Definition 9** (s-QCSD Distribution). Given positive integers \(n, k, w, s\), the s-QCSD \((n, k, w, s)\) distribution chooses \(H \sim F_2^{(n-k) \times n}\) and \(x = (x_1, \ldots, x_s)\), where \(x_i \sim \delta_{w_i}^n (F_2)\), and outputs \((H, \sigma(x) = Hx')\).

**Definition 10** (Search s-QCSD Problem). Given positive integers \(n, k, w, s\), a random systematic parity check matrix \(H \in F_2^{(n-k) \times n}\) of a quasi-cyclic code, and \(y \sim F_2^{sn}\), the search s-QCSD problem decides whether there exists \(x = (x_1, \ldots, x_s)\), where \(x_i \sim \delta_{w_i}^n (F_2)\), such that \(Hx' = y'\).

**Definition 11** (Decisional s-QCSD Problem). Given positive integers \(n, k, w, s\), a random parity check matrix \(H \in F_2^{(n-k) \times n}\) of a systematic quasi-cyclic code \(C\) and \(y \sim F_2^{sn}\), the decisional s-QCSD (s-DQCSD) problem asks to decide with nonnegligible advantage whether \((H, y')\) came from the s-QCSD distribution or the uniform distribution over \(F_2^{(n-k) \times n} \times F_2^{sn}\).

2.2. The s-QCSD Problem Assumption [29]. The search s-QCSD problem is hard on average. Let \(\mathcal{A}\) be a probabilistic polynomial time adversary. The input of \(\mathcal{A}\) is a random parity check matrix \(H \in F_2^{(n-k) \times n}\) and a vector \(y \in F_2^n\). The probability \(\text{Adv}_{\mathcal{A}}^{\text{s-QCSD}}\) of outputting vector \(x = (x_1, \ldots, x_s)\) with \(Hx' = y'\) and \(\omega(x) = w\) by \(\mathcal{A}\) is negligible, where

\[
\text{Adv}_{\mathcal{A}}^{\text{s-QCSD}} = \Pr[H \sim F_2^{(n-k) \times n}, y \sim F_2^n, x \sim \mathcal{A}(H, y, w)]
\]

\[
x = (x_1, \ldots, x_s); x_i \sim \delta_{w_i}^n (F_2); x \in \mathcal{A}(H, y, w).
\]

(6)

Let \(\text{Adv}_{\mathcal{A}}^{\text{s-QCSD}} (t) = \max_{\mathcal{A}} \left\{ \text{Adv}_{\mathcal{A}}^{\text{s-QCSD}} \right\}\), where the maximum is over all adversaries of time complexity at most \(t\).

2.3. Weight-Restricted Hash Function. Our PAKE protocol uses a weight-restricted hash function proposed in the RaCoSS scheme [34]. We denote this weight-restricted hash function by \(\mathcal{H}\). Although the RaCoSS scheme has been broken, \(\mathcal{H}\) is still secure under proper parameters [35]. Before giving the description of weight-restricted hash function, let us first introduce the definition of collision-resistant hash function.

**Definition 12**. A hash function \(\mathcal{H}\) is called collision-resistant hash function if the probability that any adversary \(\mathcal{A}\) finds two distinct values that satisfies the following condition is negligible:

\[
\Pr[(M, M') \sim \mathcal{A}: (M \neq M') \land (\mathcal{H}(M) = \mathcal{H}(M'))] \leq \text{negl}(\lambda),
\]

where \(\text{negl}(\lambda)\) is a negligible function.

**Lemma 1** (see [36]). Given a collision-resistant hash function \(\mathcal{H}: \{0, 1\}^* \rightarrow \{0, 1\}^{\kappa}\), and an encode algorithm \(\text{Enc}(\cdot)\), which encode any \(n_l\) bit message into \(n_s\) bit message with Hamming weight \(w_s\), the function \(\text{Enc}(\cdot): \{0, 1\}^* \rightarrow \{0, 1\}^{\kappa}\) is still a collision-resistant hash function.

We put the weight-restricted hash function \(\mathcal{H}\) in Algorithm 1. Simply speaking, \(\mathcal{H}\) uses a byte string message \(m\), two integers \(w_s\) and \(n_s\) as input. Then, SHA3-512 is used to calculate the position of 1. The output is a \(n\)-bit string with Hamming weight \(w_s\). According to Lemma 1, \(\mathcal{H}\) is a collision-resistant hash function.

2.4. Ouroboros Scheme. We now recall the Ouroboros scheme from [29], which is the basic of our PAKE protocol. The Ouroboros scheme uses a function \(f_w\) proposed in [37]. Simply speaking, giving the positions of the “1” is enough to obtain random vectors with fixed weight \(w_s\). For more details about this function, we refer the readers to [37]. Besides, the Ouroboros scheme uses a hash function \(\mathcal{H}': \{0, 1\}^* \rightarrow \delta_{w_s}^n (F_2)\). The decoding algorithm CE decoder is presented in Algorithm 2. In short, it puts \((x, y, \epsilon) = (x_f, y_f, w_f)\), threshold value \(t\), the weight \(w_f\) of \(r_f\) and \(r_s\), and the weight \(w_s\) of \(e\) as input. The output of this algorithm is \((r_1, r_2)\) if it succeeds.

The Ouroboros scheme is presented as follows:

(a) **Alice** generates \(h \sim F_2^2\) and \((x, y) \sim \delta_{w_f}^n (F_2)\). Then, she computes \(s = x + hy\). Finally, she sends \((h, s)\) to Bob.

(b) **Bob** generates \(r_1, r_2 \sim \delta_{w_s}^n (F_2)\) and computes \(e_s = f_w(\mathcal{H}(r_1, r_2))\). Next, he randomly generates \(e \sim \delta_{w_f}^n (F_2)\). Then, he computes \(s_1 = r_1 + tr_2 + \epsilon\) and \(s_2 = s_1 + \epsilon + e\). Finally, he sends \((s_1, s_2)\) to Alice and saves the vector \(\epsilon\) as the shared secret.

(c) **Alice** computes \(e_s = s_2 - ys_1 = xr_2 + yr_1 + e_s + e\) when receiving \(s_1\) and \(s_2\). Then, she uses the decoding algorithm CE decoder \((x, y, e_s, t, w_f, w_s, e)\) to get \(r_f\) and \(r_s\). Finally, she computes the shared secret \(\epsilon = e_s - xr_2 + yr_1 - f_w(\mathcal{H}(r_1, r_2))\).

**Theorem 1** (see [29]). The Ouroboros protocol satisfies indistinguishability under chosen plaintext attack (IND-CPA) under the 2-DQCSD and 3-DQCSD assumptions.

3. Security Model

This section reviews the formal security model from [8]. Consider the form of a PAKE protocol with two users. Users are expected to establish and use the same keys over the network that is fully controlled by a probabilistic and polynomial time adversary \(\mathcal{A}\). The adversary \(\mathcal{A}\) can initialize protocol
communications between user instances, deliver messages to
unintended recipients, and observe their reaction according to
the protocol. The adversary can reveal the session keys
established by user instances and enumerate all the passwords
in the password space \( D \) in the offline attack. In the following,
we perform the formal description of the security model.

3.1. Security Game. Let \( P \) be a PAKE protocol and \( \mathcal{U} \) be the
fixed set of users. A two-party protocol in PAKE model is
considered, and the users in \( \mathcal{U} \) are partitioned to two non-
empty entities, called Alice and Bob (A and B for short).
Before the game starts, for each entity, a password \( \pi \) is
chosen uniformly at random from password space \( D \). A set

\[
\begin{align*}
\text{Input:} & \quad \text{Message byte string } m, \text{ integer } w, \text{ integer } n. \\
\text{Output:} & \quad \text{A } n\text{-bit string with Hamming weight } w. \\
(1) & \quad \ell \leftarrow \lceil \log_2 n \rceil \\
(2) & \quad I \leftarrow \emptyset \\
(3) & \quad S \leftarrow \{m, 0x00|m, 0x01|m, 0x02|m, \ldots, 0xFF|m\} \\
(4) & \quad \text{for } s \in S \text{ do} \\
(5) & \quad x \leftarrow \text{SHA3-512}(s) \\
(6) & \quad (x_1, x_2, \ldots, x_{\lceil 512/\ell \rceil}) \leftarrow x \\
(7) & \quad \text{for } i \in [1, \ell] \text{ do} \\
(8) & \quad y_i \leftarrow \text{int}(x_i) \\
(9) & \quad \text{if } y_i < n \text{ then} \\
(10) & \quad I \leftarrow I \cup \{y_i\} \\
(11) & \quad \text{end if} \\
(12) & \quad \text{if } |I| = w \text{ then} \\
(13) & \quad \text{Break} \\
(14) & \quad \text{end if} \\
(15) & \quad \text{end for} \\
(16) & \quad \text{end for} \\
(17) & \quad \text{if } |I| < w \text{ then} \\
(18) & \quad \text{Outputs } \bot \\
(19) & \quad \text{else} \\
(20) & \quad \text{Outputs a } n\text{-bit string where } I_i\text{-th bit is } 1 (I_i \in I, 0 \leq i \leq w - 1) \text{ and other bits are } 0. \\
(21) & \quad \text{end if}
\end{align*}
\]

\text{ALGORITHM 1: Weight-restricted hash function.}
of efficiently computable cryptographic functions is specified, i.e., hash functions, and the public cryptographic parameters are generated.

3.2. User Instances. During the model, there are an unlimited number of instances running the protocol simultaneously for each user. The instance of user A is denoted as $\Pi^i_A$. An instance $\Pi^i_A$ accepts at any time, and the same rule applies to user B. When an instance accepts, it possesses a partner id pid, a session id sid, and a session key sk. The pid is the identity of the user instance that the current instance believes it is talking to. The sid is a string, which uniquely identifies this session. In general, the sid is composed of the concatenation of all messages sent and received by the instance $\Pi^i_A$ (or $\Pi^j_B$). The sk is the final target to be calculated.

3.3. Queries. The queries that adversary $\mathcal{A}$ may make during the game are as follows [8]:

(i) **Send** $(A, i, M)$: the message $M$ is send to instance $\Pi^i_A$. The instance computes what the protocol specifies and outputs the result to the adversary. We assume whether the instance accepts or not is visible to the adversary.

(ii) **Execute** $(A, i, B, j)$: an honest execution between two instances $\Pi^i_A$ and $\Pi^j_B$ is carried out, where $A \neq B$ and $\Pi^i_A$ and $\Pi^j_B$ were not used before. Finally, the transcript of this execution is given to the adversary.

(iii) **Reveal** $(A, i)$: the session key $sk^i_A$ of $\Pi^i_A$ is returned to the adversary.

(iv) **Test** $(A, i)$: this query is valid if and only if the instance is fresh, as defined below. In this case, $\Pi^i_A$ generates a random bit $b$. If $b = 1$, the real session key $sk^i_A$ is sent to the adversary; otherwise, a random session key chosen uniformly from the sk space is sent. This query is allowed only once during the game.

(v) **Oracle** $(M)$: this gives the adversary oracle access to a function $h$, which is selected at random from some probability space.

Now, we give some definitions for the formal security model.

**Definition 13** (Partnering). Let $\Pi^i_A$ and $\Pi^j_B$ be a pair of instances. $\Pi^i_A$ and $\Pi^j_B$ are partner instances if both have accepted and have the same unique session id sid and the same session key sk.

**Definition 14** (Freshness). $\Pi^i_A$ is fresh if (i) it has accepted, and (ii) an adversary $\mathcal{A}$ has not queried **Reveal**$(A, i)$ or **Reveal**$(B, j)$, where $\Pi^j_B$ is $\Pi^i_A$’s partner, if it has.

**Definition 15** (Correctness). If $\Pi^i_A$ and $\Pi^j_B$ are partner instances and both are accepted, then they conclude with the same session key sk.

**Definition 16**. Let $\text{Succ}_3$ be the event that adversary $\mathcal{A}$ asks a single test query on a fresh instance and outputs a bit $b'$ with $b' = b$ at the end of the game. The advantage of the adversary $\mathcal{A}$ is defined as follows:

$$\text{Adv}^D(\mathcal{A}) = 2\Pr(\text{Succ}_3) - 1.$$  

The following is definition of secure PAKE protocol, which is the same as in [38].

**Definition 17**. A protocol $P$ is called a secure PAKE protocol if for every polynomial time adversary $\mathcal{A}$ that makes at most $n_{\text{se}}$ ($n_{\text{se}} \leq |\mathcal{D}|$) queries of Send type to different instances, the following inequality holds, $\text{Adv}^P(\mathcal{A}) \leq n_{\text{se}}|\mathcal{D}| + \varepsilon$, where $|\mathcal{D}|$ means the size of the password space and $\varepsilon$ is a negligible function of security parameters.

4. Our PAKE Protocol

In this section, we present an efficient Ouroboros-based PAKE protocol called OPAKE. Our protocol is described in a generic fashion in Figure 1. We use a weight-restricted hash function $\mathcal{H}: \{0, 1\}^* \rightarrow \{0, 1\}^k$ as mentioned earlier. Hash functions are defined as follows: $\mathcal{H}_1: \{0, 1\}^* \rightarrow \mathcal{D}_w(F_2)$, $\mathcal{H}_2, \mathcal{H}_3: \{0, 1\}^* \rightarrow \{0, 1\}^k$, where $k$ is a security parameter. Assume that $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$, and $\mathcal{H}$ are independent random functions. Let $\mathcal{D}$ be the password space, and $\pi \in \mathcal{D}$ be the password that Alice and Bob share. Then, our protocol is shown as follows.

4.1. OPAKE

(a) **Alice** generates $R \leftarrow \{0, 1\}^k$ and chooses $h^{\text{op}} \leftarrow \mathcal{D}_w(F_2)$ and $x, y \leftarrow \mathcal{D}_w(F_2)$. Next, she computes $a = \mathcal{H}_1(n, R, A, B, h)$. Then, she computes $s = a + x + hy$. Finally, she sends $(A, h, s, R)$ to Bob.

(b) **Bob** tests whether $s, h \in \mathcal{D}_w$. If not, then he rejects the protocol. Otherwise, he generates $r_1, r_2 \leftarrow \mathcal{D}_w(F_2)$ and computes $e = f_w(\mathcal{H}_1(r_1, r_2))$. Next, he randomly generates $\varepsilon \leftarrow \mathcal{D}_w(F_2)$. Then, he computes $a = \mathcal{H}_1(n, R, A, B, h)$. Finally, he computes $s_1 = r_1 + hr_2$ and $s_2 = (s - a)r_2 + e_r + \varepsilon$ and sends $(B, s_1, s_2)$ to Alice.

(c) **Alice** computes $e = s_2 - y s_1 = x r_2 - y r_1 + e_r + \varepsilon$ when receiving $s_1$ and $s_2$. Then, she uses the decoding algorithm CE decoder $(x, y, e_1, t, w, u, w + u)$ to get $r_1$ and $r_2$. Then, she computes $\rho = e_2 - x r_2 - y r_1 - f_w(\mathcal{H}_1(r_1, r_2))$. Finally, she computes $\mu = \mathcal{H}_3(\rho, R, A, B, h)$ and sends $(A, \mu)$ to Bob.

(d) **Bob** tests whether $\mu$ is equal to $\mathcal{H}_3(\rho, R, A, B, h)$. If not, then he rejects. Otherwise, he accepts and computes $\eta = \mathcal{H}_2(\rho, R, A, B, h)$ and a session key $sk = \mathcal{H}_3(\rho, R, A, B, h)$. Finally, he sends $(B, \eta)$ to Alice.

(e) **Alice** tests whether $\eta$ is equal to $\mathcal{H}_3(\rho, R, A, B, h)$. If not, then she rejects. Otherwise, she accepts and computes a session key $sk = \mathcal{H}_3(\rho, R, A, B, h)$. If
Alice
Password: \(\pi\)
\[
\begin{align*}
S & \leftarrow \{0,1\}^k \\
R & \leftarrow \mathbb{F}_2^n \\
\rho & \leftarrow \mathbb{F}_2^n \\
x, y & \leftarrow \mathbb{F}_2^n \\
a & = H_{\text{sec}}(\pi, R, A, B, h) \\
s & = a + x + hy \\
\end{align*}
\]

Bob
Password: \(\pi\)
\[
\begin{align*}
A, h, S, R & \leftarrow \mathbb{F}_2^n \\
r_1, r_2 & \leftarrow \mathbb{F}_2^n \\
e & \leftarrow \mathbb{F}_2^n \\
a & = H_{\text{sec}}(\pi, R, A, B, h) \\
s_1 & = r_1 + hr_2 \\
s_2 & = (s - a) r_2 + e + \epsilon \\
\end{align*}
\]

\[\begin{align*}
A, h, S, R & \rightarrow \mathbb{F}_2^n
\end{align*}\]

\[\begin{align*}
B, S_1, S_2 & \rightarrow \mathbb{F}_2^n
\end{align*}\]

\[\begin{align*}
\eta & = H_1(\rho, R, A, B, h) \\
\text{if not reject} & \\
sk & = H_3(\rho, R, A, B, h)
\end{align*}\]

\[\begin{align*}
\eta & = H_2(\rho, R, A, B, h) \\
\text{if not reject} & \\
sk & = H_3(\rho, R, A, B, h)
\end{align*}\]

\[\begin{align*}
\mu & = H_1(\rho, R, A, B, h) \\
\text{if not reject} & \\
sk & = H_3(\rho, R, A, B, h)
\end{align*}\]

\[\begin{align*}
A, \mu & \rightarrow \mathbb{F}_2^n
\end{align*}\]

\[\begin{align*}
B, \eta & \rightarrow \mathbb{F}_2^n
\end{align*}\]

\[\begin{align*}
A, h, S, R & \rightarrow \mathbb{F}_2^n
\end{align*}\]

\[\begin{align*}
B, S_1, S_2 & \rightarrow \mathbb{F}_2^n
\end{align*}\]

4.2. Correctness. To show the correctness of our protocol, it is sufficient to show that the material derived from Alice and Bob is the same, i.e., \(\epsilon = \rho\). Firstly, honest Alice sends reasonable \((A, h, s, R)\) to Bob, and honest Bob uses the correct password \(\pi\) to compute \(a\). Then, Bob chooses the secret material \(\epsilon\) and encrypts it into the ciphertext \((s_1, s_2)\). Next, Alice decrypts \((s_1, s_2)\) and gets the same material according to the Ouroboros scheme. Hence, the correctness of our PAKE protocol is verified when the two participants execute the protocol honestly.

5. Formal Security Analysis of OPAKE

This section gives the formal security of OPAKE under the security model defined in Section 3. We prove that an adversary \(\mathcal{A}\) attacking the protocol OPAKE is unable to determine the \(sk\) of a fresh instance with advantage greater than that of an online dictionary attack.

Theorem 2. Let \(\mathcal{A}\) be an adversary, which runs in time \(t\), and the adversary’s advantage in attacking the protocol is bounded by

\[
\text{Adv}^{\text{OPAKE}}(\mathcal{A}) \leq \frac{n_{\text{exe}}}{|\mathcal{D}|} + O\left(n_{\text{exe}} + n_{\text{se}}\right)\left(\text{Adv}^{\text{QCS}}(\mathcal{O}(t)) + \text{Adv}^{\text{QCS}}(\mathcal{O}(t))\right) + \left(n_{\text{exe}} + n_{\text{se}}\right)\frac{n_{\text{exe}} + n_{\text{se}}}{\mathcal{P}}^2.
\]

where \(|\mathcal{D}|\) denotes the size of the password space, \(n_{\text{exe}}\) denotes the number of the queries of type \text{Execute} oracle, \(n_{\text{se}}\) denotes the number of the queries of the random oracle, and \(n_{\text{se}}\) denotes the number of queries of type \text{Send} oracle.

To prove this theorem, we present a sequence of hybrid experiments by \(P_0, P_1, P_2, P_3,\) and \(P_4\) and denote the advantage of \(\mathcal{A}\) when attacking in the experiment \(P_i\) by \(\text{Adv}(\mathcal{A}, P_i)\).

Protocol \(P_0\) in this protocol, the adversary \(\mathcal{A}\) makes a number of oracle queries in Section 3, i.e.,

\[\text{Adv}(\mathcal{A}, P_0) = \frac{n_{\text{exe}}}{|\mathcal{D}|} + O\left(n_{\text{exe}} + n_{\text{se}}\right)\left(\text{Adv}^{\text{QCS}}(\mathcal{O}(t)) + \text{Adv}^{\text{QCS}}(\mathcal{O}(t))\right) + \left(n_{\text{exe}} + n_{\text{se}}\right)\frac{n_{\text{exe}} + n_{\text{se}}}{\mathcal{P}}^2.
\]

\[\text{Sen d, Execute, Reveal, and Test. Besides, the adversary \(\mathcal{A}\) has access to five independent random oracles \(\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3,\) and \(\mathcal{H}_4\). Each random oracle is simulated by a list of input-output pairs. When receiving a new query input \(x\), the oracle checks whether \(x\) was queried before. If there exists \(x_i = x\), then the oracle returns the output in the list. Otherwise, the oracle generates a random number and returns it. Then, the oracle adds the new pair to the list.

Protocol \(P_1\): this protocol is identical to \(P_0\) except that if the oracle \text{Execute} \((A, i, B, j)\) is called between two instances \(\Pi_A^j\) and \(\Pi_B^j\) then the session keys \(sk_A^j\) and \(sk_B^j\) are set equal to a
random number selected from \([0, 1]^k\). Now, we prove that change in the \textbf{Execute} oracle affects the advantage of \(\mathcal{A}\) in a negligible value.

\[
\left| \text{Adv}(\mathcal{A}, P_1) - \text{Adv}(\mathcal{A}, P_0) \right| \leq n_{\text{exe}} \text{Adv}^{2-\text{QSD}}(\mathcal{O}(t)) + n_{\text{exe}} n_{\text{ro}} \frac{2^k}{2^{n_{\text{exe}}}}. \tag{10}
\]

where \(n_{\text{exe}}\) denotes the number of the queries of type \textbf{Execute} oracle, \(n_{\text{ro}}\) denotes the number of the queries of random oracle, and \(t\) denotes the running time of \(\mathcal{A}\).

**Proof of Claim 1:** First, we show that the probability of same random numbers generated in two different Execute oracle queries is very small. In this situation, \(\mathcal{A}\) can distinguish \(P_0\) and \(P_1\) by making the Reveal query simplify. According to the birthday paradox, it is easy to show that the probability of this situation occurring is bounded by \(\mathcal{O}(n_{\text{exe}}^2 2^n)\).

Now, we fix an oracle Execute \((A, i, B, j)\) and assume that the random numbers \((h, s, R)\) and \(\epsilon\) in this oracle call are not used in previous \textbf{Execute} query. Without the knowledge of \(\epsilon\), the output of \(\mathcal{H}_3\) is indistinguishable from a random number uniformly selected from \([0, 1]^k\). Thus, the adversary \(\mathcal{A}\) can distinguish \(P_0\) and \(P_1\) if and only if \(\mathcal{A}\) can recover the information of \(\epsilon\). We denote the probability that \(\mathcal{A}\) recovers the information of \(\epsilon\) by \(p_\epsilon\). We give two games \(G_1\) and \(G_2\) to bound \(p_\epsilon\).

**Game \(G_1\):** the adversary carries an honest execution between instances \(\Pi_1^A\) and \(\Pi_1^B\):

1. \(\Pi_1^A\) generates \(\mathcal{R} \leftarrow [0, 1]^k\) and chooses \(h \leftarrow F_2^n\) and \(x, y \leftarrow \delta^w_{\text{mu}}(F_2)\). Next, \(\Pi_1^A\) queries the random oracle \(\mathcal{H}_1\) on \((\pi, R, A, B, h)\). The reply of \(\mathcal{H}_1\) is denoted by \(\alpha\). Then, \(\Pi_1^A\) computes \(s = a + x + hy\). Finally, \(\Pi_1^A\) sends \((A, h, s, R)\) to \(\Pi_1^B\).

2. \(\Pi_1^B\) generates \(r_1, r_2 \leftarrow \delta^w_{\text{mu}}(F_2)\), queries the random oracle \(\mathcal{H}_1\) on \((r_1, r_2)\), and computes \(c_\sigma = f_w(\mathcal{H}_1(r_1, r_2))\). Next, \(\Pi_1^B\) randomly generates \(\sigma \leftarrow \delta^w_{\text{mu}}(F_2)\). Then, \(\Pi_1^B\) queries the random oracle \(\mathcal{H}_1\) on \((\pi, R, A, B, h)\). The reply of \(\mathcal{H}_1\) is denoted by \(\eta\). Finally, \(\Pi_1^B\) computes \(s_1 = r_1 + hr_2\) and \(s_2 = (s - a)r_2 + \epsilon + \eta\) and sends \((B, s_1, s_2)\) to \(\Pi_1^A\).

3. \(\Pi_1^A\) accepts \(c_\sigma = s_2 - y s_1\) when receiving \(s_1\) and \(s_2\). Next, \(\Pi_1^A\) uses the decoding algorithm CE decoder \((x, y, c_\sigma, t, u, w, w_e)\) to get \(r_1, r_2\). Then, \(\Pi_1^A\) computes \(\rho = c_\sigma - x r_2 + y r_1 - f_w(\mathcal{H}_1(r_1, r_2))\). Finally, \(\Pi_1^A\) queries the random oracle \(\mathcal{H}_1\) on \((\rho, R, A, B, h)\) (denoted by \(\mu\)) and sends \((A, \mu)\) to \(\Pi_1^B\).

4. After receiving \(\mu\) from \(\Pi_1^B\), \(\Pi_1^A\) queries the random oracle \(\mathcal{H}_2\) on \((\epsilon, R, A, B, h)\) (denoted by \(\eta\)). Finally, \(\Pi_1^A\) accepts and sends \((B, \eta)\) to \(\Pi_1^A\).

5. After receiving \(\eta\) from \(\Pi_1^B\), \(\Pi_1^A\) accepts. This game ends and adversary \(\mathcal{A}\) outputs its guess of \(\epsilon\).

**Game \(G_2\):** this game is similar to game \(G_1\), except that instances \(\Pi_1^A\) and \(\Pi_1^B\) do not query random oracles \(\mathcal{H}_1\) and \(\mathcal{H}_2\):

1. \(\Pi_1^A\) generates \(R \leftarrow [0, 1]^k\) and chooses \(h \leftarrow F_2^n\) and \(x, y \leftarrow \delta^w_{\text{mu}}(F_2)\). Next, \(\Pi_1^A\) queries the random oracle \(\mathcal{H}_1\) on \((\pi, R, A, B, h)\). The reply of \(\mathcal{H}_1\) is denoted by \(\alpha\). Then, \(\Pi_1^A\) computes \(s = a + x + hy\). Finally, \(\Pi_1^A\) sends \((A, h, s, R)\) to \(\Pi_1^B\).

2. \(\Pi_1^B\) generates \(r_1, r_2 \leftarrow \delta^w_{\text{mu}}(F_2)\), queries the random oracle \(\mathcal{H}_1\) on \((r_1, r_2)\), and computes \(c_\sigma = f_w(\mathcal{H}_1(r_1, r_2))\). Next, \(\Pi_1^B\) randomly generates \(\sigma \leftarrow \delta^w_{\text{mu}}(F_2)\). Then, \(\Pi_1^B\) queries the random oracle \(\mathcal{H}_1\) on \((\pi, R, A, B, h)\). The reply of \(\mathcal{H}_1\) is denoted by \(\eta\). Finally, \(\Pi_1^B\) computes \(s_1 = r_1 + hr_2\) and \(s_2 = (s - a)r_2 + \epsilon + \eta\) and sends \((B, s_1, s_2)\) to \(\Pi_1^A\).

3. After receiving \(s_1\) and \(s_2\) from \(\Pi_1^B\), \(\Pi_1^A\) sends a random number \(\epsilon \leftarrow [0, 1]^k\) to \(\Pi_1^B\).

4. After receiving \(\mu\) from \(\Pi_1^B\), \(\Pi_1^A\) sends a random number \(\eta \leftarrow [0, 1]^k\) to \(\Pi_1^B\).

5. After receiving \(\eta\) from \(\Pi_1^B\), \(\Pi_1^A\) accepts. This game ends, and adversary \(\mathcal{A}\) outputs its guess of \(\epsilon\).

We denote the probability that \(\mathcal{A}\) guesses the correct \(\epsilon\) in game \(G_1\) by \(p_\epsilon(G_i)\), \(i \in [1, 2]\). Let \(E\) be the event that \(\mathcal{A}\) guesses the correct \(\epsilon\) on oracle \(\mathcal{H}_1\) or \(\mathcal{H}_2\) with \((\epsilon, R, A, B, h)\).

\[
p_\epsilon = p_\epsilon(G_1) = p_\epsilon(G_1|E) \Pr(E) + p_\epsilon(G_1|\overline{E}) \Pr(\overline{E}) 
\leq \Pr(E) + p_\epsilon(G_1|\overline{E}). \tag{11}
\]

Obviously, the probability that \(\mathcal{A}\) guesses the correct \(\epsilon\) is as follows:

\[
Pr(E) = \frac{n_{\text{ro}}}{C_{\text{ro}}} \tag{12}
\]

where \(n_{\text{ro}}\) means the number of queries to random oracle \(\mathcal{H}_1\) and \(\mathcal{H}_2\). By assuming \(C_{\text{ro}} \geq 2^k\), we have \(\Pr(E) \leq n_{\text{ro}}/2^k\). It is easy to see that \(\mu\) and \(\eta\) are indistinguishable from the random numbers in \([0, 1]^k\) when \((\epsilon, R, A, B, h)\) is not queried in random oracle \(\mathcal{H}_1\) and \(\mathcal{H}_2\). So, \(p_\epsilon(G_1|\overline{E}) = p_\epsilon(G_2)\). In the following, we show that
\[ p_t(G_2) \leq \text{Adv}^{3-\text{QCSD}}(\Theta(t)) + \text{Adv}^{3-\text{QCSD}}(\Theta(t)). \]

Given a reasonable \((h, s)\), finding the correct information \(\alpha\) such that \(s = \alpha + x + hy\) is a 2-QCSD problem. Besides, given \((h, s)\) and \(x^\frac{1}{8}w^e(F_2)\), we construct an algorithm \(W\) to solve 3-QCSD problem by running \(A\) on a simulation of the game \(G_2\). Algorithm \(W\) runs exactly as \(G_2\) except for the change that \(\pi_B^i\) computes the encryption of message \(x\) in step (2). If \(A\) returns the correct \(x\), then our algorithm \(W\) can solve the 3-QCSD problem. Hence,

\[ |\text{Adv}(A, P_1) - \text{Adv}(A, P_0)| \leq n_{ex} \text{Adv}^{2-\text{QCSD}}(\Theta(t)) + n_{ex} \text{Adv}^{3-\text{QCSD}}(\Theta(t)) + n_{ex} n_{se}. \]

Thus, the claim is desired.

**Protocol P2:** this protocol is identical to \(P_1\) except when \(\pi_B^0\) receives the message \((A, h, s, R)\) generated from \(\pi_A^i\), and if \(\pi_A^i\) and \(\pi_B^0\) both accept, they are given the same random

\[ |\text{Adv}(A, P_2) - \text{Adv}(A, P_1)| \leq n_{ex} \text{Adv}^{2-\text{QCSD}}(\Theta(t)) + n_{ex} n_{se} \text{Adv}^{3-\text{QCSD}}(\Theta(t)). \]

where \(n_{ex}\) denotes the number of queries of the type Send and \(t\) denotes the running time of \(A\).

**Proof of Claim 2:** when instance \(\pi_B^0\) receives the message \((A, h, s, R)\) generated from \(\pi_A^i\), the message \((B, s_1, s_2)\) is sent by \(\pi_A^i\) and got by the adversary \(A\). Since the message \((h, s, R)\) was generated by \(\pi_A^i\), \(A\) has no information about the secret private key \((x, y)\) according to the 2-QCSD problem. The advantage of solving the message \(\epsilon\) from \(s_1\) and \(s_2\) by the adversary \(A\) is bounded by \(\text{Adv}^{3-\text{QCSD}}(\Theta(t))\). In conclusion, suppose the adversary \(A\) makes \(n_{ex}\) oracle queries of Send oracle, adversary's advantage of distinguishing between \(P_1\) and \(P_2\) is bounded by \(n_{ex} \text{Adv}^{2-\text{QCSD}}(\Theta(t)) + n_{ex} n_{se} \text{Adv}^{3-\text{QCSD}}(\Theta(t))\). Therefore, the claim is desired.

**Protocol P2:** in this protocol, we assume that instance \(\pi_B^0\) receives a message \((B, s_1, s_2)\) generated from instance \(\pi_B^0\), while \(\pi_B^0\) receives a message \((A, h, s, R)\) from \(\pi_A^i\). In this situation, both \(\pi_A^i\) and \(\pi_B^0\) accept and are given a same random session key \(sk \leftarrow \{0, 1\}^k\) if their session key were not replaced with a random key in protocol \(P_2\).

**Claim 3.** For any polynomial time adversary \(A\) making \(n_{ex}\) queries of Send oracle to different instances, we obtain

\[ \text{Adv}(A, P_4) \leq \text{Adv}^{2-\text{QCSD}}(\Theta(t)) + \text{Adv}^{3-\text{QCSD}}(\Theta(t)). \]

Moreover,

\[ \text{Adv}(A, P_5) \leq n_{ex} n_{ro} + \text{Adv}^{2-\text{QCSD}}(\Theta(t)) + \text{Adv}^{3-\text{QCSD}}(\Theta(t)). \]

Suppose \(A\) makes \(n_{ex}\) queries of Execute at all. We have

\[ \text{Adv}(A, P_5) = \text{Adv}(A, P_2). \]
Proof of Claim 5: we consider two cases.

**Case 1.** Assume that instance $\pi_1'$ receives a message $(B, s_1, s_2)$ generated from adversary $\mathcal{A}$. After receiving the message, instance $\pi_1'$ computes $\rho$ from the message. Then, $\pi_1'$ gets $\mu$ from the query of the random oracle $\mathcal{H}_1$ on $(\rho, R, A, B, h)$. Next, $\pi_1'$ returns $\mu$ to the adversary. In this situation, $\mathcal{A}$ generates from adversary $\mathcal{A}$ has no knowledge of $\rho$. Another possibility is that $\mathcal{A}$ generates $\rho$ with probability $1/2^k$ since $\mathcal{A}$ has no knowledge of $\rho$. Another possibility is that $\mathcal{A}$ succeeds to generate the correct form $\rho$. Obviously, $p_0$ is restricted by two factors. The first factor is that adversary $\mathcal{A}$ guesses the correct password $\pi$ and the adversary $\mathcal{A}$ has no knowledge of $\epsilon$. Another possibility is that $\mathcal{A}$ guesses the correct $\eta$ with probability $1/2^k$ since $\mathcal{A}$ has no knowledge of $\rho$. The probability of this case is $1/|\mathcal{H}_1|$. The second factor is that adversary $\mathcal{A}$ has to generate the correct form $(h, s)$ containing $\alpha$ and recover the message $\epsilon$ from $(s_1, s_2)$. As analyzed before, the probability in this situation is bounded by $n_{\text{exe}}/2^k + \text{Adv}_2^{\mathcal{H}_2} (\mathcal{O}(t)) + \text{Adv}^{\mathcal{H}_2} (\mathcal{O}(t))$. We use $n_{\text{exe}}$ to denote the number of this kind of queries. Thus, the success probability of $\mathcal{A}$ in this case is bounded by

$$\Pr(\text{Succ}_{\text{Case1}}) \leq n_{\text{exe}} \left( \frac{1}{|\mathcal{H}_2|} + \frac{n_{\text{ro}}}{2^k} + \text{Adv}_2^{\mathcal{H}_2} (\mathcal{O}(t)) + \text{Adv}^{\mathcal{H}_2} (\mathcal{O}(t)) + \frac{1}{2^k} \right).$$

**Case 2.** Assume that instance $\pi_2'$ receives a message $(A, h, s, R)$ generated from adversary $\mathcal{A}$. After receiving the message, instance $\pi_2'$ generates $(r_1, r_2) = \delta_{\alpha} (F_2)$, queries the random oracle $\mathcal{H}_2$ on $(r_1, r_2)$, and computes $e = f_w (\mathcal{H}_2 (r_1, r_2))$. Next, $\Pi_2'$ randomly generates $\epsilon \in \mathcal{S}_{w, \alpha} (F_2)$. Then, $\Pi_2'$ queries the random oracle $\mathcal{H}_1$ on $(\pi, R, A, B, h)$. The reply of $\mathcal{H}_1$ is denoted by $\alpha$. Finally, $\Pi_2'$ computes $s_1 = r_1 + h r_2$ and $s_2 = (s - \alpha) r_2 + e + \epsilon$ and returns $(B, s_1, s_2)$ to $\mathcal{A}$. In this situation, $\mathcal{A}$ has to generate a $\mu$, which is equal to the output from random oracle $\mathcal{H}_1$ on $(\epsilon, R, A, B, h)$. Similar to the previous case, one possibility is that $\mathcal{A}$ guesses the correct $\eta$ with probability $1/2^k$ since $\mathcal{A}$ has no knowledge of $\epsilon$. Another possibility is that $\mathcal{A}$ guesses the correct password $\pi$. We use $n_{\text{exe}}$ to denote the number of this kind of queries. Thus, the success probability of $\mathcal{A}$ in this case is bounded by

$$\Pr(\text{Succ}_{\text{Case2}}) \leq n_{\text{exe}} \left( \frac{1}{|\mathcal{H}_2|} + \frac{1}{2^k} \right).$$

Hence, we get

$$\Pr(\text{Succ}_{\text{Case2}}) \leq n_{\text{exe}} \left( \frac{1}{|\mathcal{H}_2|} + \frac{1}{2^k} \right).$$

Therefore, we have

$$\text{Adv}(\mathcal{A}, P_1) = 2 \Pr(\text{Succ}_{P_1}) - 1 \leq \frac{2 n_{\text{exe}}}{|\mathcal{H}_2|} + \frac{2 n_{\text{exe}}}{2^k} + 2 n_{\text{exe}} \text{Adv}_2^{\mathcal{H}_2} (\mathcal{O}(t)) + 2 n_{\text{exe}} \text{Adv}^{\mathcal{H}_2} (\mathcal{O}(t)) + \frac{2 n_{\text{exe}} n_{\text{ro}}}{2^k}.$$

By combining all the claims, we have the advantage for the adversary $\mathcal{A}$ in the real attack, i.e.,

$$\text{Adv}^{\text{OPAKE}}(\mathcal{A}) \leq n_{\text{exe}} \left( \frac{1}{|\mathcal{H}_2|} + \mathcal{O} (n_{\text{exe}} + n_{\text{exe}}) \left( \text{Adv}_2^{\mathcal{H}_2} (\mathcal{O}(t)) + \text{Adv}^{\mathcal{H}_2} (\mathcal{O}(t)) \right) + \left( n_{\text{exe}} + n_{\text{exe}} \right) \frac{n_{\text{ro}}}{2^k} + \frac{n_{\text{exe}}}{2^k} \right).$$

---

**Table 1: Time consumption of different security levels of OPAKE.**

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>w</th>
<th>u</th>
<th>t</th>
<th>k</th>
<th>Security</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPAKE-80</td>
<td>5851</td>
<td>47</td>
<td>47</td>
<td>30</td>
<td>512</td>
<td>80</td>
<td>6.41</td>
</tr>
<tr>
<td>OPAKE-128</td>
<td>13691</td>
<td>75</td>
<td>75</td>
<td>45</td>
<td>512</td>
<td>128</td>
<td>7.53</td>
</tr>
<tr>
<td>OPAKE-256</td>
<td>40013</td>
<td>147</td>
<td>147</td>
<td>85</td>
<td>512</td>
<td>256</td>
<td>9.14</td>
</tr>
</tbody>
</table>
Hence, Theorem 2 is desired.

6. Performance Analysis

In this section, we provide the efficiency evaluation of our scheme. The security parameters we used are adopted from the Ouroboros scheme[29]. The scheme implementation is written in C++. The program has been performed on a computer running Linux. The computer has an Intel Core i7-8750H CPU@2.20 GHz and 4 GB of memory. For each parameter set, the performance of our proposed protocol is shown in Table 1. The results show that the efficiency of our scheme is considerable.

7. Conclusion

In this study, we propose a new password-authenticated key exchange based on Ouroboros key exchange scheme with formal security proof. Our PAKE scheme enjoys several desired properties, including (1) our scheme has considerable efficiency; (2) our scheme provides mutual explicit authentication; and (3) our scheme is resistant to quantum attacks and an attacker who gets the session key cannot use it to perform an offline dictionary attack. The security of our scheme is reduced to decoding random quasi-cyclic codes in the random oracle model.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

Security and Communication Networks


