Provably Secure Password Reset Protocol: Model, Definition, and Generic Construction

Satsuya Ohata[1,2], Takahiro Matsuda[2], and Kanta Matsuura[1]

[1] The University of Tokyo, Tokyo, Japan. {satsuya,kanta}@iis.u-tokyo.ac.jp
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan. t-matsuda@aist.go.jp

Abstract. Many online services adopt a password-based user authentication system because of its usability. However, several problems have been pointed out on it, and one of the well-known problems is that a user forgets his/her password and cannot login the services. To solve this problem, most online services support a mechanism with which a user can reset a password. In this paper, we consider a provable security treatment for a password reset protocol. We formalize a model and security definitions, propose a generic construction based on a pseudorandom function and public key encryption. In addition, we implement a prototype of our protocol to evaluate its efficiency.

Keywords: Password Reset Protocol, Provable Security.

1 Introduction

1.1 Background and Motivation

User authentication is one of the fundamental research themes not only in theory but also in practice. Although there are some unsolved problems, password-based user authentication systems are widely used in practice because of their usability and some other advantages. One of the unsolved problems is that users may forget their passwords, and this problem cannot be avoided as long as the user is a human. One may expect that we can overcome this problem by alternatives such as graphical password [30], biometrics [18], etc. However, these authentication methods also need backup authentication systems when users are unable to perform the primary authentication due to operational errors, physical problems, etc. To make matters worse, they have their own problems. Their problems in primary authentication may include insufficient entropy of biometric information, wolves/lambs [12], and the initial/operational cost of additional devices. Their problems in backup authentication can be even more complicated. For example, if an officer can attend on users at authentication devices, some of the problems in the primary authentication may be solved promptly (e.g. operational errors) but others may stay still hard (e.g. severe physical problems such as a burn on a finger in biometrics). It should be noted that relying on such an officer costs a lot, and may cause some other problems (e.g. privacy problems).

A potential advantage of password-based authentication is a clear reset scenario in backup authentication: if a password which is valid in the next execution of the primary authentication can be securely delivered to a user who is unable to perform the primary authentication, the user can be rescued by the delivered password. Thus we reach our main research question in this paper: *can we design a provably secure protocol for this reset (namely, a provably secure password reset protocol)?*

In existing websites which rely on password-based authentication, backup authentication for reissuing a valid password often uses personal information (e.g. e-mail address, answers to secret questions, etc.) which was provided by the user in the initial registration procedure. Although such backup authentication mechanisms are convenient for users, their security relies on nothing but heuristic evaluation or intuition. Even in the case of a provably secure primary authentication protocol such as PAKE (password-authenticated key exchange) [5, 2, 8, 21], its provable security is meaningless in practice if it is used with a heuristic (and hence, potentially weak) password reset protocol. Bonneau et al. [7], in fact, showed that the security of the secret question (which is a popular backup authentication mechanism) is not enough. In order to prevent the cat-and-mouse game between attacks and heuristic countermeasures completely, not only primary authentication but also backup authentication should be provably secure.

Inspired by the above views, we investigate provably secure password reset protocols in this paper. In contrast to the existing usable security papers regarding backup authentication [31, 20, 17, 24, 26, 27, 25, 19, 16], we follow the provable security approach in cryptography.

We firstly define a model, then provide security definitions of a password reset protocol, and finally show provably secure protocols. In particular, we propose generic constructions of a provably secure password reset protocol based on a pseudorandom function and public key encryption.

The Difficulty of Model Design In a password reset protocol, it is demanded that a user can re-register a refreshed password under the situation that the user does not have his/her password. However, in an authentication system that uses only user's identity and password which are provided by the user in the initial registration procedure, a legitimate user who forgets his/her password and an adversary are indistinguishable because the former does not have a unique information. Therefore, when we consider a password reset protocol, we have to adopt a different model from standard user authentication systems. We will easily be able to construct a provably secure password reset protocol by assuming the existence of a trusted third party (TTP) in the same way as the case of applied cryptosystems such as identity-based encryption (IBE) [28, 6]. However, as the fact that mass surveillance has been carried out by NSA becomes clear now, we would like to avoid assuming such a big brother. A security definition should capture the real world, and be achievable by a practical protocol (so that we need not require an unrealistic assumption such as a secure channel between a user and a server with which a refreshed password can always be securely delivered).

1.2 Our Contribution

In this paper, we consider a provably secure password reset protocol, formalize a model and security definitions, propose a construction, prove its security, and show the efficiency of our protocol via prototype implementation. Because of the difficulty that we point out in Section 1.1, we have to consider a different model from standard user authentication systems to construct a provably secure password reset protocol. In this paper, we propose a model that introduces a key for password reset. In our model, the system generates a reset key in the initial registration procedure. When a user wants to register/reset his/her password, he/she uses this key in the password registration/reset phase. We assume that the reset key is securely stored, and the user does not lose the reset key even if the user forgets his/her password. We discuss the validity of this assumption in Section 1.3. In our model, a user can choose both primary and refreshed passwords. Moreover, our model also captures the leakage of a password. In a standard password-based authentication, if an adversary steals a password of a legitimate user and changes the inside states of the authentication system (e.g. password, information for backup authentication, etc.), the legitimate user cannot take back his/her user account. However, in our model, a user who had his/her password stolen and lost the user account can take back it by using his/her reset key which is securely stored. To the best of our knowledge, none of the previous user authentication protocols can realize this property.

First, in this paper, we formalize a model and security definitions of a password reset protocol. For simplicity, first, we consider security against passive adversaries. In this security definition, an adversary is allowed to get all the information by observing the transcripts between a user and the server. After that (in Section 5.2), we also consider security against active adversaries that can mount man-in-the-middle attacks and concurrent attacks by extending the security definitions for passive adversaries. Then we propose a generic construction based on a pseudorandom function and public key encryption. The security that we require for these building blocks is popular one, and a number of concrete schemes that satisfy our requirements have been already known. Therefore, we can construct many efficient and simple concrete password reset protocols from this generic construction. Finally, we evaluate the performance of our protocol by implementing a prototype. The result shows that our protocol has good efficiency, and can be used in practice.

1.3 Introduction of Reset Key

In our model, we introduce a special key for password reset, which we call a reset key, and assume that it is securely stored. Indeed, this is relatively a strong assumption. Here, we explain the reasons why we adopt this methodology and how this assumption is reasonable.

1. This assumption is similar to the setting of key-insulated cryptography [13], where a secret key is updatable by a higher-level secret key that is assumed to be securely stored. We emphasize that this is not only widely accepted in public-key cryptography, but its potential feasibility was pointed out in practical security research (e.g. [23, 14]). One may think that if we allow these assumptions, then we may as well

assume that a user does not lose his/her password. However, as considered in key-insulated cryptography, it is (to some extent) reasonable to assume that the user does not lose the reset key because the reset key is only required in the case of emergency and can therefore be securely stored. One may also think that this assumption does not hold in the system where a user is forced to change his/her password frequently. However, the assumption still holds because the user who remembers his/her password can change his/her password in the system without using a reset key.

2. There exist security protocols in the real world which work as (a valiant of) the key-insulated setting. For example, widely used hardware security tokens have a long-term key in itself and generate a one-time password by using it. If this long-time key is revealed, we can compute a one-time password. We can consider that the security of these tokens is assured in the key-insulated model.
3. Our model (to some extent) captures the implementations of password reset protocols used in the real world. In Facebook, for example, information to reset a password is sent to a user's email address registered in the initial registration procedure. Here, the user's password to log-in to the email account can be seen as a reset key. Our model captures an intuition in such real-world protocols, and provides provable security simultaneously.
4. The problem that we have to solve this time is the one that we do not really care in the research of cryptography. For example, in the case of IBE, TTP receives a user's ID (e.g. user's email address) and derives a corresponding secret key. However, the framework of IBE does not offer how to distinguish whether the user really has that ID or not. In order to use IBE in practice, we have to rely on an infrastructure other than IBE. An easygoing way to solve this problem is, for instance, to send a derived secret key to a user's email address. In this example, we can think of the password to log-in to the email account as a reset key. This is almost the same case as the above example of account recovery in Facebook.

Even if we introduce a reset key in the password reset protocol, it does not mean that the construction or security proof becomes trivial. Intuitively, we can achieve the password reset functionality by using a symmetric key encryption (SKE). User regards a reset key as a key of SKE, encrypts a new password by using a reset key, and sends a ciphertext to the server. However, this simple method is vulnerable to a replay attack. Moreover, this SKE-based construction seems to fail to satisfy the security definition which is described in Section 3.2. We will explain the reason of this in Section 4.

1.4 Paper Organization

The remainder of this paper is organized as follows. In Section 2, we review the basic notation and the definitions of cryptographic primitives used in this paper. In Section 3, we introduce the models and security definitions for a password reset protocol. In addition, we explain how this protocol is used in practice. In Section 4, we present our generic construction of a password reset protocol and prove its security. In Section 5, we discuss the extension of our proposed construction. In Section 6, we show the performance evaluation of our protocol. Section 7 is the conclusion.

2 Preliminaries

In this section, we review the basic notation, and the models and security definitions of a pseudorandom function and public key encryption.

2.1 Notations

$\mathbf{N}$ denotes the set of all natural numbers. "$x \leftarrow y$" denotes that x is chosen uniformly at random from y if y is a finite set, x is output from y if y is a function or an algorithm. "PPT" stands for *probabilistic polynomial time*. If $\mathcal{A}$ is a probabilistic algorithm then "$y \leftarrow \mathcal{A}(x)$" denotes that $\mathcal{A}$ computes y as output by taking x as input. If $\mathcal{A}$ is a protocol between interactive algorithms P and Q, we write "$(p_{out}, q_{out}) \leftarrow \mathcal{A}(P(p_{in}) \leftrightarrow Q(q_{in}))$" to mean that P and Q take p_{in} and q_{in} as input, respectively, interact with each other, and finally P and Q locally output p_{out} and q_{out}, respectively. "$x := y$" denotes that x is defined as y. "ϕ" denotes the empty string. "ε" denotes a negligible function. "k" always denotes the security parameter.

2.2 Pseudorandom Function

We say that $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ is a pseudorandom function (PRF) if

1. there exists an efficient algorithm that takes a key $K \in \{0,1\}^k$ and a string $x \in \{0,1\}^*$ as input, and outputs $F(K,x)$.
2. for all PPT adversaries $\mathcal{A}$, the advantage in the following PRF game played with the challenger $\mathcal{B}$ is negligible: First, $\mathcal{B}$ picks the challenge bit $b \in \{0,1\}$ and a key $K \in \{0,1\}^k$ uniformly at random. $\mathcal{A}$ can adaptively make queries x (without loss of generality, we assume that $\mathcal{A}$ does not make the same query twice). If $b = 0$, then $\mathcal{B}$ responds with $F(K,x)$. Otherwise (that is, $b = 1$), $\mathcal{B}$ returns a random string in the range of F. Finally, $\mathcal{A}$ outputs its guess bit b' for b. We define the advantage of $\mathcal{A}$ in this game by $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A}}(k) := |\Pr[b' = b] - 1/2|$.

2.3 Public Key Encryption

A public key encryption (PKE) scheme consists of the following three PPT algorithms $(\mathsf{PKG}, \mathsf{PEnc}, \mathsf{PDec})$.

PKG This is the key generation algorithm that takes 1^k as input, and outputs a pair of public key pk and decryption key dk. This process is written as $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$.

PEnc This is the encryption algorithm that takes a public key pk and a plaintext m as input, and outputs a ciphertext c. This process is written as $c \leftarrow \mathsf{PEnc}(pk, m)$.

PDec This is the (deterministic) decryption algorithm that takes a decryption key dk and a ciphertext c as input, and outputs a decryption result m (which could be the special symbol $\perp$ meaning that c is invalid). This process is written as $m \leftarrow \mathsf{PDec}(dk, c)$.

We require the standard correctness for a PKE scheme. Namely, for any $k \in \mathbf{N}$, any $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$, and any plaintext m, we have $m = \mathsf{PDec}(dk, \mathsf{PEnc}(pk, m))$.

Security Definition We recall the definition of indistinguishability against multi-challenge chosen ciphertext attacks (mIND-CCA, for short) [1] of PKE, which is defined by the following game between the challenger $\mathcal{B}$ and an adversary $\mathcal{A}$: First, $\mathcal{B}$ picks the challenge bit $b \in \{0,1\}$, computes $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$, and gives pk to $\mathcal{A}$. $\mathcal{A}$ can adaptively make encryption and decryption queries. For an encryption query (m_0, m_1), where (m_0, m_1) is a message pair of equal length, $\mathcal{B}$ computes $c^* \leftarrow \mathsf{PEnc}(pk, m_b)$, and then returns the ciphertext c^* to $\mathcal{A}$. For a decryption query c, $\mathcal{B}$ responds with $m \leftarrow \mathsf{PDec}(dk, c)$, except that if c is one of the challenge ciphertexts c^* returned to $\mathcal{A}$ as a response to an encryption query, then the challenger returns $\perp$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a guess bit b' for b. $\mathcal{A}$ wins the game if $b = b'$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{\mathsf{mINDCCA\text{-}PKE}}_{\mathcal{A}}(k) = |\Pr[b' = b] - 1/2|$. We say a PKE scheme is mIND-CCA secure, if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{mINDCCA\text{-}PKE}}_{\mathcal{A}}(k)$ is negligible. It was shown that ordinary IND-CCA security and multi-challenge IND-CCA security are polynomially equivalent [1].

3 Password Reset Protocol

In this section, we present the model and the security definitions of a password reset protocol. The proposed model and the security definitions (which are described in Section 3.2) are an extension of the existing model and the security definition for identity-based identification [22, 4]. We assume that a reset key is securely distributed to the client in some way. We also assume that the reset key is securely stored, and the client does not lose the reset key even if the client forgets his/her password as assumed in key-insulated cryptography [13]. One may think that if we allow these assumptions, then we may as well assume that a client does not lose his/her password. As we discussed in Section 1.3, however, it is (to some extent) reasonable to assume that the client does not lose the reset key.

3.1 Model

A password reset protocol consists of the following two PPT algorithms ($\mathsf{SSetup}, \mathsf{RKG}$) and two subprotocols $\mathsf{PRR}(C_P(\cdot) \leftrightarrow S_P(\cdot))$ and $\mathsf{Auth}(C_A(\cdot) \leftrightarrow S_A(\cdot))$. Let $\mathcal{PW}$ denote the password space.

SSetup This is the server setup algorithm that takes 1^k as input, and outputs a public parameter pp and a secret key sk. This process is written as $(pp, sk) \leftarrow \mathsf{SSetup}(1^k)$.

RKG This is the reset key generation algorithm that takes a secret key sk and a client's identity $ID \in \{0,1\}^*$ as input, and outputs a reset key rk. This process is written as $rk \leftarrow \mathsf{RKG}(sk, ID)$.

PRR This is the interactive protocol for password (re-)registration between the PPT algorithms C_P and S_P. Let the identity of a client that runs the algorithm C_P be ID. The algorithm C_P takes the identity ID, a password $pw \in \mathcal{PW}$, and a reset key rk as input, the algorithm S_P takes the identity ID, a reset key rk, and a secret key sk as input, and then these algorithms interact with each other. As a result of the interaction, C_P and S_P locally output ϕ and pw_s (which could be $\perp$), respectively[3]. This process is written as $(\phi, pw_s) \leftarrow \mathsf{PRR}(C_P(ID, pw, rk) \leftrightarrow S_P(ID, rk, sk))$[4].

Auth This is the interactive authentication protocol between the PPT algorithms C_A and S_A. Let the identity of a client that runs the algorithm C_A be ID. The algorithm C_A takes the identity ID and a password $pw \in \mathcal{PW}$ as input, the algorithm S_A takes the identity ID, pw_s, and a secret key sk as input, and then these algorithms interact with each other. As a result of the interaction, C_A and S_A locally output ϕ and $\top/\perp$, respectively, where $\top$ (resp. $\perp$) means "accept" (resp. "reject"). This process is written as $(\phi, \top/\perp) \leftarrow \mathsf{Auth}(C_A(ID, pw) \leftrightarrow S_A(ID, pw_s, sk))$[4].

Correctness We require the following correctness property for a password reset protocol. For any $(pp, sk) \leftarrow \mathsf{SSetup}(1^k)$, any $pw \in \mathcal{PW}$, any $ID \in \{0,1\}^*$, any $rk \leftarrow \mathsf{RKG}(sk, ID)$, any $(\phi, pw_s) \leftarrow \mathsf{PRR}(C_P(ID, pw, rk) \leftrightarrow S_P(ID, rk, sk))$, we have $(\phi, \top) \leftarrow \mathsf{Auth}(C_A(ID, pw) \leftrightarrow S_A(ID, pw_s, sk))$.

Remark Here, we explain how to use this protocol in practice. First, the server executes SSetup and generates a public parameter pp and a secret key sk. In the initial registration procedure, the server runs RKG and generates a reset key rk for each client. When a client registers an initial password, a client and the server execute PRR interactively, and a client registers an initial password pw. In this PRR protocol, the client executes the algorithm C_P, and the server executes the algorithm S_P. A client authenticates himself/herself by using an ID and a pw in Auth protocol. In this Auth protocol, a client executes the algorithm C_A, and the server executes the algorithm S_A. When a client forgets his/her password pw, a client and the server execute PRR interactively, and the client re-registers a refreshed password pw'. In this PRR protocol, the client executes the algorithm C_P, and the server executes the algorithm S_P.

3.2 Security Definitions

In this subsection, we give the formal security definitions of a password reset protocol and explain what situations our definitions capture. First, we consider passive attacks in this section. Later (in Section 5.2), we also consider active attacks. The passive attack captures the situation in which an adversary observes the transactions between a client and the server from outside. In our security definitions, an adversary can get all the information by observing the transactions between a client and the server. Here, we have to consider the two types of security for a password reset protocol. The first one is that an adversary who does not have a correct password cannot pass the authentication. We call it *security against impersonation attacks*. The second one is that an adversary who does not have a correct reset key cannot (re-)register a password. We call it *security against illegal registration attacks*. Let $\mathcal{D}$ be the dictionary of user's password.

[3] This pw_s denotes the information that is supposed to be stored in the server and is used when the user with password pw next time requests authentication of him/her. We do not require $pw_s = pw$ hold in general. (See, e.g. the construction in Section 5.1.)

[4] The algorithms S_P and S_A of our proposed scheme need not use a secret key sk. In general, however, S_P and S_A are allowed to use sk. In our extended password reset protocol in Section 5.1, in fact, we need sk in both of the algorithms S_P and S_A.

Impersonation First, we consider security against impersonation under passive attacks (Imp-PA) for a password reset protocol. This security is defined using the following Imp-PA game which is played by the challenger $\mathcal{B}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. First, $\mathcal{B}$ executes $(pp, sk) \leftarrow \mathsf{SSetup}(1^k)$, and generates an empty list L into which tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$ where $flag_p, flag_r \in \{0, 1\}$ will be stored. These $flag_p$ and $flag_r$ are used to indicate whether a client with ID is "corrupted" by $\mathcal{A}$ in the sense that either pw or rk is known to $\mathcal{A}$, in which case $\mathcal{A}$ is not allowed to use the ID for its attack. After key generation, $\mathcal{B}$ gives pp to $\mathcal{A}_1$. Then $\mathcal{A}_1$ can adaptively make the following types of queries.

Client create query ($\mathsf{CCreate}$)**:** On input $ID \in \{0, 1\}^*$, $\mathcal{B}$ responds as follows. If there exists a tuple of the form $(ID, *, *, *, *, *)$ in the list L, $\mathcal{B}$ does nothing. Otherwise, $\mathcal{B}$ executes $rk \leftarrow \mathsf{RKG}(sk, ID)$, and stores $(ID, \phi, \phi, rk, 0, 1)$ into the list L. If $\mathcal{A}_1$ makes the following queries ($\mathsf{PR}, \mathsf{RKR}, \mathsf{Trans}_{\mathsf{PRR}}$, and $\mathsf{Trans}_{\mathsf{Auth}}$) with an identity ID, then this ID must have appeared as a $\mathsf{CCreate}$ query (and thus be stored in the list L).

Password reveal query (PR)**:** On input ID, $\mathcal{B}$ finds the tuple of the form $(ID, pw, *, *, *, *)$ in the list L, and returns pw to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, 0, *)$.

Reset key reveal query (RKR)**:** On input ID, $\mathcal{B}$ finds the tuple of the form $(ID, *, *, rk, *, *)$ in the list L, and returns rk to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, rk, *, 0)$.

Password (re-)registration transcript query ($\mathsf{Trans}_{\mathsf{PRR}}$)**:** On input (ID, pw'), $\mathcal{B}$ responds as follows.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ first finds the tuple $(ID, *, *, rk, *, *)$ in the list L, executes $(\phi, pw'_s) \leftarrow \mathsf{PRR}(C_P(ID, pw', rk) \leftrightarrow S_P(ID, rk, sk))$, and returns the transcript $trans_{\mathsf{PRR}}$ of PRR and the result of registration $\top/\bot$ to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw'_s, *, 0, *)$.
2. If $pw' = \phi$, $\mathcal{B}$ first chooses a random password $pw' \in \mathcal{D}$. Next, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L, executes $(\phi, pw'_s) \leftarrow \mathsf{PRR}(C_P(ID, pw', rk) \leftrightarrow S_P(ID, rk, sk))$, and returns the transcript $trans_{\mathsf{PRR}}$ of PRR and the result of registration $\top/\bot$ to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw'_s, *, 1, *)$.

Authentication transcript query ($\mathsf{Trans}_{\mathsf{Auth}}$)**:** On input (ID, pw'), $\mathcal{B}$ responds as follows.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ first finds the tuple $(ID, *, pw_s, *, *, *)$ in the list L, executes $\mathsf{Auth}(C_A(ID, pw') \leftrightarrow S_A(ID, pw_s, sk))$, and returns the transcript $trans_{\mathsf{Auth}}$ of Auth and the result of authentication $\top/\bot$ to $\mathcal{A}_1$.
2. If $pw' = \phi$, $\mathcal{B}$ first finds the tuple $(ID, pw, pw_s, *, *, *)$ in the list L, executes $\mathsf{Auth}(C_A(ID, pw) \leftrightarrow S_A(ID, pw_s, sk))$, and returns the transcript $trans_{\mathsf{Auth}}$ of Auth and the result of authentication $\top/\bot$ to $\mathcal{A}_1$.

Finally, $\mathcal{A}_1$ outputs (ID^*, st). To win the Imp-PA game, the tuple $(ID^*, pw^*, pw^*_s, rk^*, flag^*_p, flag^*_r)$ must exist in the list L and satisfy $flag^*_p = 1$ and $flag^*_r = 1$ (if this is satisfied, we say that ID^* satisfies the "winning precondition"). If these conditions are not satisfied, $\mathcal{B}$ decides that $\mathcal{A}$ has lost the Imp-PA game. Otherwise, $\mathcal{B}$ gives st to $\mathcal{A}_2$. Then $\mathcal{A}_2$ and $\mathcal{B}$ interactively execute $\mathsf{Auth}(\mathcal{A}_2(st) \leftrightarrow S_A(ID^*, pw^*_s, sk))$. During the execution of this Auth protocol, $\mathcal{A}_2$ can adaptively make queries in the same way as $\mathcal{A}_1$. However, $\mathcal{A}_2$ is not allowed to use ID^* in the PR, RKR, and $\mathsf{Trans}_{\mathsf{PRR}}$ queries. Finally, $\mathcal{A}$ wins if the S_A's output of Auth is $\top$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{\mathsf{Imp\text{-}PA}}_{\mathcal{A}}(k) = \Pr[\mathcal{A}\ wins]$.

Definition 1. *Let q_A be the number of $\mathsf{Trans}_{\mathsf{Auth}}$ queries by $\mathcal{A}_1$. We say that a password reset protocol is Imp-PA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{Imp\text{-}PA}}_{\mathcal{A}}(k) = O(q_A)/|\mathcal{D}| + \varepsilon(k)$.*

Illegal Registration Second, we consider security against illegal registration under passive attacks (IR-PA) for a password reset protocol. This security is defined using the following IR-PA game which is played by the challenger $\mathcal{B}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{B}$'s initial procedure and $\mathcal{A}_1$'s queries of this IR-PA game are exactly the same as the Imp-PA game. Finally, $\mathcal{A}_1$ outputs (ID^*, st). To win the IR-PA game, ID^* must satisfy the winning precondition. If these conditions are not satisfied, $\mathcal{B}$ decides that $\mathcal{A}$ has lost the IR-PA game. Otherwise, $\mathcal{B}$ gives st to $\mathcal{A}_2$. Then $\mathcal{A}_2$ and $\mathcal{B}$ interactively execute $\mathsf{PRR}(\mathcal{A}_2(st) \leftrightarrow S_P(ID^*, rk^*, sk))$. During the execution of this PRR protocol, $\mathcal{A}_2$ can adaptively make the queries in the same way as $\mathcal{A}_1$. However, $\mathcal{A}_2$ is not allowed to use ID^* in the PR, RKR, and $\mathsf{Trans}_{\mathsf{PRR}}$ queries. Finally, $\mathcal{A}$ wins if S_P's output of PRR is different from $\bot$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{\mathsf{IR\text{-}PA}}_{\mathcal{A}}(k) = \Pr[\mathcal{A}\ wins]$.

Definition 2. *We say that a password reset protocol is IR-PA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{IR\text{-}PA}}_{\mathcal{A}}(k)$ is negligible.*

Remark Here we explain what situations PR, RKR, $\mathsf{Trans}_{\mathsf{PRR}}$, and $\mathsf{Trans}_{\mathsf{Auth}}$ queries allowed for an adversary try to capture.

An adversary may register the authentication system as a legitimate user and learn which ID/pw is weak, which is captured by PR and RKR queries. Furthermore, the adversary may eavesdrop the communication between honest users and the server to obtain all the transcripts. This is captured by $\mathsf{Trans}_{\mathsf{PRR}}$ and $\mathsf{Trans}_{\mathsf{Auth}}$ queries. Note that for both types of queries, the challenger behaves differently depending on whether pw' in the adversary's input is ϕ or not. The former case ($pw' = \phi$) captures the situation where the protocols are run by honest users.; The latter case ($pw' \in \mathcal{PW}$) has different meanings depending on the types of queries. A $\mathsf{Trans}_{\mathsf{PRR}}$ query with $pw' \in \mathcal{PW}$ captures the situation where the adversary itself tries to learn information from the transcript of the PRR protocol by executing it honestly, using the adversarially chosen password pw'. We may also be able to think of it as capturing the situation where an honest user registers a password that is known to an adversary for some reason (e.g. because of the choice of an easy-to-guess password).; A $\mathsf{Trans}_{\mathsf{Auth}}$ query with $pw' \in \mathcal{PW}$ captures the case where again the adversary itself tries to learn information from the transcript of Auth protocol by executing it honestly, using the (adversarially chosen) password pw'. It also in some sense captures the situation where an honest user forgets (or mistypes) his/her password.

4 Proposed Construction

In this section, we propose a generic construction of a password reset protocol which satisfies the security definitions in Section 3.2.

4.1 Construction

In this subsection, we show a generic construction of a password reset protocol based on a PRF and PKE.

Intuition Before showing our construction, we explain an intuition of our proposed protocol. In our protocol, we set a reset key $rk := F(K, ID)$. In the password reset procedure, a user encrypts a randomness (which is sent by the server), a reset key, and a password by using a PKE scheme, and sends a ciphertext to the server. The server decrypts the ciphertext and gets a password. The authentication procedure is very similar to the password reset procedure. A user encrypts a randomness (which is sent by the server) and a password by using a PKE scheme, and sends a ciphertext to the server. The server decrypts the ciphertext, and checks whether the decryption result matches the registered password or not. The security of PKE (and the PRF) ensures that the transcripts do not leak the information of the passwords of honest users, and hence an adversary who wants to impersonate an uncorrupted user (with an unknown password) essentially has to guess the password. In both PRR and Auth protocols, the randomness chosen by the server prevents "replay" attacks.

How Straightforward SKE-based Construction May Fail Suppose we want to reduce the security of a password reset protocol to the security of SKE. Recall that in our Imp-PA security game, an adversary $\mathcal{A}$ is allowed to issue reset key reveal (RKR) queries for ID's that are not the challenge ID^*. This means that the reduction algorithm cannot embed its problem instance regarding SKE into the reset keys of users who will be corrupted. Therefore, the most natural approach for designing the reduction algorithm (that attacks the security of SKE) using an adversary $\mathcal{A}$ against the Imp-PA security of a considered password reset protocol, will be to guess the index ℓ of $\mathcal{A}$'s $\mathsf{CCreate}$ query (for which $\mathcal{A}$ specifies ID^*) and embed the instance of the reduction algorithm's problem into the challenge user's ID^*. However, once we do this, the number of CCreate queries appears in the numerator of the formula of the advantage, namely, what we will be able to show is something like $\mathsf{Adv}^{\mathsf{Imp\text{-}PA}}_{\mathcal{A}} \leq O(q_C q_A)/|\mathcal{D}| + \varepsilon(k)$, where q_C is the number of $\mathcal{A}$'s $\mathsf{CCreate}$ queries. However, Imp-PA security requires that $\mathsf{Adv}^{\mathsf{Imp\text{-}PA}}_{\mathcal{A}}$ is upper-bounded by $O(q_A)/|\mathcal{D}| + \varepsilon(k)$, and thus the straightforward approach using SKE is not sufficient for proving Imp-PA security.

Although there could exist a way to avoid this problem by developing a new proof strategy, we try to solve this problem by using PKE scheme in this paper.

Our Construction Now, we formally describe the password reset protocol. Let $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ be a pseudorandom function and $(\mathsf{PKG}, \mathsf{PEnc}, \mathsf{PDec})$ be a PKE scheme. Using these as building blocks, our password reset protocol is constructed as in Fig. 1.

$\mathsf{SSetup}(1^k)$: $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ Choose a random $K \in \{0,1\}^k$ $pp := pk$; $sk := (K, dk)$ return (pp, sk).
$\mathsf{RKG}(sk, ID)$: $(K, dk) \leftarrow sk$ $rk := F(K, ID)$ return rk.
$(\phi, pw_s) \leftarrow \mathsf{PRR}(C_P(ID, pw, rk) \leftrightarrow S_P(ID, rk, sk))$: 1. S_P chooses a randomness $r \in \{0,1\}^k$ and sends it to C_P 2. C_P executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw)$ and sends it to S_P 3-1. S_P executes $ID'\|r'\|rk'\|pw_s \leftarrow \mathsf{PDec}(dk, c)$ 3-2. If $ID' = ID$, $r' = r$, and $rk' = rk$ hold, S_P returns pw_s else S_P returns $\perp$.
$(\phi, \top/\perp) \leftarrow \mathsf{Auth}(C_A(ID, pw) \leftrightarrow S_A(ID, pw_s, sk))$: 1. S_A chooses a randomness $r \in \{0,1\}^k$ and sends it to C_A 2. C_A executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$ and sends it to S_A 3-1. S_A executes $ID'\|r'\|pw' \leftarrow \mathsf{PDec}(dk, c)$ 3-2. If $ID' = ID$, $r' = r$, and $pw' = pw_s$ hold, S_A returns $\top$ else S_A returns $\perp$.

Fig. 1. The proposed generic construction of a password reset protocol

4.2 Security Proof

In this subsection, we show security proofs of the proposed password reset protocol in Fig. 1.

Theorem 1. *If F is a PRF and the PKE scheme is mIND-CCA secure*[5]*, then the proposed password reset protocol in Fig. 1 satisfies Imp-PA security.*

Proof of Theorem 1. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an Imp-PA adversary of the password reset protocol, q_A be the number of $\mathsf{Trans}_{\mathsf{Auth}}$ queries by $\mathcal{A}_1$. Here, q_A is a polynomial of the security parameter k. Consider the following sequence of games.

Game 0. This is exactly the Imp-PA game.

Game 1. This game proceeds in the same way as Game 0, except that the first messages r of S_A picked in the executions of PRR (in the response to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{PRR}}$ query) and Auth (either in the response to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{Auth}}$ query or in the challenge phase), are picked from $\{0,1\}^k \backslash \{r$'s that are already used$\}$, so that they are all distinct and never collide. For notational convenience, in this and subsequent games, we introduce the list $\mathcal{R}$ that is used to store r's that are used in the response to the $\mathsf{Trans}_{\mathsf{PRR}}$ query, $\mathsf{Trans}_{\mathsf{Auth}}$ query, and in the execution of Auth in the challenge phase, and we make the challenger choose r uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$ every time it needs to choose r for PRR and Auth, and put the used r into the list $\mathcal{R}$.

Game 2. This game proceeds in the same way as Game 1, except that if $\mathcal{A}_1$ issues RKG queries on ID, then instead of using the result of $F(K, ID)$, $\mathcal{B}$ picks rk uniformly at random from the range of F, and uses it as the reset key corresponding to ID.

Game 3. This game proceeds in the same way as Game 2, except that When $\mathcal{B}$ calculates the output of S_A (i.e. $\top/\perp$) only by checking $pw = pw_s$ where pw_s is the value found in the tuple corresponding to ID in the list L, without running PDec.

Game 4. This game proceeds in the same way as Game 3, except for the following points.

- If $\mathcal{A}_1$ issues a $\mathsf{Trans}_{\mathsf{PRR}}$ query on ID, then instead of using the result of $\mathsf{PEnc}(pk, ID\|r\|rk\|pw)$, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|0^{|rk|+|pw|})$, and uses r and c as the $trans_{\mathsf{PRR}}$ corresponding to ID.
- If $\mathcal{A}_1$ issues a $\mathsf{Trans}_{\mathsf{Auth}}$ query on ID, then instead of using the result of $\mathsf{PEnc}(pk, ID\|r\|pw)$, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|0^{|pw|})$, and uses r and c as the $trans_{\mathsf{Auth}}$ corresponding to ID.

[5] Strictly speaking, multi-challenge 1-bounded CCA secure PKE [11] is enough for the security proof.

For $i \in \{0,1,2,3,4\}$, we define the event W_i as the event that $\mathcal{A}$ wins in Game i. The advantage of $\mathcal{A}$ is, by definition, $\mathsf{Adv}^{\mathsf{Imp\text{-}PA}}_{\mathcal{A}}(k) = \Pr[W_0]$. We complete the proof by using the following inequality, and the upper bounds in the terms in the right hand side are shown in Lemmas 1 to 5.

$$\Pr[W_0] \leq \sum_{i=0}^{3} |\Pr[W_i] - \Pr[W_{i+1}]| + \Pr[W_4] \tag{1}$$

Lemma 1. $|\Pr[W_0] - \Pr[W_1]|$ *is negligible.*

Proof. The difference $|\Pr[W_0] - \Pr[W_1]|$ can be upperbounded by the statistical distance between the distributions of r's used in PRR (in the responses to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{PRR}}$ queries) and Auth (in the responses to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{Auth}}$ queries and in Auth the challenge phase) in Game 0 and those in Game 1. Since the number of r's in the games is at most $(q_P + q_A + 1)$, the statistical distance between the distributions is at most $(q_P + q_A + 1)^2/2^k$. This completes the proof of Lemma 1. □

Lemma 2. *If F is a PRF,* $|\Pr[W_1] - \Pr[W_2]|$ *is negligible.*

Proof. We show that we can construct an adversary $\mathcal{B}$ against the PRF F. The description of $\mathcal{B}$ is as follows:

First, the challenger chooses a key $K \in \{0,1\}^k$ and the challenge bit $\{0,1\}$ uniformly at random (which are both unknown to $\mathcal{B}$). $\mathcal{B}$ executes $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ and generates an empty list L which will be used to store tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$. $\mathcal{B}$ also generates an empty list $\mathcal{R}$. After that, $\mathcal{B}$ gives $pp := pk$ to $\mathcal{A}_1$.

When $\mathcal{A}_1$ makes a $\mathsf{CCreate}$ query ID, $\mathcal{B}$ responds as follows.

1. If there exists a tuple $(ID, *, *, *, *, *)$ in the list L, $\mathcal{B}$ does nothing.
2. Otherwise, $\mathcal{B}$ submits the identity ID to the challenger, and receives rk. This rk is $F(K, ID)$ if $b = 0$ and is a random string in the range of F if $b = 1$. After that, $\mathcal{B}$ stores $(ID, \phi, \phi, rk, 0, 1)$ into the list L.

When $\mathcal{A}_1$ makes a PR query ID, $\mathcal{B}$ finds the tuple $(ID, pw, *, *, *, *)$ in the list L, and returns pw to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, 0, *)$.

When $\mathcal{A}_1$ makes a RKR query ID, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L, and returns rk to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, *, 0)$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans}_{\mathsf{PRR}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness $r \in \{0,1\}^k$. Next, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw')$ and $ID'\|r'\|rk'\|pw'_s \leftarrow \mathsf{PDec}(dk, c)$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw'_s, *, 0, *)$ and adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ first chooses a random password $pw' \in \mathcal{D}$ (where $\mathcal{D}$ is the dictionary from which an honest user is assumed to sample his/her password) and a randomness $r \in \{0,1\}^k$. Next, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw')$ and $ID'\|r'\|rk'\|pw'_s \leftarrow \mathsf{PDec}(dk, c)$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw'_s, *, 1, *)$ and adds r to the list $\mathcal{R}$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans}_{\mathsf{Auth}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, pw, pw_s, *, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k \backslash \mathcal{R}$, executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw')$ and $ID''\|r''\|pw'' \leftarrow \mathsf{PDec}(dk, c)$. Next, $\mathcal{B}$ sets $z := \top$ if $ID'' = ID$, $r'' = r$, and $pw'' = pw_s$ hold. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k \backslash \mathcal{R}$, executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$ and $ID''\|r''\|pw'' \leftarrow \mathsf{PDec}(dk, c)$. Next, $\mathcal{B}$ sets $z := \top$ if $ID'' = ID$,$r'' = r$, and $pw'' = pw_s$ hold. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.

Finally, $\mathcal{A}_1$ terminates with (ID^*, st). $\mathcal{B}$ outputs $b' = 1$ and aborts when the identity ID^* does not satisfy the winning precondition. Otherwise, $\mathcal{B}$ chooses r^* uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$, and gives r^* and st to $\mathcal{A}_2$. $\mathcal{B}$ can respond to the queries from $\mathcal{A}_2$ in the same way as $\mathcal{B}$ did for $\mathcal{A}_1$. However, when $\mathcal{A}_2$ submits ID^* as a RKR or $\mathsf{Trans_{PRR}}$ query, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ terminates with c^*. Next, $\mathcal{B}$ executes $r'^* \| pw'^* \leftarrow \mathsf{PDec}(dk, c^*)$. Then, $\mathcal{B}$ finds the tuple $(ID^*, *, pw_s^*, *, *, *)$ in the list L and checks whether the conditions $r'^* = r^*$ and $pw'^* = pw_s^*$ hold or not. If these conditions hold, $\mathcal{B}$ terminates with $b' = 0$. Otherwise, $\mathcal{B}$ terminates with $b' = 1$.

The above completes the description of $\mathcal{B}$. It is not hard to see that $\mathcal{B}$ perfectly simulates Game 1 for $\mathcal{A}$ when $\mathcal{B}$'s challenge bit $b = 0$, and Game 2 when $b = 1$. When the challenge bit of $\mathcal{B}$ is 0 and $\mathcal{B}$ does not abort before $\mathcal{A}$ terminates, $\mathcal{B}$'S responses to $\mathcal{A}$'s queries are performed in exactly the same way as those in Game 1. In addition, $\mathcal{B}$ outputs 0 only if $\mathcal{B}$ does not abort and $\mathcal{A}_2$ succeeds in outputting a ciphertext $c^* = \mathsf{PEnc}(pk, ID^* \| r^* \| pw_s^*)$. Therefore, $\Pr[b' = 0 | b = 0] = \Pr[W_1]$. On the other hand, when the challenge bit of $\mathcal{B}$ is 1, the response of the challenger of $\mathcal{B}$ is a random string, and this situation is the same as Game 2. With almost the same discussion as above, we have $\Pr[b' = 0 | b = 1] = \Pr[W_2]$. Therefore, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{PRF}}(k) = |\Pr[b' = b] - 1/2| = (1/2)|\Pr[b' = 0 | b = 0] - \Pr[b' = 0 | b = 1]| = (1/2)|\Pr[W_1] - \Pr[W_2]|$. Using this equality, and recalling the assumption that the underlying F is a PRF, we conclude that $|\Pr[W_1] - \Pr[W_2]|$ is negligible. This completes the proof of Lemma 2. □

Lemma 3. $\Pr[W_2] = \Pr[W_3]$.

Proof. Notice that the difference between Game 2 and Game 3 is only in how a $\mathsf{Trans_{Auth}}$ query is answered. More concretely, $\mathcal{B}$ runs PDec, obtains pw_s, and returns $\top$ to $\mathcal{A}$ if $pw = pw_s$ holds in Game 2. On the other hand, $\mathcal{B}$ simply checks whether the $pw = pw_s$ holds or not, and returns $\top / \perp$ to $\mathcal{A}$ in Game 3. Here, pw_s is the value in the tuple corresponding to ID in the list L. However, the results in Game 2 and Game 3 always agree, due to the correctness of the underlying PKE scheme. Therefore, Game 2 and Game 3 are identical. This completes the proof of Lemma 3. □

Lemma 4. *If the PKE scheme is mIND-CCA secure,* $|\Pr[W_3] - \Pr[W_4]|$ *is negligible.*

Proof. We show that we can construct a multi-challenge IND-CCA adversary $\mathcal{B}$ against the underlying PKE scheme. The description of $\mathcal{B}$ is as follows:

First, the challenger executes $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ and chooses the challenge bit $b \in \{0,1\}$ uniformly at random. Then, the challenger gives pk to $\mathcal{B}$. $\mathcal{B}$ chooses a random key $K \in \{0,1\}^k$ for PRF F and generates an empty list L which will be used to store tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$. $\mathcal{B}$ also generates an empty list $\mathcal{R}$. After that, $\mathcal{B}$ gives $pp := pk$ to $\mathcal{A}_1$.

When $\mathcal{A}_1$ makes a $\mathsf{CCreate}$ query ID, $\mathcal{B}$ responds as follows.

1. If there exists a tuple $(ID, *, *, *, *, *)$ in the list L, $\mathcal{B}$ does nothing.
2. Otherwise, $\mathcal{B}$ chooses $rk \in \{0,1\}^k$ uniformly at random from the range of F and gives rk to $\mathcal{A}_1$. After that, $\mathcal{B}$ stores the tuple $(ID, \phi, \phi, rk, 0, 1)$ into the list L.

When $\mathcal{A}_1$ makes a PR query ID, $\mathcal{B}$ finds the tuple $(ID, pw, *, *, *, *)$ in the list L, and returns pw to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, 0, *)$.

When $\mathcal{A}_1$ makes a RKR query ID, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L, and returns rk to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, *, 0)$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans_{PRR}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness $r \in \{0,1\}^k$, submits $(ID \| r \| rk \| pw', ID \| r \| 0^{|rk|+|pw'|})$ to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID \| r \| rk \| pw')$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID \| r \| 0^{|rk|+|pw'|})$ if $b = 1$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw', *, 0, *)$ and adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ chooses a random password $pw' \in \mathcal{D}$ (where $\mathcal{D}$ is the dictionary) and a randomness $r \in \{0,1\}^k$, submits $(ID \| r \| rk \| pw', ID \| r \| 0^{|rk|+|pw'|})$ to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID \| r \| rk \| pw')$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID \| r \| 0^{|rk|+|pw'|})$ if $b = 1$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw', *, 1, *)$ and adds r to the list $\mathcal{R}$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans}_{\mathsf{Auth}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, pw, pw_s, *, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k \backslash \mathcal{R}$, submits $(ID\|r\|pw', ID\|r\|0^{|pw'|})$ to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID\|r\|pw')$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID\|r\|0^{|pw'|})$ if $b = 1$. Next, $\mathcal{B}$ sets $z := \top$ if $pw' = pw_s$ holds. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k \backslash \mathcal{R}$, submits $(ID\|r\|pw, ID\|r\|0^{|pw|})$ as a challenge query to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID\|r\|pw)$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID\|r\|0^{|pw|})$ if $b = 1$. Next, $\mathcal{B}$ sets $z := \top$ if $pw = pw_s$ holds. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.

Finally, $\mathcal{A}_1$ terminates with (ID^*, st). $\mathcal{B}$ outputs $b' = 1$ and aborts when the identity ID^* does not satisfy the winning precondition. Otherwise, $\mathcal{B}$ chooses r^* uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$, and gives r^* and st to $\mathcal{A}_2$. $\mathcal{B}$ can respond to the queries from $\mathcal{A}_2$ in the same way as $\mathcal{B}$ did for $\mathcal{A}_1$. However, when $\mathcal{A}_2$ submits ID^* as a RKR or $\mathsf{Trans}_{\mathsf{PRR}}$ query, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ terminates with c^*. If this c^* is one of the ciphertexts used as a response to the $\mathsf{Trans}_{\mathsf{PRR}}$ or $\mathsf{Trans}_{\mathsf{Auth}}$ queries, $\mathcal{B}$ stops the Imp-PA game, decides that $\mathcal{A}$ has lost the Imp-PA game, and terminates with $b' = 1$. Next, $\mathcal{B}$ submits c^* as a decryption query[6] to the challenger, and receives $ID^*\|r'^*\|pw'^*$. Then, $\mathcal{B}$ finds the tuple $(ID^*, *, pw_s^*, *, *, *)$ in the list L and checks whether the conditions $r'^* = r^*$ and $pw'^* = pw_s^*$ hold or not. If these conditions hold, $\mathcal{B}$ terminates with output $b' = 0$. Otherwise, $\mathcal{B}$ terminates with output $b' = 1$.

The above completes the description of $\mathcal{B}$. It is not hard to see that $\mathcal{B}$ perfectly simulates Game 3 when $\mathcal{B}$'s challenge bit $b = 0$, and Game 4 when $b = 1$. When the challenge bit of $\mathcal{B}$ is 0 and $\mathcal{B}$ does not abort before $\mathcal{A}$ terminates, $\mathcal{B}$ responses to $\mathcal{A}$'s queries are distributed identically to those in Game 3. In addition, $\mathcal{B}$ outputs 0 only if $\mathcal{B}$ does not abort and $\mathcal{A}_2$ succeeds in outputting a ciphertext c^* satisfying $\mathsf{PDec}(dk, c^*) = ID^*\|r^*\|pw_s^*$. Therefore, $\Pr[b' = 0|b = 0] = \Pr[W_3]$. On the other hand, when the challenge bit of $\mathcal{B}$ is 1, $\mathcal{A}_2$ succeeds in outputting a ciphertext c^* satisfying $\mathsf{PDec}(dk, c^*) = ID^*\|r^{\|}0^{|pw_s^*|}$, and this situation is the same as Game 4. With almost the same discussion as above, we have $\Pr[b' = 0|b = 1] = \Pr[W_4]$. Therefore, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{mIND\text{-}CCA}}(k) = |\Pr[b' = b] - 1/2| = (1/2)|\Pr[b' = 0|b = 0] - \Pr[b' = 0|b = 1]| = (1/2)|\Pr[W_3] - \Pr[W_4]|$. Using this equality, and recalling the assumption that the underlying PKE scheme is mIND-CCA secure, we conclude that $|\Pr[W_3] - \Pr[W_4]|$ is negligible. This completes the proof of Lemma 4. □

Lemma 5. $\Pr[W_4] \leq (q_A + 1)/|\mathcal{D}|$.

Proof. Note that in Game 4, the "transcript" part in the responses to the $\mathsf{Trans}_{\mathsf{PRR}}$ and $\mathsf{Trans}_{\mathsf{Auth}}$ queries contain no information of pw. However, if an adversary makes a $\mathsf{Trans}_{\mathsf{Auth}}$ query (ID, pw) with $pw \neq \phi$, the "server's output" part (i.e. $\top$ or $\bot$) leaks whether $pw = pw'$. However, other than this, no information about pw leaks. Since pw is chosen randomly from $\mathcal{D}$, the probability that $\mathcal{A}$ wins in Game 4 is at most $(q_A+1)/|\mathcal{D}|$. This completes the proof of Lemma 5. □

Lemmas 1 to 5 guarantee that the right hand side of the inequality (1) is upper-bounded by $O(q_A)/|\mathcal{D}| + \varepsilon(k)$. This completes the proof of Theorem 1. □

Theorem 2. *If F is a PRF and the PKE scheme is mIND-CCA secure[7], then the proposed password reset protocol in Fig. 1 satisfies IR-PA security.*

Proof of Theorem 2. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an Imp-PA adversary of the password reset protocol. Let q_P be the number of $\mathsf{Trans}_{\mathsf{PRR}}$ queries by $\mathcal{A}_1$. Here, q_P is a polynomial of the security parameter. Consider the following sequence of games.

Game 0. This is exactly the IR-PA game.

Game 1. This game proceeds in the same way as Game 0, except that the first messages r of S_P picked in the execution of PRR (either in the response to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{PRR}}$ query or in the challenge phase) and Auth (in the response to $\mathcal{A}$'s $\mathsf{Trans}_{\mathsf{Auth}}$ query), are picked from $\{0,1\}^k \backslash \{r$'s that are already used $\}$, so

[6] This is the only decryption query that $\mathcal{B}$ submits, which is the reason why 1-bounded CCA security [11] suffices.

[7] Multi-challenge 1-bounded CCA secure PKE is enough for the security proof.

that they are all distinct and never collide. For notational convenience, in this and subsequent games, we introduce the list $\mathcal{R}$ that is used to store r's that are used in the response to the $\mathsf{Trans}_{\mathsf{PRR}}$ query or in the execution of PRR in the challenge phase and $\mathsf{Trans}_{\mathsf{Auth}}$ query, and we make the challenger choose r uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$ every time it needs to choose r for PRR and Auth, and put the used r into the list $\mathcal{R}$.

Game 2. This game proceeds in the same way as Game 1, except that if $\mathcal{A}_1$ issues a $\mathsf{Trans}_{\mathsf{PRR}}$ query on ID, then instead of using the result of $F(K, ID)$, $\mathcal{B}$ picks rk uniformly at random from the range of F, and uses it as the reset key corresponding to ID.

Game 3. This game proceeds in the same way as Game 2, except that if $\mathcal{A}_1$ issues a $\mathsf{Trans}_{\mathsf{PRR}}$ query on ID, then instead of using the result of $\mathsf{PEnc}(pk, ID\|r\|rk\|pw)$, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|0^{|rk|+|pw|})$, and uses r and c as the $trans_{\mathsf{PRR}}$ corresponding to ID.

For $i \in \{0,1,2,3\}$, we define the event W_i as the event that $\mathcal{A}$ wins in Game i. The advantage of $\mathcal{A}$ is, by definition, $\mathsf{Adv}^{\mathsf{IR\text{-}PA}}_{\mathcal{A}}(k) = \Pr[W_0]$. We complete the proof by using the following inequality, and the upper bounds in the terms in the right hand side are shown in Lemmas 6 to 9.

$$\Pr[W_0] \leq \sum_{i=0}^{2} |\Pr[W_i] - \Pr[W_{i+1}]| + \Pr[W_3] \tag{2}$$

Lemma 6. $|\Pr[W_0] - \Pr[W_1]|$ *is negligible.*

Proof. This proof follows closely to that of Lemma 1. Therefore, we omit it. □

Lemma 7. *If the F is a PRF,* $|\Pr[W_1] - \Pr[W_2]|$ *is negligible.*

Proof. This proof follows closely to that of Lemma 2. Therefore, we omit it. □

Lemma 8. *If the PKE scheme is mIND-CCA secure,* $|\Pr[W_2] - \Pr[W_3]|$ *is negligible.*

Proof. We show that we can construct a multi-challenge IND-CCA adversary $\mathcal{B}$ against the underlying PKE scheme. The description of $\mathcal{B}$ is as follows:

First, the challenger executes $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ and chooses the challenge bit $b \in \{0,1\}$ uniformly at random. Then, the challenger gives pk to $\mathcal{B}$. $\mathcal{B}$ chooses a random key $K \in \{0,1\}^k$ for PRF F and generates an empty list L which will be used to store tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$. $\mathcal{B}$ also generates an empty list $\mathcal{R}$. After that, $\mathcal{B}$ gives $pp := pk$ to $\mathcal{A}_1$.

When $\mathcal{A}_1$ makes a $\mathsf{CCreate}$ query ID, $\mathcal{B}$ responds as follows.

1. If there exists a tuple $(ID, *, *, *, *, *)$ in the list L, $\mathcal{B}$ does nothing.
2. Otherwise, $\mathcal{B}$ chooses $rk \in \{0,1\}^k$ uniformly at random from the range of F and gives rk to $\mathcal{A}_1$. After that, $\mathcal{B}$ stores the tuple $(ID, \phi, \phi, rk, 0, 1)$ into the list L.

When $\mathcal{A}_1$ makes a PR query ID, $\mathcal{B}$ finds the tuple $(ID, pw, *, *, *, *)$ in the list L, and returns pw to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, 0, *)$.

When $\mathcal{A}_1$ makes a RKR query ID, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L, and returns rk to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, *, 0)$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans}_{\mathsf{PRR}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness $r \in \{0,1\}^k \backslash \mathcal{R}$, submits $(ID\|r\|rk\|pw', ID\|r\|0^{|rk|+|pw'|})$ to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID\|r\|rk\|pw')$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID\|r\|0^{|rk|+|pw'|})$ if $b = 1$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw', *, 0, *)$ and adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ chooses a random password $pw' \in \mathcal{D}$ (where $\mathcal{D}$ is the dictionary) and a randomness $r \in \{0,1\}^k \backslash \mathcal{R}$, submits $(ID\|r\|rk\|pw', ID\|r\|0^{|rk|+|pw'|})$ to the challenger, and receives c. This c is $\mathsf{PEnc}(pk, ID\|r\|rk\|pw')$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID\|r\|0^{|rk|+|pw'|})$ if $b = 1$. Then, $\mathcal{B}$ returns $trans_{\mathsf{PRR}} := (r, c)$ and the registration result $z := \top$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L by $(ID, pw', pw', *, 1, *)$ and adds r to the list $\mathcal{R}$.

When $\mathcal{A}_1$ makes a $\mathsf{Trans}_{\mathsf{Auth}}$ query (ID, pw') where $pw' \in \mathcal{PW} \cup \{\phi\}$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ finds the tuple $(ID, pw, pw_s, *, *, *)$ in the list L.

1. If $pw' \in \mathcal{PW}$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k$ and executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw')$. Next, $\mathcal{B}$ sets $z := \top$ if $pw' = pw_s$ holds. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.
2. If $pw' = \phi$, $\mathcal{B}$ chooses a randomness r from $\{0,1\}^k$ and executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$. Next, $\mathcal{B}$ sets $z := \top$ if $pw = pw_s$ holds. Otherwise, $\mathcal{B}$ sets $z := \bot$. Then, $\mathcal{B}$ returns $trans_{\mathsf{Auth}} := (r, c)$ and the authentication result z to $\mathcal{A}_1$. After that, $\mathcal{B}$ adds r to the list $\mathcal{R}$.

Finally, $\mathcal{A}_1$ terminates with (ID^*, st). $\mathcal{B}$ outputs $b' = 1$ and aborts when the identity ID^* does not satisfy the winning precondition. Otherwise, $\mathcal{B}$ chooses r^* uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$, and gives r^* and st to $\mathcal{A}_2$. $\mathcal{B}$ responds to the queries from $\mathcal{A}_2$ in the same way as $\mathcal{B}$ did for $\mathcal{A}_1$. However, when $\mathcal{A}_2$ submits ID^* as a RKR or $\mathsf{Trans}_{\mathsf{PRR}}$ query, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ terminates with c^*. If this c^* is one of the ciphertexts as a response to the $\mathsf{Trans}_{\mathsf{PRR}}$ queries, $\mathcal{B}$ stops the IR-PA game, decides that $\mathcal{A}$ has lost the IR-PA game, and terminates with $b' = 1$. Next, $\mathcal{B}$ submits c^* as a decryption query[8] to the challenger, and receives $ID'^*\|r'^*\|rk'^*\|pw'^*$. Then, $\mathcal{B}$ finds the tuple $(ID^*, *, pw_s^*, rk^*, *, *)$ in the list L and checks whether the conditions $ID'^* = ID^*$, $r'^* = r^*$, $rk'^* = rk^*$, and $pw'^* = pw_s^*$ hold or not. If these conditions hold, $\mathcal{B}$ terminates with $b' = 0$. Otherwise, $\mathcal{B}$ terminates with $b' = 1$.

The above completes the description of $\mathcal{B}$. It is not hard to see that $\mathcal{B}$ perfectly simulates Game 2 when $\mathcal{B}$'s challenge bit $b = 0$, and Game 3 when $b = 1$. When the challenge bit of $\mathcal{B}$ is 0 and $\mathcal{B}$ does not abort before $\mathcal{A}$ terminates, $\mathcal{B}$ responses to $\mathcal{A}$'s queries are distributed identically to those in Game 2. In addition, $\mathcal{B}$ outputs 0 only if $\mathcal{B}$ does not abort and $\mathcal{A}_2$ succeeds in outputting a ciphertext that is decrypted to the $ID^*\|r^*\|rk^*\|pw_s^*$. Therefore, $\Pr[b' = 0 | b = 0] = \Pr[W_2]$. On the other hand, when the challenge bit of $\mathcal{B}$ is 1, $\mathcal{A}_2$ succeeds in outputting a ciphertext that is decrypted to the $ID^*\|r^*\|0^{|rk^*|+|pw_s^*|}$, and this situation is the same as Game 3. With almost the same discussion as above, we have $\Pr[b' = 0 | b = 1] = \Pr[W_3]$. Therefore, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{mIND\text{-}CCA}}(k) = |\Pr[b' = b] - 1/2| = (1/2)|\Pr[b' = 0 | b = 0] - \Pr[b' = 0 | b = 1]| = (1/2)|\Pr[W_2] - \Pr[W_3]|$. Using this equality, and recalling the assumption that the underlying PKE scheme is mIND-CCA secure, we conclude that $|\Pr[W_2] - \Pr[W_3]|$ is negligible. This completes the proof of Lemma 8. □

Lemma 9. $\Pr[W_3]$ *is negligible.*

Proof. Note that in Game 3, the "transcript" part in the responses to the $\mathsf{Trans}_{\mathsf{PRR}}$ queries contain no information about rk. Therefore, the probability that $\mathcal{A}$ succeeds the guess of rk (that is, the probability that $\mathcal{A}$ wins in Game 3) is negligible. This completes the proof of Lemma 9. □

Lemmas 6 to 9 guarantee that the right hand side of the inequality (2) is negligible, and thus $\mathcal{A}$ has negligible advantage in the IR-PA game. This completes the proof of Theorem 2. □

5 Extensions

In this section, we discuss two extensions of our password reset protocol.

5.1 Password Reset Protocol with Password Salting

In practice, it is recommended not to store the client's raw password into the server. The server stores "processed data" instead of the client's password itself, and uses it for the authentication. Our proposed password reset protocol can be easily extended to a protocol with password salting. In the setup procedure, a server chooses another key K'. In the password re-registration procedure, the server decrypts the password pw from a ciphertext ct, executes $pw_s := F(K', pw)$, and stores pw_s. In the authentication procedure, server checks whether the condition $F(k', pw') = pw_s$ holds or not, and outputs $\top/\bot$. In this scheme, pw_s which is stored in the server collides if different users set the same password. To prevent this situation, we modify the scheme to compute $F(K', ID\|pw)$ instead of $F(K', pw)$. Although secret key size of this scheme becomes bigger compared to the original scheme, we can avoid it by using the domain separation technique. That

[8] This is the only decryption query that $\mathcal{B}$ submits, which is the reason why 1-bounded CCA security [11] suffices.

$\mathsf{SSetup}(1^k)$: $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ Choose a random $K \in \{0,1\}^k$ $pp := pk$; $sk := (K, dk)$ return (pp, sk).
$\mathsf{RKG}(sk, ID)$: $(K, dk) \leftarrow sk$ $rk := F(K, 0\|ID)$ return rk.
$(\phi, pw_s) \leftarrow \mathsf{PRR}(C_P(ID, pw, rk) \leftrightarrow S_P(ID, rk, sk))$: $K \leftarrow sk$ 1. S_P chooses a randomness $r \in \{0,1\}^k$ and sends it to C_P 2. C_P executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw)$ and sends it to S_P 3-1. S_P executes $ID'\|r'\|rk'\|pw' \leftarrow \mathsf{PDec}(dk, c)$ 3-2. S_P executes $pw_s := F(K, 1\|ID'\|pw')$ 3-3. If $ID' = ID$, $r' = r$, and $rk' = rk$ hold, S_P returns pw_s else S_P returns $\perp$.
$(\phi, \top/\perp) \leftarrow \mathsf{Auth}(C_A(ID, pw) \leftrightarrow S_A(ID, pw_s, sk))$: $K \leftarrow sk$ 1. S_A chooses a randomness $r \in \{0,1\}^k$ and sends it to C_A 2. C_A executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$ and sends it to S_A 3-1. S_A executes $ID'\|r'\|pw' \leftarrow \mathsf{PDec}(dk, c)$ 3-2. If $ID' = ID$, $r' = r$, and $pw_s = F(K, 1\|ID\|pw')$ hold, S_A returns $\top$ else S_A returns $\perp$.

Fig. 2. The proposed generic construction of a password reset protocol with password salting

is, instead of preparing the new key K', the server uses a key K for two purposes by adding the prefix bit. When the server generates a reset key rk, it executes $rk := F(K, 0\|ID)$. When the server generates pw_s, it generates $pw_s := F(K, 1\|ID\|pw)$.

Our password reset protocol with password salting is constructed as in Fig. 2. The security proofs for this extended scheme follow closely to the security proofs of original scheme. Therefore, we omit them in this paper.

5.2 Security against Active Adversary

In Section 3.2, we only considered security against passive attacks. In this section, we give the formal security definitions against active attacks for a password reset protocol by extending the security definitions for passive ones, and show that our proposed protocol in Section 4 (and Section 5.1) satisfies them under the same assumptions on the building blocks.

Impersonation First, we consider security against impersonation under active attacks (Imp-AA) for a password reset protocol. Imp-AA security is defined using the following Imp-AA game which is played by the challenger $\mathcal{B}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. First, $\mathcal{B}$ executes $(pp, sk) \leftarrow \mathsf{SSetup}(1^k)$, and generates an empty list L into which tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$ where $flag_p, flag_r \in \{0,1\}$ will be stored. These $flag_p$ and $flag_r$ are used to indicate whether a client with ID is "corrupted" by $\mathcal{A}$ in the sense that either pw or rk is known to $\mathcal{A}$, in which case $\mathcal{A}$ is not allowed to use the ID for its attack. Moreover, $\mathcal{B}$ generates two lists L'_P and L'_A into which tuples of the form $(sid, ID, pw, st_c, st_s, trans_{sid})$ will be stored. Here, sid means a session ID that uniquely determined an execution of the protocol, st_c (resp. st_s) means the state information stored in the client (resp. server) side during the protocol execution corresponding to sid, and $trans_{sid}$ means a transcript during the protocol execution corresponding to sid. After that, $\mathcal{B}$ gives pp to $\mathcal{A}_1$. Then $\mathcal{A}_1$ can adaptively make the following types of queries[9]. We explain the meaning of $\mathsf{SSession}$ and Send queries in the paragraph of the remark right after Definition 4.

[9] $\mathsf{Trans}_{\mathsf{PRR}}$ and $\mathsf{Trans}_{\mathsf{Auth}}$ queries are not considered in the following list of queries because $\mathcal{A}$ can perform those functionalities by making $\mathsf{Send}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{Auth}}$ queries, respectively.

Client create query (CCreate)**:** This is exactly the same as the $\mathsf{CCreate}$ query in the Imp-PA game. If $\mathcal{A}_1$ makes the following queries ($\mathsf{RKR}, \mathsf{PRR}, \mathsf{SSession}_{\mathsf{PRR}}, \mathsf{SSession}_{\mathsf{Auth}}$) with an identity ID, then this ID must have appeared as a $\mathsf{CCreate}$ query (and thus be stored in the list L).

Password reveal query (PR)**:** This is exactly the same as the PR query in the Imp-PA game.

Reset key reveal query (RKR)**:** This is exactly the same as the RKR query in the Imp-PA game.

Start session query for password (re-)registration ($\mathsf{SSession}_{\mathsf{PRR}}$) On input $(ID, pw \in \mathcal{PW} \cup \{\phi\})$, $\mathcal{B}$ responds as follows. First, $\mathcal{B}$ generates a unique session ID sid and returns it to $\mathcal{A}$. If $pw \in \mathcal{PW}$, $\mathcal{B}$ stores $(sid, ID, pw, \phi, \phi, \phi)$ into the list L'_P and updates the tuple in the list L by $(ID, *, *, *, 0, *)$. Otherwise (that is, $pw = \phi$), $\mathcal{B}$ chooses a password pw' uniformly at random from $\mathcal{D}$, stores $(sid, ID, pw', \phi, \phi, \phi)$ into the list L'_P, and updates the tuple in the list L by $(ID, *, *, *, 1, *)$. When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{PRR}}$ query with a session ID sid, then this sid must have been generated by this $\mathsf{SSession}_{\mathsf{PRR}}$ query previously.

Start session query for authentication ($\mathsf{SSession}_{\mathsf{Auth}}$) On input $(ID, pw \in \mathcal{PW} \cup \{\phi\})$, $\mathcal{B}$ responds as follows. If $pw = \mathcal{PW}$, $\mathcal{B}$ generates a unique session ID sid and stores $(sid, ID, pw, \phi, \phi, \phi)$ into the list L'_A. Otherwise (that is, $pw = \phi$), $\mathcal{B}$ finds a tuple $(ID, pw, *, *, *, *)$ in the list L, generates a unique session ID sid, and stores $(sid, ID, pw, \phi, \phi, \phi)$ into the list L'_A. When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{Auth}}$ query with a session ID sid, then this sid must have been generated by this $\mathsf{SSession}_{\mathsf{Auth}}$ query previously.

Send query for password (re-)registration ($\mathsf{Send}_{\mathsf{PRR}}$) On input (sid, i, M) $\mathcal{B}$ interprets M as an $i-$th incoming message of the PRR protocol, and returns an appropriate answer specified in the protocol. (In other words, $\mathcal{B}$ executes the "next message function" of the PRR protocol and returns the answer to $\mathcal{A}$.) More concretely, see the remark right after Definition 4.

Send query for authentication ($\mathsf{Send}_{\mathsf{Auth}}$) On input (sid, i, M) $\mathcal{B}$ interprets M as an $i-$th incoming message of the Auth protocol, and returns an appropriate answer specified in the protocol. (In other words, $\mathcal{B}$ executes the "next message function" of the Auth protocol and returns the answer to $\mathcal{A}$.) More concretely, see the remark right after Definition 4.

Finally, $\mathcal{A}_1$ outputs (ID^*, st). To win the Imp-AA game, the tuple $(ID^*, pw^*, pw_s^*, rk^*, flag_p^*, flag_r^*)$ must exist in the list L and satisfy $flag_p^* = 1$ and $flag_r^* = 1$ (if this is satisfied, we say that ID^* satisfies the "winning precondition"). If these conditions are not satisfied, $\mathcal{B}$ decides that $\mathcal{A}$ has lost the Imp-AA game. Otherwise, $\mathcal{B}$ gives st to $\mathcal{A}_2$. Then $\mathcal{A}_2$ and $\mathcal{B}$ interactively execute $\mathsf{Auth}(\mathcal{A}_2(st) \leftrightarrow S_A(ID^*, pw_s^*, sk))$. During the execution of this Auth protocol, $\mathcal{A}_2$ can adaptively make the queries in the same way as $\mathcal{A}_1$. However, $\mathcal{A}_2$ is not allowed to use ID^* in the PR, RKR, $\mathsf{SSession}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{PRR}}$ queries. Finally, $\mathcal{A}$ wins if S_A's output of Auth is $\top$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Imp\text{-}AA}}(k) = \Pr[\mathcal{A}\ wins]$.

Definition 3. *Let q_{S_A} be the number of $\mathsf{SSession}_{\mathsf{Auth}}$ queries by $\mathcal{A}_1$. We say that a password reset protocol is Imp-AA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Imp\text{-}AA}}(k)$ is $O(q_{S_A})/|\mathcal{D}| + \varepsilon(k)$.*

Illegal Registration Second, we consider security against illegal registration under active attacks (IR-AA) for a password reset protocol. This security is defined using the following IR-AA game which is played by the challenger $\mathcal{B}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{B}$'s initial procedure and $\mathcal{A}_1$'s queries of this IR-AA game are exactly the same as the Imp-AA game. Finally, $\mathcal{A}_1$ outputs (ID^*, st). To win the IR-AA game, the tuple $(ID^*, pw^*, pw_s^*, rk^*, flag_p^*, flag_r^*)$ must exist in the list L and satisfy $flag_r^* = 1$. If these conditions are not satisfied, $\mathcal{B}$ decides that $\mathcal{A}$ has lost the IR-AA game. Otherwise, $\mathcal{B}$ gives st to $\mathcal{A}_2$. Then $\mathcal{A}_2$ and $\mathcal{B}$ interactively execute $\mathsf{PRR}(\mathcal{A}_2(st) \leftrightarrow S_P(ID^*, rk^*, sk))$. During the execution of this PRR protocol, $\mathcal{A}_2$ can adaptively make the queries in the same way as $\mathcal{A}_1$. However, $\mathcal{A}_2$ is not allowed to use ID^* in the RKR, $\mathsf{SSession}_{\mathsf{PRR}}$, and $\mathsf{Send}_{\mathsf{PRR}}$ queries. Finally, $\mathcal{A}$ wins if S_P's output of PRR is different from $\perp$. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IR\text{-}AA}}(k) = \Pr[\mathcal{A}\ wins]$.

Definition 4. *We say that a password reset protocol is IR-AA secure if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IR\text{-}AA}}(k)$ is negligible.*

Remark

1. First, we explain the meaning of $\mathsf{SSession}_{\mathsf{Auth}}$ query of the form (ID, pw) and $\mathsf{Send}_{\mathsf{Auth}}$ query of the form (sid, i, M). Although the following examples are the cases of Auth, the same is true in PRR. These queries are extension of a Send query that we can see in a security model of key exchange protocols. Before

querying $\mathsf{Send}_{\mathsf{Auth}}$, an adversary $\mathcal{A}$ has to query $\mathsf{SSession}_{\mathsf{Auth}}$ query of the form (ID, pw) and obtain sid previously. When this query is issued, $\mathcal{B}$ generates the tuple $(sid, ID, pw, \phi, \phi, \phi)$ in the list L'. If $\mathcal{A}$ issues Send query of the form (sid, i, M), $\mathcal{A}$ can obtain the correct execution result of algorithm that is generated by client/server. Here, M is inserted into the $(i+1)$-th message. In C_P and S_P algorithms of our PRR protocol (in Fig. 1), for example, a client receives a randomness r from the server, executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw)$, and returns c to the server. In this situation, $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, M)$ and obtains $\mathsf{PEnc}(pk, ID\|M\|rk\|pw)$ from $\mathcal{B}$. Since there exists no input message in $\mathsf{Send}_{\mathsf{PRR}}$ of the form $(sid, 1, M)$, this M is always ϕ. In contrast to the security definitions for a key exchange protocol, the attack flags ($flag_p$ and $flag_r$) may revive by resetting a password that is randomly chosen by $\mathcal{B}$. Therefore, we introduce not only a $\mathsf{Send}_{\mathsf{PRR}}$ query but also a $\mathsf{SSession}_{\mathsf{PRR}}$ query to manage the attack flags.
2. We require that a Send query is executed in order. For example, an adversary $\mathcal{A}$ is not allowed to query $(sid, 2, \cdot)$ without querying $(sid, 1, M)$. Moreover, $\mathcal{A}$ is also not allowed to query $(sid, 3, M)$ right after querying $(sid, 1, M)$. This is because executions of algorithms without previous steps do not occur in practice. Moreover, we do not consider an adversary that queries $(sid, 1, M)$ after $(sid, 2, M)$. If the adversary wants to issue these queries, he/she has to issue $\mathsf{SSession}$ queries and obtain new sid. The formalization that captures the above situation somewhat similar to the situation considered in the context of resettable security [10]) is a future work.

Construction and Security Proofs Even if we consider security against active adversaries, the construction is exactly the same as in Fig. 1. Here, we show the security proofs.

Theorem 3. *If F is a PRF and the PKE scheme is mIND-CCA secure*[10]*, then the proposed password reset protocol in Fig. 1 satisfies Imp-AA security.*

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an Imp-AA adversary of the password reset protocol. Let q_{P_i} and q_{A_i} be the numbers of a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form (sid, i, M) and $\mathsf{Send}_{\mathsf{Auth}}$ query of the form (sid, i, M) by $\mathcal{A}_1$, respectively. Here, q_{P_i} and q_{A_i} are polynomials of the security parameter. In our protocol, PRR and Auth are both two-pass protocols. Therefore, we only have to consider the following three cases for $\mathsf{Send}_{\mathsf{PRR}}$ queries.

1. $r \leftarrow \mathsf{Send}_{\mathsf{PRR}}(sid, 1, \phi)$: This means that $\mathcal{A}$ gives ϕ to the server and obtains a randomness r. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 1, \phi)$, $\mathcal{B}$ chooses a randomness r, returns it to $\mathcal{B}$, and updates the tuple in the list L'_P by $(sid, *, *, *, r, r)$.
2. $c \leftarrow \mathsf{Send}_{\mathsf{PRR}}(sid, 2, r)$: This means that $\mathcal{A}$ gives a randomness r to the client and obtains a ciphertext c. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_P and $(ID, *, *, rk, *, *)$ in the list L. Then, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw)$, returns c to $\mathcal{A}$, and updates the tuples in the list L'_P by $(sid, *, *, *, *, *\|c)$ and in the list L by $(ID, pw, *, *, *, *)$.
3. $\top/\bot \leftarrow \mathsf{Send}_{\mathsf{PRR}}(sid, 3, c)$: This means that $\mathcal{A}$ gives a ciphertext c to the server and obtains a registration result $\top/\bot$. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 3, c)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, r, *)$ in the list L'_P and $(ID, *, *, rk, *, *)$ in the list L. Then, $\mathcal{B}$ responds as follows: $\mathcal{B}$ executes $ID'\|r'\|rk'\|pw_s \leftarrow \mathsf{PDec}(dk, c)$. If this decryption result is $\bot$, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}$ and does not execute the following steps. Then, $\mathcal{B}$ sets $z := \top$ if $ID = ID'$, $r = r'$, and $rk = rk'$ hold. Otherwise, $\mathcal{B}$ sets $z := \bot$. After that, $\mathcal{B}$ updates the tuples in the list L by $(ID, *, pw_s, *, *, *)$ and returns z to $\mathcal{A}$.

Similar to the cases of $\mathsf{Send}_{\mathsf{PRR}}$ queries, we only have to consider the following three cases for $\mathsf{Send}_{\mathsf{Auth}}$ queries.

1. $r \leftarrow \mathsf{Send}_{\mathsf{Auth}}(sid, 1, \phi)$: This means that $\mathcal{A}$ gives ϕ to the server and obtains a randomness r. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 1, \phi)$, $\mathcal{B}$ chooses a randomness r, returns it to $\mathcal{B}$, and updates the tuple in the list L'_A by $(sid, *, *, *, r, r)$.
2. $c \leftarrow \mathsf{Send}_{\mathsf{Auth}}(sid, 2, r)$: This means that $\mathcal{A}$ gives a randomness r to the client and obtains a ciphertext c. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_A. Then, $\mathcal{B}$ executes $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$, returns c to $\mathcal{A}$, and updates the tuple in the list L'_A by $(sid, *, *, *, *, *\|c)$.

[10] In the case of security against passive adversaries, 1-bounded CCA secure PKE is enough for the security proof. In the case of security against active adversaries, however, we need (full) CCA secure PKE for the security proof.

3. $\top/\bot \leftarrow \mathsf{Send}_{\mathsf{Auth}}(sid, 3, c)$: This means that $\mathcal{A}$ gives a ciphertext c to the server and obtains an authentication result $\top/\bot$. If $\mathcal{A}$ issues $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 3, c)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, r, *)$ in the list L'_A and $(ID, *, pw_s, *, *, *)$ in the list L. Then, $\mathcal{B}$ executes $ID'\|r'\|pw'_s \leftarrow \mathsf{PDec}(dk, c)$. If the conditions $ID' = ID$, $r' = r$, and $pw'_s = pw_s$ are satisfied, $\mathcal{B}$ returns $z := \top$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ returns $z := \bot$ to $\mathcal{A}$. Then, $\mathcal{B}$ updates the tuple in the list L'_A by $(sid, *, *, *, *, *\|z)$.

Proof of Theorem 3. Consider the following sequence of games.

Game 0. This is exactly the Imp-AA game.

Game 1. This game proceeds in the same way as Game 0, except that the first messages r of S_A picked in the execution of PRR (in the response to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{PRR}}$ query) and Auth (either in the response to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{Auth}}$ query, or Auth in the challenge phase), are picked from $\{0,1\}^k \backslash \{r$'s that are already used $\}$, so that they are all distinct and never collide. For notational convenience, in this and subsequent games, we introduce the list $\mathcal{R}$ that is used to store r's that are used in the response to a $\mathsf{Send}_{\mathsf{PRR}}$ query, $\mathsf{Send}_{\mathsf{Auth}}$ query, or in the execution of Auth in the challenge phase, and we make the challenger choose r uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$ every time it needs to choose r for $\mathsf{Send}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{Auth}}$, and put the used r into the list $\mathcal{R}$.

Game 2. This game is the same as Game 1, except that as in Game 2 considered in the proof of Theorem 1, we replace the output of $F(K, \cdot)$ with a random string.

Game 3. This game proceeds in the same way as Game 2, except the following points:

- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, r)$, then after calculating the response c and updating the tuple in L'_P as in the previous game, $\mathcal{B}$ also stores the value r used by in the same entry in L'_P (say, by appending r in the last of the entry). (Here, note that the value r used by $\mathcal{A}$ may not be identical to the value chosen and used by $\mathcal{B}$ as a response to the $\mathsf{Send}_{\mathsf{PRR}}$ query $(sid, 1, \phi)$ previously made by $\mathcal{A}$.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 3, c)$, then instead of decrypting c, $\mathcal{B}$ responds with $\top$ if $r = \widehat{r}$ holds, and $\bot$ otherwise, where r is the value used as a response to $\mathcal{A}$'s previous $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 1, \phi)$, and $\widehat{r}$ is the value contained in the previous $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{PRR}}$ query $(sid, 2, r)$. (Note that both of the values can be found in L'_P.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, then after calculating the response c and updating the tuple in L'_A as in the previous game, $\mathcal{B}$ also stores the value r in the same entry in L'_A (say, by appending r in the last of the entry). (Here, note that the value used by $\mathcal{A}$ may not be identical to the value chosen and used by $\mathcal{B}$ as a response to the $\mathsf{Send}_{\mathsf{Auth}}$ query $(sid, 1, \phi)$ previously made by $\mathcal{A}$.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 3, c)$, then instead of decrypting c, $\mathcal{B}$ responds with $\top$ if $r = \widehat{r}$ and $pw = pw_s$ hold, and $\bot$ otherwise, where r is the value used as a response to $\mathcal{A}$'s previous $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 1, \phi)$, and $\widehat{r}$ is the value contained in the previous $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, both of which can be found in L'_A, pw is the value also contained in the entry in L'_A corresponding to sid, and pw_s is the value stored in L such that ID in L and ID in the entry corresponding to sid in L'_A and match.

Game 4. This game proceeds in the same way as Game 3, except the following points.

- If $\mathcal{A}_1$ issues $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_P and $(ID, *, *, rk, *, *)$ in the list L. Then, $\mathcal{B}$ executes $c^* \leftarrow \mathsf{PEnc}(pk, ID\|r\|0^{|rk|+|pw|})$ and uses c^* instead of $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|rk\|pw)$.
- If $\mathcal{A}_1$ issues $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_P, executes $c^{**} \leftarrow \mathsf{PEnc}(pk, ID\|r\|0^{|pw|})$, and uses c^{**} instead of $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$.

For $i \in \{0, 1, 2, 3, 4\}$, we define the event W_i as the event that $\mathcal{A}$ wins in Game i. The advantage of $\mathcal{A}$ is, by definition, $\mathsf{Adv}^{\mathsf{Imp\text{-}AA}}_{\mathcal{A}}(k) = \Pr[W_0]$. We complete the proof by using the following inequality, and the upper bounds in the terms in the right hand side are shown in Lemmas 10 to 14.

$$\Pr[W_0] \leq \sum_{i=0}^{3} |\Pr[W_i] - \Pr[W_{i+1}]| + \Pr[W_4] \tag{3}$$

where the above equality can be derived by the triangle inequality.

Lemma 10. $|\Pr[W_0] - \Pr[W_1]|$ *is negligible.*

Proof. The difference $|\Pr[W_0] - \Pr[W_1]|$ can be upperbounded by the statistical distance between the distributions of r's used in PRR and Auth (in the responses to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{PRR}}$ queries, $\mathsf{Send}_{\mathsf{Auth}}$ queries, and Auth in the challenge phase) in Game 0 and those in Game 1. Since the number of r's in the games is at most $(q_{P_1}+q_{A_1}+1)$, the statistical distance between the distributions is at most $(q_{P_1}+q_{A_1}+1)^2/2^k$. This completes the proof of Lemma 10. □

Lemma 11. *If F is a PRF,* $|\Pr[W_1] - \Pr[W_2]|$ *is negligible.*

Proof. This proof follows closely to that of Lemma 2. Therefore, we omit it. □

Lemma 12. $\Pr[W_2] = \Pr[W_3]$.

Proof. Notice that the difference between Game 2 and Game 3 is only in how $\mathsf{Send}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{Auth}}$ queries are answered. More concretely, $\mathcal{B}$ runs PDec, obtains r' and pw_s, and returns $\top$ to $\mathcal{A}$ if $r' = r$ and $pw = pw_s$ hold in Game 2. On the other hand, $\mathcal{B}$ simply checks whether the conditions $\widehat{r} = r$ (and $pw = pw_s$ in the case of $\mathsf{Send}_{\mathsf{Auth}}$ query) hold or not, and returns $\top/\bot$ to $\mathcal{A}$ in Game 3. Here, pw_s is the value in the tuple corresponding to ID in the list L, and r, $\widehat{r}$, and pw are the values in the tuple corresponding to sid in the list L'_P (or L'_A). However, the results in Game 2 and Game 3 always agree, due to the correctness of the underlying PKE scheme. Therefore, Game 2 and Game 3 are identical. This completes the proof of Lemma 12. □

Lemma 13. *If the PKE scheme is mIND-CCA secure,* $|\Pr[W_3] - \Pr[W_4]|$ *is negligible.*

Proof. We show that we can construct a multi-challenge IND-CCA adversary $\mathcal{B}$ against the underlying PKE scheme. The description of $\mathcal{B}$ is as follows:

First, the challenger executes $(pk, dk) \leftarrow \mathsf{PKG}(1^k)$ and chooses a bit $b \in \{0,1\}$. Then, the challenger gives pk to $\mathcal{B}$. $\mathcal{B}$ chooses a random key $K \in \{0,1\}^k$ for PRF F and generates an empty list L which will be used to store tuples of the form $(ID, pw, pw_s, rk, flag_p, flag_r)$. $\mathcal{B}$ also generates two empty lists L'_P and L'_A which will be used to store tuples of the form $(sid, ID, pw, st_c, st_s, trans_{sid}, rand)$. Moreover, $\mathcal{B}$ generates an empty list $\mathcal{R}$. After that, $\mathcal{B}$ gives $pp := pk$ to $\mathcal{A}_1$.

When $\mathcal{A}_1$ makes a $\mathsf{CCreate}$ query ID, $\mathcal{B}$ responds as follows.

1. If there exists a tuple $(ID, *, *, *, *, *)$ in the list L, $\mathcal{B}$ does nothing.
2. Otherwise, $\mathcal{B}$ chooses $rk \in \{0,1\}^k$ uniformly at random from the range of F and gives rk to $\mathcal{A}_1$. After that, $\mathcal{B}$ stores the tuple $(ID, \phi, \phi, rk, 0, 1)$ into the list L.

When $\mathcal{A}_1$ makes a PR query ID, $\mathcal{B}$ finds the tuple $(ID, pw, *, *, *, *)$ in the list L, and returns pw to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, 0, *)$.

When $\mathcal{A}_1$ makes a RKR query ID, $\mathcal{B}$ finds the tuple $(ID, *, *, rk, *, *)$ in the list L, and returns rk to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L by $(ID, *, *, *, *, 0)$.

When $\mathcal{A}_1$ makes a $\mathsf{SSession}_{\mathsf{PRR}}$ query $(ID, pw \in \mathcal{PW} \cup \{\phi\})$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ generates a unique sid and returns it to $\mathcal{A}$. If $pw \in \mathcal{PW}$, $\mathcal{B}$ stores $(sid, ID, pw, \phi, \phi, \phi, \phi)$ into the list L'_P and updates the tuple in the list L by $(ID, *, *, *, 0, *)$. Otherwise (that is $pw = \phi$), $\mathcal{B}$ chooses pw' uniformly at random from $\mathcal{D}$, stores $(sid, ID, pw', \phi, \phi, \phi, \phi)$ into the list L'_P, and updates the tuple in the list L by $(ID, *, *, *, 1, *)$.

When $\mathcal{A}_1$ makes a $\mathsf{SSession}_{\mathsf{Auth}}$ query $(ID, pw \in \mathcal{PW} \cup \{\phi\})$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ generates a unique sid and returns it to $\mathcal{A}$. If $pw \in \mathcal{PW}$, $\mathcal{B}$ stores $(sid, ID, pw, \phi, \phi, \phi, \phi)$ into the list L'_A. Otherwise, (that is, $pw = \phi$), $\mathcal{B}$ finds a tuple $(ID, pw, *, *, *, *)$ in the list L, generates a unique session ID sid, and stores $(sid, ID, pw, \phi, \phi, \phi, \phi)$ into the list L'_A.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 1, \phi)$, $\mathcal{B}$ responds as follows: $\mathcal{B}$ first chooses a randomness r^* from $\{0,1\}^k \backslash \mathcal{R}$ and returns it to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L'_P by $(sid, *, *, *, r^*, r^*, *)$ and adds r^* to the list $\mathcal{R}$.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ finds the tuples $(ID, *, *, rk, *, *)$ in the list L and $(sid, ID, pw, *, *, *, *)$ in the list L'_P. Then, $\mathcal{B}$ submits $(ID\|r\|rk\|pw, ID\|r\|0^{|rk|+|pw|})$ to the challenger and receives c^*. This c^* is $\mathsf{PEnc}(pk, ID\|r\|rk\|pw)$ if $b = 0$ and is $\mathsf{PEnc}(pk,$

$ID\|r\|0^{|rk|+|pw|}$) if $b = 1$. Then, $\mathcal{B}$ returns c^* to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuples in the list L'_P by $(sid, *, *, *, *, *\|c^*, r)$ and L by $(ID, pw, *, *, *, *)$.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 3, c)$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ finds the tuple $(sid, ID, pw, *, r, *\|c', \widehat{r})$ and in the list L'_P. If the conditions $c = c'$ and $r = \widehat{r}$ hold, $\mathcal{B}$ sets $pw_s := pw$, updates the tuple $(ID, *, pw_s, *, *, *)$ in the list L, and returns $z := \top$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ returns $z := \bot$ to $\mathcal{A}$. After that, $\mathcal{B}$ updates the tuple in the list L'_P by $(sid, *, *, *, *, *\|z)$.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 1, \phi)$, $\mathcal{B}$ responds as follows: $\mathcal{B}$ first chooses a randomness r^* from $\{0,1\}^k \backslash \mathcal{R}$ and returns it to $\mathcal{A}_1$. Then, $\mathcal{B}$ updates the tuple in the list L'_A by $(sid, *, *, *, r^*, r^*)$ and adds r^* to the list $\mathcal{R}$.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ finds the tuple $(sid, ID, pw, *, *, *)$ in the list L'_A. Then, $\mathcal{B}$ submits $(ID\|r\|pw, ID\|r\|0^{|pw|})$ to the challenger and receives c^*. This c^* is $\mathsf{PEnc}(pk, ID\|r\|pw)$ if $b = 0$ and is $\mathsf{PEnc}(pk, ID\|r\|0^{|pw|})$ if $b = 1$. Then, $\mathcal{B}$ returns c^* to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L'_A by $(sid, *, *, *, *, *\|c^*)$.

When $\mathcal{A}_1$ makes a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 3, c)$, $\mathcal{B}$ responds as follows: First, $\mathcal{B}$ finds the tuples $(ID, *, pw_s, *, *, *)$ in the list L and $(sid, ID, pw, *, r, *, \widehat{r})$ in the list L'_A. If the conditions $r = \widehat{r}$ and $pw = pw_s$ hold, $\mathcal{B}$ returns $z := \top$ to $\mathcal{A}_1$. Otherwise, $\mathcal{B}$ returns $z := \bot$ to $\mathcal{A}_1$. After that, $\mathcal{B}$ updates the tuple in the list L'_A by $(sid, *, *, *, *, *\|z)$.

Finally, $\mathcal{A}_1$ terminates with (ID^*, st). $\mathcal{B}$ outputs $b' = 1$ and aborts when the identity ID^* does not satisfy the winning precondition. Otherwise, $\mathcal{B}$ chooses r^* uniformly at random from $\{0,1\}^k \backslash \mathcal{R}$, and gives r^* and st to $\mathcal{A}_2$. $\mathcal{B}$ can respond to the queries from $\mathcal{A}_2$ in the same way as $\mathcal{B}$ did for $\mathcal{A}_1$. However, when $\mathcal{A}_2$ submits ID^* as PR, RKR, $\mathsf{SSession}_{\mathsf{PRR}}$, and $\mathsf{Send}_{\mathsf{PRR}}$ queries, $\mathcal{B}$ returns $\bot$ to $\mathcal{A}_2$. Finally, $\mathcal{A}_2$ terminates with c^*. If this c^* is one of the ciphertexts used as a response to the $\mathsf{Send}_{\mathsf{PRR}}$ or $\mathsf{Send}_{\mathsf{Auth}}$ queries, $\mathcal{B}$ stops the Imp-AA game, decides that $\mathcal{A}$ has lost the Imp-AA game, and terminates with $b' = 1$. Next, $\mathcal{B}$ submits c^* as a decryption query to the challenger, and receives $ID^*\|r'^*\|pw'^*$. Then, $\mathcal{B}$ finds the tuple $(sid, ID^*, pw^*, *, *, *)$ in the list L'_P, checks whether the conditions $r'^* = r^*$ and $pw'^* = pw^*$ hold or not. If these conditions hold, $\mathcal{B}$ terminates with output $b' = 0$. Otherwise, $\mathcal{B}$ terminates with output $b' = 1$.

The above completes the description of $\mathcal{B}$. It is not hard to see that $\mathcal{B}$ perfectly simulates Game 3 when $\mathcal{B}$'s challenge bit $b = 0$, and Game 4 when $b = 1$. When the challenge bit of $\mathcal{B}$ is 0 and $\mathcal{B}$ does not abort before $\mathcal{A}$ terminates, $\mathcal{B}$'s responses to $\mathcal{A}$'s queries are distributed identically to those in Game 3. In addition, $\mathcal{B}$ outputs 0 only if $\mathcal{B}$ does not abort and $\mathcal{A}_2$ succeeds in outputting a ciphertext c^* satisfying $\mathsf{PDec}(dk, c^*) = ID^*\|r^*\|pw'^*$. Therefore, $\Pr[b' = 0|b = 0] = \Pr[W_3]$. On the other hand, when the challenge bit of $\mathcal{B}$ is 1, $\mathcal{B}$ outputs 1 only if $\mathcal{A}_2$ succeeds in outputting a ciphertext c^* satisfying $\mathsf{PDec}(dk, c^*) = ID^*\|r'^*\|0^{|pw'^*|}$, and this situation is the same as Game 4. With almost the same discussion as above, we have $\Pr[b' = 0|b = 1] = \Pr[W_4]$. Therefore, $\mathsf{Adv}^{\mathsf{mIND\text{-}CCA}}_{\mathcal{B}}(k) = |\Pr[b' = b] - 1/2| = (1/2)|\Pr[b' = 0|b = 0] - \Pr[b' = 0|b = 1]| = (1/2)|\Pr[W_3] - \Pr[W_4]|$. Using this equality, and recalling the assumption that the underlying PKE scheme is mIND-CCA secure, we conclude that $|\Pr[W_3] - \Pr[W_4]|$ is negligible. This completes the proof of Lemma 13. □

Lemma 14. $\Pr[W_4] \leq O(q_{A_3})/|\mathcal{D}| + \varepsilon(k)$.

Proof. Note that in Game 4, the responses to the $\mathsf{Send}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{Auth}}$ queries contain no information of pw. However, if an adversary makes a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 3, c)$, the "server's output" part (i.e. $\top$ or $\bot$) leaks whether $pw = pw_s$ or not. However, other than this, no information about pw leaks. Since pw is chosen randomly from $\mathcal{D}$, the probability that $\mathcal{A}$ wins in Game 4 is at most $(q_{A_3} + 1)/|\mathcal{D}|$. Here, q_{A_3} is the number of $\mathsf{Send}_{\mathsf{Auth}}$ queries of the form $(sid, 3, M)$ by $\mathcal{A}$ and this is clearly equal or less than q_{S_A}. This completes the proof of Lemma 14. □

Lemmas 10 to 14 guarantee that the right hand side of the inequality (3) is negligible, and thus $\mathcal{A}$ has negligible advantage in the Imp-AA game. This completes the proof of Theorem 3. □

Theorem 4. *If F is a PRF and the PKE scheme is mIND-CCA secure*[11]*, then the proposed password reset protocol in Fig. 1 satisfies IR-AA security.*

Proof of Theorem 4. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an IR-AA adversary of the password reset protocol. Let q_{P_i} and q_{A_i} be the numbers of $\mathsf{Send}_{\mathsf{PRR}}$ query of the form (sid, i, M) and $\mathsf{Send}_{\mathsf{Auth}}$ query of the form (sid, i, M) by

[11] We need (full) CCA secure PKE for the security proof.

$\mathcal{A}_1$, respectively. Here, q_{P_i} and q_{A_i} are polynomials of the security parameter. Same as the case of Theorem 3, we only have to consider the three cases for $\mathsf{Send}_{\mathsf{PRR}}$ and $\mathsf{Send}_{\mathsf{Auth}}$ queries, respectively. Since the response of challenger is completely same as the case of Theorem 3, we omit them. Consider the following sequence of games.

Game 0. This is exactly the IR-AA game.

Game 1. This game proceeds in the same way as Game 0, except that the first messages r of S_A picked in the execution of PRR (in the response to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{PRR}}$ query) and Auth (either in the response to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{Auth}}$ query, or Auth in the challenge phase), are picked from $\{0,1\}^k \backslash \{r$'s that are already used $\}$, so that they are all distinct and never collide. As with Game 1 in Theorem 3, we introduce the list $\mathcal{R}$ and use it in the same manner.

Game 2. This game proceeds in the same way as Game 1, except that except that as in Game 2 considered in the proof of Theorem 1, we replace the output of $F(K,\cdot)$ with a random string.

Game 3. This game proceeds in the same way as Game 2, except the following points:

- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 2, r)$, then after calculating the response c and updating the tuple in L'_P as in the previous game, $\mathcal{B}$ also stores the value r used by in the same entry in L'_P (say, by appending r in the last of the entry). (Here, note that the value r used by $\mathcal{A}$ may not be identical to the value chosen and used by $\mathcal{B}$ as a response to the $\mathsf{Send}_{\mathsf{PRR}}$ query $(sid, 1, \phi)$ previously made by $\mathcal{A}$.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 3, c)$, then instead of decrypting c, $\mathcal{B}$ responds with $\top$ if $r = \widehat{r}$ holds, and $\bot$ otherwise, where r is the value used as a response to $\mathcal{A}$'s previous $\mathsf{Send}_{\mathsf{PRR}}$ query of the form $(sid, 1, \phi)$, and $\widehat{r}$ is the value contained in the previous $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{PRR}}$ query $(sid, 2, r)$. (Note that both of the values can be found in L'_P.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, then after calculating the response c and updating the tuple in L'_A as in the previous game, $\mathcal{B}$ also stores the value r in the same entry in L'_A (say, by appending r in the last of the entry). (Here, note that the value used by $\mathcal{A}$ may not be identical to the value chosen and used by $\mathcal{B}$ as a response to the $\mathsf{Send}_{\mathsf{Auth}}$ query $(sid, 1, \phi)$ previously made by $\mathcal{A}$.)
- If $\mathcal{A}$ sends a $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 3, c)$, then instead of decrypting c, $\mathcal{B}$ responds with $\top$ if $r = \widehat{r}$ and $pw = pw_s$ hold, and $\bot$ otherwise, where r is the value used as a response to $\mathcal{A}$'s previous $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 1, \phi)$, and $\widehat{r}$ is the value contained in the previous $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, both of which can be found in L'_A, pw is the value also contained in the entry in L'_A corresponding to sid, and pw_s is the value stored in L such that ID in L and ID in the entry corresponding to sid in L'_A and match.

Game 4. This game proceeds in the same way as Game 3, except the following points.

- If $\mathcal{A}_1$ issues $\mathsf{Send}_{\mathsf{PRR}}(sid, 2, r)$ queries, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_P and $(ID, *, *, rk, *, *)$ in the list L. Then, $\mathcal{B}$ executes $c^* \leftarrow \mathsf{PEnc}(pk, ID \| r \| 0^{|rk|+|pw|})$ and uses c^* instead of $c \leftarrow \mathsf{PEnc}(pk, ID \| r \| rk \| pw)$.]
- If $\mathcal{A}_1$ issues $\mathsf{Send}_{\mathsf{Auth}}(sid, 2, r)$ queries, $\mathcal{B}$ first finds the tuples $(sid, ID, pw, *, *, *)$ in the list L'_P, executes $c^{**} \leftarrow \mathsf{PEnc}(pk, ID \| r \| 0^{|pw|})$, and uses c^{**} instead of $c \leftarrow \mathsf{PEnc}(pk, ID \| r \| pw)$.

For $i \in \{0,1,2,3,4\}$, we define the event W_i as the event that $\mathcal{A}$ wins in Game i. The advantage of $\mathcal{A}$ is, by definition, $\mathsf{Adv}^{\mathsf{IR\text{-}AA}}_{\mathcal{A}}(k) = \Pr[W_0]$. We complete the proof by using the following inequality, and the upper bounds in the terms in the right hand side are shown in Lemmas 15 to 19.

$$\Pr[W_0] \leq \sum_{i=0}^{3} |\Pr[W_i] - \Pr[W_{i+1}]| + \Pr[W_4] \tag{4}$$

where the above equality can be derived by the triangle inequality.

Lemma 15. $|\Pr[W_0] - \Pr[W_1]|$ *is negligible.*

Proof. This proof follows closely to that of Lemma 10. Therefore, we omit it. □

Lemma 16. *If F is a PRF,* $|\Pr[W_1] - \Pr[W_2]|$ *is negligible.*

Proof. This proof follows closely to that of Lemma 11. Therefore, we omit it. □

Lemma 17. $\Pr[W_2] = \Pr[W_3]$.

Proof. This proof follows closely to that of Lemma 12. Therefore, we omit it. □

Lemma 18. *If the PKE scheme is mIND-CCA secure,* $|\Pr[W_3] - \Pr[W_4]|$ *is negligible.*

Proof. The proof of this lemma follows closely to that of Lemma 13, and thus we only highlight the differences. In the proof of Lemma 13, the mIND-CCA adversary uses its challenge query to respond to $\mathcal{A}$'s $\mathsf{Send}_{\mathsf{Auth}}$ query of the form $(sid, 2, r)$, while the mIND-CCA adversary in this proof can simply respond with $c \leftarrow \mathsf{PEnc}(pk, ID\|r\|pw)$ which is exactly what the challenger in Game 3 does. This guarantees that the mIND-CCA adversary perfectly simulates Game 3 (resp. Game 4) for $\mathcal{A}$ if its challenge bit is $b = 0$ (resp. $b = 1$). Furthermore, the mIND-CCA adversary outputs $b' = 0$ only if $\mathcal{A}$ succeeds in registering an illegal password (which can be detected by using the decryption oracle, as explained in the proof of Lemma 8). Put together, the mIND-CCA adversary's advantage is exactly $|\Pr[W_3] - \Pr[W_4]|$, which is negligible. This completes the proof of Lemma 13. □

Lemma 19. $\Pr[W_4]$ *is negligible.*

Proof. Note that in Game 4, the responses to the $\mathsf{Send}_{\mathsf{PRR}}$ queries contain no information of rk. Therefore, the probability that $\mathcal{A}$ succeeds the guess of rk (that is, the probability that $\mathcal{A}$ wins in Game 4) is negligible. This completes the proof of Lemma 19. □

Lemmas 15 to 19 guarantee that the right hand side of the inequality (4) is negligible, and thus $\mathcal{A}$ has negligible advantage in the IR-AA game. This completes the proof of Theorem 4. □

6 Implementation

To show the practical feasibility and test the performance of our protocol, we implemented a prototype of our protocol (excluding the communication part) in Python. We implemented our first scheme (Fig. 1), and all cryptographic operations are performed using the python cryptography toolkit (Pycrypto 2.6.1). We expect that performance can be improved by using other appropriate libraries (e.g. Number Theory Library [29]).

We implement our scheme on a laptop computer (Windows 7 (64bit), Core i7-M640 2.80GHz, 8GB RAM). Table 1 shows the computational costs of our first protocol. We implement $F(K, x)$ in our construction as $H(K\|x)$ where H is SHA-256. This means that we regard SHA-256 hash function as a random oracle. We adopt RSA-OAEP(2048bit) as a PKE scheme. We omit to calculate the the execution time of SSetup algorithm because this operation is executed only once and does not rely on the participation of users. PRR1(Auth1) denotes the first server side execution, PRR2(Auth2) denotes the first client side execution, and PRR3(Auth3) denotes the second server side execution. We set the length of ID as 10 alphanumeric characters, randomnesses that is used in the first pass of PRR/Auth as 128bit, and passwords that is used in the PRR/Auth as 16 alphanumeric characters. We execute each algorithm 10000 times, and show its average time in Table 1.

Although the executions of PRR3 and Auth3 (the dominant part of these algorithms is decryption of RSA-OAEP(2048bit)) take more time than other algorithms, even these two algorithms need less than 50ms. Therefore, we can see that our scheme is suitable for practical use.

7 Conclusion

In this paper, we proposed a model, security definitions, and a construction of a provably secure password reset protocol. Our generic construction is based on a PRF and PKE. We can construct a number of concrete password reset protocols from this generic construction.

Countermeasures against server breach is one of the future works. It is interesting to introduce a server decentralization to the password reset protocol like Camenisch et al.'s decentralized password verification protocol [9]. Another future work is to propose another model and security definitions. In this paper, we proposed a model that introduces a reset key and two security definitions (impersonation/illegal registration). However, there may exist a more appropriate model and security definitions depending on the system context. We believe that this paper opens a door to rigorous security treatment of password reset protocols and that our proposed model/schemes can be a foundation there.

Table 1. Performance of our scheme (computational cost)

Name of algorithms	Execution time [μs]
RKG	2.61
PRR1	2.68
PRR2	3.67×10^3
PRR3	4.67×10^4
Auth1	2.71
Auth2	3.62×10^3
Auth3	4.67×10^4

References

1. M. Bellare, A. Boldyreva, and S. Micali. Public Key Encryption in a Multi-user Setting: Security Proofs and Improvements. In *Proc. of EUROCRYPT 2000*, LNCS 1807, pp. 259–274, 2000.
2. M. Bellare, D. Pointcheval, P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. In *Proc. of EUROCRYPT 2000*, LNCS1807, pages. 139–155, 2000.
3. M. Bellare, C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *J. Cryptology*, Volume 21, No.4, pages. 469–491, 2008.
4. M. Bellare, C. Namprempre, G. Neven. Security Proofs for Identity-Based Identification and Signature Schemes. In *J. Cryptology*, Volume 22, No.1, pages. 1–61, 2009.
5. S.M. Bellovin, M. Merritt. Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks. In *Proc. of IEEE Symposium on Security and Privacy 1992*, pages. 72–84, 1992.
6. D. Boneh, M.K. Franklin. Identity-Based Encryption from the Weil Pairing. In *Proc. of CRYPTO 2001*, LNCS2139, pages. 213–229, 2001.
7. J. Bonneau, E. Bursztein, I. Caron, R. Jackson, M. Williamson. Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google. In *Proc. of WWW 2015*, pages. 141–150, 2015.
8. V. Boyko, P.D. MacKenzie, S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In *Proc. of EUROCRYPT 2000*, LNCS1807, pages. 156–171, 2000.
9. J. Camenisch, A. Lehmann, G. Neven Optimal Distributed Password Verification. In *Proc. of CCS 2015*, pages. 182–194, 2015.
10. R. Canetti, O. Goldreich, S. Goldwasser, S. Micali. Resettable zero-knowledge (extended abstract). *STOC 2000*, pp.235–244, 2000.
11. R. Cramer, G. Hanaoka, D. Hofheinz, H. Imai, E. Kiltz, R. Pass, A. Shelat, V. Vaikuntanathan. Bounded CCA2-Secure Encryption. In *Proc. of ASIACRYPT 2007*, LNCS4833, pages. 502–518, 2007.
12. G.R. Doddington, W. Liggett, A.F. Martin, M.A. Przybocki, D.A. Reynolds. SHEEP, GOATS, LAMBS and WOLVES: a statistical analysis of speaker performance in the NIST 1998 speaker recognition evaluation. In *Proc. of ICSLP 1998*, 1998.
13. Y. Dodis, J. Katz, S. Xu, M. Yung. Key-Insulated Public Key Cryptosystems. In *Proc. of EUROCRYPT 2002*, LNCS2332, pages. 65–82, 2002.
14. R. Geambasu, T. Kohno, A.A. Levy, H.M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *Proc. of Usenix Security 2009*, pages. 299–316, 2009.
15. S. Goldwasser, S. Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In *Proc. of STOC 1982*, pages. 365–377, 1982.
16. A. Hang, A.D. Luca, M. Richter, M. Smith, H. Hussmann. Where Have You Been? Using Location-Based Security Questions for Fallback Authentication. In *Proc. of SOUPS 2015*, 2015.
17. M. Jakobsson, E. Stolterman, S. Wetzel, L. Yang. Love and Authentication. In *Proc. of CHI 2008*, pages. 197–200, 2008.
18. A.K. Jain, A. Ross, S. Prabhakar. An Introduction to Biometric Recognition. In *IEEE Transactions on Circuits and Systems for Video Technology*, Volume 14, No.1, pages. 4–20, 2004.
19. A. Javed, D. Bletgen, F. Kohlar, M. Dürmuth, J. Schwenk. Secure Fallback Authentication and the Trusted Friend Attack. In *Proc. of ICDCSW 2014*, pages. 22–28, 2014.
20. M. Just, D. Aspinall. Personal Choice and Challenge Questions: A Security and Usability Assessment. In *Proc. of SOUPS 2008*, 2008.
21. J. Katz, V. Vaikuntanathan. Round-Optimal Password-Based Authenticated Key Exchange. In *J. Cryptology*, Volume 26, No.4, pages. 714–743, 2013.

22. K. Kurosawa, K. Heng. From Digital Signature to ID-based Identification/Signature. In *Proc. of PKC 2004*, LNCS2947, pages. 248–261, 2004.
23. D. Ma, G. Tsudik. A New Approach to Secure Logging. In *Proc. of DBSEC 2008*, LNCS5094, pages. 48–63, 2008.
24. A. Rabkin. Personal Knowledge Questions for Fallback Authentication: Security Questions in the Era of Facebook. In *Proc. of SOUPS 2008*, 2008.
25. R.W. Reeder, S. Schechter. When the Password Doesn't Work: Secondary Authentication for Websites. In *IEEE Security and Privacy*, Volume 9, No.2, pages. 43–49, 2011.
26. S.E. Schechter, A.J.B. Brush, S. Egelman. It's No Secret. Measuring the Security and Reliability of Authentication via "Secret" Questions. In *Proc. of IEEE Symposium on Security and Privacy 2009*, pages. 375–390, 2009.
27. S. Schechter, R.W. Reeder. 1+1=You: Measuring the Comprehensibility of Metaphors for Configuring Backup Authentication. In *Proc. of SOUPS 2009*, 2009.
28. A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proc. of CRYPTO 1984*, LNCS196, pages. 47–53, 1984.
29. NTL: A Library for doing Number Theory, http://www.shoup.net/ntl/
30. L. Standing, J. Conezio, R.N. Haber. Perception and Memory for Pictures: Single-trial Learning of 2500 Visual Stimuli. In *Psychonomic Science*, Volume 19, Issue.2, pages. 73–74, 1970.
31. M. Zviran, W.J. Haga. User Authentication by Cognitive Passwords: An Empirical Assessment. In *Proc. of JCIT 1990*, pages. 137–144, 1990.