UC-secure Two-Server Password-Based Authentication Protocol and Its Applications

Lin Zhang
Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences
Beijing, P.R. China
zhanglin@tca.iscas.ac.cn

Zhenfeng Zhang
Trusted Computing and Information Assurance Laboratory
Institute of Software, Chinese Academy of Sciences
Beijing, P.R. China
zfzhang@tca.iscas.ac.cn

Xuexian Hu
State Key Laboratory of Mathematical Engineering and Advanced Computing
Zhengzhou, P.R. China
xuexian_hu@hotmail.com

ABSTRACT

A two-server password-based authentication (2PA) protocol is a special kind of authentication primitive that provides additional protection for the user's password. Through a 2PA protocol, a user can distribute his low-entropy password between two authentication servers in the initialization phase and authenticate himself merely via a matching password in the login phase. No single server can learn any information about the user's password, nor impersonate the legitimate user to authenticate to the honest server.

In this paper, we first formulate and realize the security definition of two-server password-based authentication in the well-known universal composability (UC) framework, which thus provides desirable properties such as composable security. We show that our construction is suitable for the asymmetric communication model in which one server acts as the front-end server interacting directly with the user and the other stays backstage.

Then, we show that our protocol could be easily extended to more complicate password-based cryptographic protocols such as two-server password-authenticated key exchange (2PAKE) and two-server password-authenticated secret sharing (2PASS), which enjoy stronger security guarantees and better efficiency performances in comparison with the existing schemes.

CCS Concepts

•**Security and privacy** → **Authentication;** *Security protocols;* Access control;

Keywords

Universal composability, two-server password-based authentication, key exchange, secret sharing

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China
© 2016 ACM. ISBN 978-1-4503-4233-9/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2897845.2897872

1. INTRODUCTION

To date, password-based protocols have already widely been deployed throughout the network, benefiting from password's convenience and low-cost. Without the aid of additional auxiliary equipments, participants only share low-entropy passwords remembered by humans to finish the designated cryptographic tasks over a hostile communication environment. However, these protocols are inherently vulnerable to the off-line dictionary attack where a passive adversary exhaustively enumerates possible passwords, and attempts to get the matching one to the observed transcripts of instances. Additionally, the on-line dictionary attack, an inevitable one by cryptographic ways because of relatively small dictionary space, allows adversaries to login repeatedly with various password attempts. Thus, how to resist off-line attacks and restrict to adversaries eliminating at most one password per protocol execution, has become the basic security requirement of password-based protocols.

Although many schemes are carefully designed to avoid off-line dictionary attacks, they just pay attention to the situation where the adversary can only attack the communication link, without considering the security when servers are corrupted. Indeed, server compromise causes the growing threat to personal passwords in the real network, and there's no sign of let-up [21, 33]. In the single-server scenario, the server always stores the "data file" containing the secrets of all the clients in the system. An attacker may obtain all the passwords once the server is compromised. If unfortunately, even worse, the adversary could freely login victims' accounts, and perform other malicious behaviors with legal status, which inevitably gives rise to further leakage of privacy protected by passwords. Even though the hashed or salted passwords (or other information related to them) are stored on servers instead [4, 27, 22], the brute-force attack still helps the adversary to try guesses repeatedly for the password and check them against the values from password files. Since 2012 a computer cluster boasting 25 AMD Radeon graphics cards has already made 350 billion password guesses per second, and all eight-character passwords fall in hours [29]. Hence, the password cracking is severe and unavoidable during a data breach.

Distributed Password-Based Authentication (DPA) addresses this concern effectively. As an economic and practical solution, it distributes the capability to verify passwords and trust over multiple servers. More specifically, it makes the password or relevant information split into pieces and stored on different servers, respectively. Once a user accesses the target servers, they verify the password attempt via their corporation so as to authenticate this specified entity. Correlatively, Camenisch et al. [9] provided an efficient single-round distributed password verification protocol, realizing the UC-security [11] against transient corruption. Not entirely the same as DPA, It emphasises on the security of the password verification process among servers rather potential disclosure of passwords in the communication channels between the user and servers. Besides independent interest, DPA can be viewed as an authentication modular of threshold password-authenticated key exchange [28, 5, 35] and password-authenticated secret sharing [10, 8, 6].

Threshold Password-Authenticated Key Exchange was introduced by Ford and Kaliski [19]. In a (t, n) T-PAKE protocol, a user with a single password suffices to establish n session keys with n servers separately, if only he possesses a matching one. Meanwhile, the coalition of no more than t malicious servers can learn nothing about the password. It significantly mitigates the damages caused by server compromise. Subsequent to their work, a number of efficient schemes [24, 5, 34, 36] were given. Nevertheless, they all give no formal proof of security in accepted models. MacKenzie et al. [28] extended the BPR model [2] into multi-server setting, and put forth the first provably secure scheme under the derived security model. Later, two-server schemes [26, 35] in the standard model were followed.

Password-Authenticated Secret Sharing (in case of t-out-of-n) is deemed to allow a user to distribute his secret protected by a password among n servers, and later, with a matching password attempt, to reconstruct the secret by interacting with any $t + 1$ servers included in the initial set of n servers. In real-world applications, PASS protocols can make private data conveniently synchronized over the cloud among ones' personal devices. An earlier way to realize the functionalities above, such as [19, 24], relies on the Threshold PAKE protocol, and transmits the secrets via the secure channels they have established. However, the composition of TPAKE and secure channel is not as efficient and concise as direct construction of PASS. The real sense of PASS was introduced and formalized by Bagherzandi et al. [1] (which is called as "PPSS"). The first ideal functionality for two-server PASS and the concrete proposal in the UC framework were put forward by Camenisch et al. [10] in the PKI setting, then followed by the t-out-of-n one [8] and the adaptively secure one [6]. Afterwards, Jarecki et al.'s protocol [25], based on Verifiable Oblivious PRF, allowed the construction of the round-optimal scheme with minimal overhead known, and achieved the game-based security.

1.1 Our Contributions

Firstly, we suggest a two-server password-based authentication protocol in the password-only setting (even between servers) where parties do not have to remember others' public keys, capturing the explicit verification of users' passwords by the threshold way. Our scheme provides explicit notifications to both servers about the success or failure of an authentication attempt, and prevents the adversary from learning the password attempt or honest servers' shares on condition that at least one server is honest. Therefore it is nicely compatible with the throttling mechanism that temporarily blocks user accounts for the sake of protection as they successively fail several login attempts or some servers' abnormal behaviors are observed. This functionality is actually of great importance to security and practicability. Otherwise, without explicit validation, the corrupted server could possibly treat the other uncorrupted one as a black-box oracle to check the correctness of attempts—i.e. an online attack without awareness. More realistically, unrestricted login requests from the adversary in the same username could exhaust computational resources of servers.

In security respect, we prove that our scheme satisfies UC-security against the static adversary, based on the decisional Diffie-Hellman assumption and the random-oracle model [3]. Compared with the game-based security models [2, 28, 1], the universal composablity framework has conspicuous advantages in composition property and distribution of passwords. Indeed, They mainly arise from the stipulation in this model that upper entity (known as the "environment") of the target session, also as a distinguisher, is capable of arbitrarily invoking protocol instances, providing private inputs to participants. To avoid repetition, more descriptive details about UC framework are presented in Section 2.4.

In terms of setting, our scheme abandons the PKI assumption, even between servers. It can inherently deter the phishing attack. That is, If, an undesired case, an honest party obliviously communicates with malicious peers in default of certified public key, which directly results in the exposure of confidential information. We instructs the user to encrypt randomized quotient of passwords instead of plain password shares. In addition, the PKI seemly provides enough convenience to construct protocol schemes. Nevertheless, as a user wants to communicate with servers, he has to access to root certification authorities for their public keys, and explicitly authenticate servers. On the minus side, it adds to users the burden of checking key validity. For practical deployment, the design should evade any use of PKI model in the password-based system. Otherwise, to some degree the benefits of passwords would be counteracted.

From an architecture perspective, two-server password-based protocols come in two types: symmetric and asymmetric. The symmetric type, such as [26, 10, 36, 35, 6], allows two servers equally to contribute to authentication. Additionally, and they are both exposed to users or external adversaries, and this type demands for the synchronization at the user side due to the simultaneous interactions with two servers. In the asymmetric type, such as [5, 34], only the front-end server is public to the user and responsible for delivering the messages, while the back-end server as a reserved aider, stays behind the scenes in relatively safe status. From a security point of view in the real network, the latter is obviously superior to the former. Without use of the PKI assumption, our scheme supports the asymmetric architecture, and, unlike [34], there no exists unequal levels of trust in both servers. An malicious front-end server derives no knowledge from illegal behaviors, like impersonating or tampering with messages from the other parties, but the result of authentication, when communicating with the honest back-end one.

After proposing the new scheme above, our second main contribution is considering the 2PA protocol as a core building block to extend to other contexts, so that we give the practical yet UC-secure instantiations of 2PAKE and 2PASS schemes. Incidentally, they also inherit the advantages of no PKI requirements for any parties. More precisely, by virtue of smooth projective hash functions (SPHF) [15], as implicit proofs of membership for certain languages, our 2PAKE UC-realizes the corresponding ideal functionality we provide, and it's the first provably secure one in the UC framework. We make key establishment materials pass along with authentication information, and bind to them via a simulation-sound NIZK, to achieve that end. And our 2PASS analogously employs SPHF but as proofs of legal servers rather than session keys of secure channels. That technique may eliminate the unnecessary utilizations of zero-knowledge proofs. As far as we know, the resulting protocol also has a better computation complexity than the other UC-secure PASS schemes. Comparisons of some existing schemes are shown in Table 1 and Table 2, respectively.

Table 1: Comparison of two-server PAKE schemes

scheme	struc.	secur.[1]	setup	msgs[2]	com.user[3]	com.sers
KMT+[26]	sym	game	CRS	6	15	70|70
YLW[36]	sym	none	RO	6	4	5|5
YHB[35]	sym	game	ID	7	23	32|32
YDH[34]	asym	none	RO	10	5	6|3
Ours	asym	UC	RO	7	12	20|20

[1] The "secur." column enumerates the accepted security models in which schemes are proved;
[2] The "msgs" column counts the total number of transmitted messages among parties.
[3] The last two columns counts exponentiations in a prime-order group performed by the client and servers, respectively.

Table 2: Comparison of PASS schemes in RO model

scheme	range	secur.	setting	msgs	com.user	com.sers
BJSL[1]	(t,n)	game	PKI	3	$8t+17$	16
JKK[25]	(t,n)	game	CRS	2	$2t+3$	3
CLLN[8]	(t,n)	UC	CRS	10	$14t+24$	$7t+28$
CLN[10]	$(1,2)$	UC	PKI	8	19	26 | 30
Ours	$(1,2)$	UC	CRS	8	18	18

2. PRELIMINARIES

In this section, we first briefly recall cryptographic primitives two kinds of public-key encryption schemes, smooth projective hashing and simulation-sound non-interactive zero-knowledge proofs used in the construction of our schemes, and then give a review of the UC framework [11].

2.1 Public-Key Encryption Schemes

A CPA-secure public-key encryption scheme consists of three algorithms $(\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$. The key generation algorithm kg generates the pair (pk, sk). The encryption algorithm takes as input pk, a message m, the randomness w and outputs a ciphertext $C{:=}\mathsf{enc}_{pk}(m;w)$, while the decryption takes as input sk, a ciphertext C and outputs $m{:=}\mathsf{dec}_{sk}(C)$. We require it to have the indistinguishability under chosen plaintext attack. The ElGamal cryptosystem [17] that meets such requirements is needed in this literature.

A labeled CCA2-secure public-key encryption scheme is also defined by three algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$. Its description of algorithms is similar to CCA2, except that the Enc and Dec algorithms have an additional input parameter, referred to as a *label*, and the Dec consists progress of verifying the ciphertext. More precisely, the Dec will not decrypt the ciphertext, unless the input *label* is consistent with one used in the Enc algorithm, and the ciphertext is verified, otherwise the algorithm outputs a failure symbol $\perp$. For the CCA2-security, the adversary is allowed to query the decryption oracle on the 2-tuples composed of the ciphertexts and labels that are different from the challenge pair.

2.2 Smooth Projective Hash Functions

The notion of smooth projective hash functions was introduced by Cramer and Shoup [15]. Here, we focus on our eventual application, similarly to [23], and show a description based on the ElGamal encryption scheme.

For some ElGamal scheme $(\mathsf{kg}, \mathsf{enc}, \mathsf{dec})$ with respect to public key pk, Let X be the range of the encryption algorithm, and define the language $L_{pk,m}{:=}\{C|\exists w : C{:=}\mathsf{enc}_{pk}(m;w)\}$ for a fixed plaintext $m \in D$. Obviously, $L_{pk,m} \subset X$.

Formally, a *smooth projective hash function* for the language $L_{pk,m}$ is a keyed family of functions mapping elements in the ciphertext space X to the set Π, along with a *projection function* α: $K \to S$, where K is the *hash key* space, and S is the *projected key* space, satisfying:

1. There exist four algorithms for (1) uniformly sampling the hash key $hk \leftarrow K$, (2) deriving the projected key $hp := \alpha(hk)$ for $hk \in K$, (3) computing the hash value $\mathsf{Hash}(hk, C, m)$, for $C \in X$ and $m \in D$, and (4) computing the projected hash value $\mathsf{ProjHash}(hp, C, m, w)$, where w is the witness $C \in L_{pk,m}$.
2. The *correctness* guarantees that if $C \in L_{pk,m}$ with w a witness of this membership, then the results given by these two hash algorithms are equivalent, i.e., $\mathsf{Hash}(hk, C, m) = \mathsf{ProjHash}(hp, C, m, w)$.
3. The *smoothness* assures that if $C \notin L_{pk,m}$, the following two distributions are statistically indistinguishable: $\{(hp, H)|hk \leftarrow K; hp := \alpha(hk); H := \mathsf{Hash}(hk, C, m)\}$, $\{(hp, H)|hk \leftarrow K; hp := \alpha(hk); H \leftarrow \Pi\}$.

2.3 Simulation-Sound NIZK

Simulation-sound non-interactive zero-knowledge proofs introduced by [30, 16], are used to prove some certain relations among the ciphertexts in this paper. The simulation-soundness, as the extended property of soundness, guarantees that a malicious prover cannot give a convincing proof for any new invalid statement, even if it has seen polynomial many simulated proofs of invalid statements of its choice. Additionally, we need the proof system to be zero-knowledge that the adversary cannot distinguish the simulated proof from a real one. In our schemes below, we use $\mathsf{NIZK}\{(w) : (w, v) \in R_\Phi\}$ to denote the proof that a predicate $\Phi(v) = 1$ for a public value v and the corresponding witness w. Moreover, the *label* containing the open information can be appended, on account of a labeled zero-knowledge proof that is bound to a certain context. The instantiations of proofs are provided in Section 3.3.

2.4 Universal Composability Framework

Universal composability [11] is the definition of secure computation that considers an execution of a protocol in the setting involving an *environment* $\mathcal{Z}$, an adversary and parties. This framework focuses on two worlds—the real-world and the ideal-world. $\mathcal{Z}$'s whose aim is to differentiate between the two worlds, provides the inputs to the parties and views their outputs. On one hand, as usual, the real-world consists of participants of the protocol and an *adversary* $\mathcal{A}$ who controls the communication channel and potentially attacks them. On the other hand, in the ideal-world, there exists an entirely trusted entity $\mathcal{F}$ called *ideal functionality*, and participates of the protocol simply hand their inputs to $\mathcal{F}$. The *ideal adversary* $\mathcal{S}$ directly interacts with $\mathcal{F}$, and their communication essentially models the information it can obtain and its abilities to attack the protocols. Namely, the functionality describes the security goals we expect. Intuitively, the adversary, with a variety of means of attacks, should not learn more information than the functionality's outputs to it. Thus, security requires that, for any adversary $\mathcal{A}$ attacking a protocol ρ, there exists an ideal adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish the case that it is interacting with $\mathcal{A}$ and parties in the real-world, and the case which it is interacting with $\mathcal{S}$ and a functionality $\mathcal{F}$ in the ideal-world. If so, we say that ρ *UC-realizes* $\mathcal{F}$. From the point of view of the environment, the real-world protocol is at least as secure as the ideal-world one.

UC framework brings several fundamental advantages compared with game-based security models [2, 4, 28, 1]. The widely recognized one is that the executions of UC-secure protocols remain secure in arbitrary, possibly malicious network environments—essentially what one should expect from security protocols deployed in the real-world. However, other security models generally just concern about stand-alone or non-concurrently settings. Specifically, protocols proven UC-secure satisfy strong composability properties: (1) they remain secure even though many and different protocol instances (secure or not) may run concurrently, and (2) the powerful universal composition theorem guarantees that they can be securely used as a sub-routine protocol of other UC protocols. With respect to the case of password-based protocols, we would get some further benefits from the security notions in the UC framework. The stand-alone security models, such as the BPR model [2], are somewhat limited in that they assume that passwords are randomly chosen from some pre-determined, independent and known distribution, which rarely holds in the real deployment[1]. More than that, they fail to depict the cases where the honest participants with incorrect but related passwords interacts with others—when a user obliviously types errors. Rather, the UC framework can handle arbitrary password distribution, even if related passwords are used, or peers have different ones, since their inputs are all up to the environment, which elegantly captures the requirements of a more realistic scene.

3. TWO-SERVER PASSWORD-BASED AUTHENTICATION PROTOCOL

In this section, we'll begin by presenting a detailed description of the security definitions in the UC framework, followed by a corresponding scheme to realize it.

3.1 Ideal Functionality

Aiming at formulating a security definition of the two-server password-based authentication (2PA) that guarantees stronger security properties such as protocol composability and fit for arbitrary password distribution, we present an ideal functionality for 2PA in the UC framework. Our formulation is inspired by the UC security notions of traditional PAKE [12, 23] as well as the security properties of existing distributed password-based protocols [10, 8]. It, denoted as $\mathcal{F}_{2\text{PA}}$, is described in Figure 1.

In the ideal-world, a 2PA protocol operates in a setting with a user $\mathcal{U}$, two servers $\mathcal{P}_1$, $\mathcal{P}_2$, an adversary $\mathcal{S}$, the environment $\mathcal{Z}$, and the ideal functionality $\mathcal{F}_{2\text{PA}}$. The session identifier tuple $sid := (username, \mathcal{P}_1, sid')$ for $sid' \in \{0,1\}^*$, with sub-session identifiers *ssid* mentioned below, is provided by $\mathcal{Z}$, and imposed to be globally unique so that concurrent sessions can be distinguished. Note that the *sid* contains no information about the identity of $\mathcal{P}_2$. This is defined to support the asymmetric communication model where the user does not directly connect with the back-end server, or be aware of him in a protocol instance. In order to avoid the repeated representation, we assume that the ideal functionality only cares about the first query or inputs for each *ssid*, and straightly ignores the subsequent ones.

The Initialization query models the process that a user $\mathcal{U}$ with *username* chooses p' to create an account on the front-server $\mathcal{P}_1$ and the back-end one $\mathcal{P}_2$, and distributively stores the password shares in two servers. To simplify descriptions of the ideal functionality, we assume that only in this initialization phase, are all parties honest, and the user temporarily knows his peers $\mathcal{P}_1$ and $\mathcal{P}_2$. Actually, this idealized simplification hardly impairs the security of the subsequent protocol execution phase in which we allow the adversary to attack actively. Once receiving the query $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2, p')$, $\mathcal{F}_{2\text{PA}}$ locally takes down this intact input as a record. Then $\mathcal{F}_{2\text{PA}}$ shall unhesitatingly label this record fresh. At last, the functionality generates a public delayed output[2] to both servers.

Though the NewSession queries, these parties begin new login sessions with the expected ones. On one hand, $\mathcal{F}_{2\text{PA}}$ creates a corresponding record for user $\mathcal{U}$'s password attempt p (that may not be p') referred to him, and labels this record fresh or corrupted according to his current state. In this paper, we just take into account the static corruption—the adversary $\mathcal{S}$ could selectively corrupt some of the participants, but only prior to the beginning of a protocol execution, namely, before any one of the NewSession queries from $\mathcal{Z}$ is sent. From corruption on, the adversary will not only obtain their inputs, but also fully control the rest of their behaviors. On the other hand, once both servers join the specified session, $\mathcal{F}_{2\text{PA}}$ checks up and recovers the Initialization record for the corresponding *sid* related to them. If such a record exists, the functionality generates a NewSession record collectively owned by $\mathcal{P}_1$ and $\mathcal{P}_2$. Otherwise, $\mathcal{F}_{2\text{PA}}$ puts these queries from servers on hold. Note that $\mathcal{F}_{2\text{PA}}$ is required to mark it corrupted, only in the case that they are both corrupted, and it is labeled as fresh in other cases. It is such defined since we hope that a single dishonest server is prevented from learning the knowledge about the password. And we still try to guarantee that the fresh

[1]In 2014, Wang et al. [31] had revealed that passwords are not uniformly distributed but actually follow a Zipf's law, and more details can be referred to Fig.3 of [32].

[2]As defined in [11], we say that an ideal functionality $\mathcal{F}$ sends a delayed output v to a party $\mathcal{P}$, if it first sends to the adversary a message that it is ready to generate an output to $\mathcal{P}$. If the output is public, then the value v is included in the message to the adversary. If the output is private, then v is not mentioned in this message

Upon input $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2, p')$ **from** $\mathcal{U}$**:**
$\mathcal{F}$ creates a record $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2, p')$, and labels it fresh. Then it sends a public delayed output $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2)$ to $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively.

Upon input $(\mathsf{NewSession}, sid, ssid, p)$ **from** $\mathcal{U}$**:**
$\mathcal{F}$ sends $(\mathsf{NewSession}, sid, ssid, \mathcal{U})$ to $\mathcal{S}$. Then $\mathcal{F}$ creates a record $(\mathsf{NewSession}, sid, ssid, \mathcal{U}, p)$, and labels it fresh for the honest user. If $\mathcal{U}$ has been corrupted, $\mathcal{F}$ labels the record corrupted instead, and relay this query to $\mathcal{S}$.

Upon input $(\mathsf{NewSession}, sid, ssid)$ **from** $\mathcal{P}_i$**:**
$\mathcal{F}$ notifies $\mathcal{S}$ by sending $(\mathsf{NewSession}, sid, ssid, \mathcal{P}_i)$. If there's a fresh record $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2, p')$, it has received two NewSession queries from servers, and at least one is honest, $\mathcal{F}$ creates a fresh record $(\mathsf{NewSession}, sid, ssid, \mathcal{P}_1, \mathcal{P}_2, p')$. If both servers are corrupted, it's marked as corrupted, and sent to $\mathcal{S}$.

Upon receiving $(\mathsf{TestPwd}, sid, ssid, \mathcal{T}, \widehat{p})$ **from** $\mathcal{S}$**, where** $\mathcal{T} = \{\mathcal{P}_1, \mathcal{P}_2\}$ **or** $\mathcal{U}$**:**
If $\mathcal{T} = \{\mathcal{P}_1, \mathcal{P}_2\}$, there exists a fresh record $(\mathsf{NewSession}, sid, ssid, \mathcal{P}_1, \mathcal{P}_2, p')$ (resp. $(\mathsf{NewSession}, sid, ssid, \mathcal{U}, p)$ and $\mathcal{T} = \mathcal{U}$), then do: If $p' = \widehat{p}$ (resp. $p = \widehat{p}$), $\mathcal{F}$ relabels it compromised and replies to $\mathcal{S}$ with "correct guess". Otherwise, $\mathcal{F}$ relabels it interrupted and replies with "wrong guess".

Upon receiving $(\mathsf{Result}, sid, ssid, \mathcal{R})$ **from** $\mathcal{S}$**, where** $\mathcal{R} \in \{\mathcal{U}, \mathcal{P}_1, \mathcal{P}_2\}$**:**
If there is a stored record for $\mathcal{R}$, then:
• If this record is compromised, then $\mathcal{F}$ gives a public delayed output $(\mathsf{Result}, sid, ssid, \mathsf{succ})$ to $\mathcal{R}$.
• If his record is interrupted, or the other record (if it exists) is labeled as interrupted or compromised, or $p' \neq p$, then $\mathcal{F}$ sends a public delayed output $(\mathsf{Result}, sid, ssid, \mathsf{fail})$ to $\mathcal{R}$.
• If there exists the other record, and each record is fresh or corrupted with matching passwords, then $\mathcal{F}$ sends a public delayed output $(\mathsf{Result}, sid, ssid, \mathsf{succ})$ to $\mathcal{R}$.
Either way, $\mathcal{F}$ marks this record completed.

Figure 1: Ideal functionality $\mathcal{F}_{\text{2PA}}$

record with a "defect" can successfully finish the task despite malicious behaviors from the malicious server—robustness.

In the unauthenticated channel, an adversary should be capable of delaying or revising the messages from the honest user or servers, and impersonating them to take part in the protocol instance with its own attempt. To model that, we introduce the TestPwd query furnished by $\mathcal{S}$ to capture the adversary's attack ability from the angles of the disguised user and servers. Particularly, though only one server is a fake while the other honestly runs, we still deem it to be a TestPwd query. A record is relabeled as compromised if the adversary makes a correct password guess, and it is relabeled as interrupted instead otherwise. In either case, $\mathcal{S}$ explicitly takes the verification results for granted in our model.

In the end, the adversary $\mathcal{S}$ decides at which point the results of the authentication are delivered to participants through the Result queries. Upon receiving the queries, $\mathcal{F}_{\text{2PA}}$ provides the results to them according to the labels of records and the facts that whether the passwords are matching or not. When one record has been compromised, i.e. $\mathcal{S}$ has a correct guess, $\mathcal{F}_{\text{2PA}}$ should output a successful notification succ to the receiver $\mathcal{R}$, so do both records (no matter each is fresh or corrupted) with matching passwords. The adversary can always "hijack" the original party query, and substitute it with its own, which gives rise to failure of his connection. Hence, one gets a fail result, as long as the other record has been interrupted or compromised, or his record has a non-matching password with the other. Once $\mathcal{F}_{\text{2PA}}$ sending a Result output to $\mathcal{R}$, his NewSession record is marked as completed to avoid constant on-line guessing attacks from $\mathcal{S}$ even after the authentication has ended.

Remarkably, in the UC framework, as per the formalism of [11], assume that multiple protocol instances are running concurrently. The globally unique session identifiers *ssid* are generally used to differentiate among these instances. As the case in the real-world, multiple execution instances often invoke the same CRS. Roughly, we have to consider the multi-session extension $\widehat{\mathcal{F}}_{\text{2PA}}$ through the JUC theorem [14]. We refer to [12, 10] for more discussions.

3.2 Our Scheme

Now we present our scheme for realizing $\widehat{\mathcal{F}}_{\text{2PA}}$ above, relying on the common reference string functionality $\mathcal{F}_{\text{CRS}}$ [13], the secure message transmission functionality $\mathcal{F}_{\text{SMT}}$ [11] and the random oracle model. We illustrate the protocol in Figure 2. Assume that $\mathcal{F}_{\text{CRS}}$ describes a cyclic group $\mathbb{G}$ of prime order q and generator g generated through an algorithm of parameters generation by a security parameter κ, together with two ElGamal public keys h' and h randomly sampled from group $\mathbb{G}$ for which the corresponding discrete logarithms are unknown. $\mathcal{F}_{\text{SMT}}$ (only employed in the initialization phase) guarantees that the transmission is ideally authenticated, and in addition that the adversary has no access to the contents of the transmitted message. As for random oracles, we only use them to generate SS-NIZKs.

Once activated with the input $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2, p')$ from the environment $\mathcal{Z}$, where $p' \in \mathbb{G}$, the user $\mathcal{U}$ uniformly chooses p_1 from the group $\mathbb{G}$, and splits the password p' into p_1 and p_2 such that $p' := p_1 p_2$. Then $\mathcal{U}$ sends p_1, p_2 to the server $\mathcal{P}_1$, $\mathcal{P}_2$ through $\mathcal{F}_{\text{SMT}}$, respectively. It is understand-

	$\mathcal{U}(p)$		$\mathcal{P}_1(p_1)$		$\mathcal{P}_2(p_2)$
Step 1.	$r \leftarrow \mathbb{Z}_q^*; u := g^r$ $v := h'^r p$	$\xrightarrow{u, v}$			
Step 2.			$r_1 \leftarrow \mathbb{Z}_q^*; u_1 := g^{r_1}$ $v_1 := h^{r_1} p_1$	$\xrightarrow{u, v, u_1, v_1}$	
Step 3.		$\xleftarrow{u_1, v_1, u_2, v_2}$		$\xleftarrow{u_2, v_2}$	$r_2 \leftarrow \mathbb{Z}_q^*; u_2 := g^{r_2};$ $v_2 := h^{r_2} p_2$
Step 4.	$s \leftarrow \mathbb{Z}_q^*; A := h^s$ $B := (v_1 v_2 / p)^s$	$\xrightarrow{A, B, \pi^\ell}$			
Step 5.			if π^ℓ fails, abort; $s_1 \leftarrow \mathbb{Z}_q^*; A_1 := A^{s_1}$ $B_1 := B^{s_1} A^{-r_1 s_1}$	$\xrightarrow{A, B, \pi^\ell, A_1, B_1, \pi_1^{\ell_1}}$	
Step 6.				$\xleftarrow{A_2, B_2, \pi_2^{\ell_2}}$	if π^ℓ or $\pi_1^{\ell_1}$ fails, abort $s_2 \leftarrow \mathbb{Z}_q^*; A_2 := A_1^{s_2}$ $B_2 := B_1^{s_2} A_1^{-r_2 s_2}$ if $B_2 \neq 1$, abort
Step 7.		$\xleftarrow{\mathsf{trancripts}(\mathcal{P}_1, \mathcal{P}_2)}$	if $\pi_2^{\ell_2}$ fails or $B_2 \neq 1$, abort		
Step 8.	Check the transcripts				

Figure 2: Two-server password-based authentication protocol

ably assumed that the adversary $\mathcal{A}$ learns nothing but that messages are sent and other indifferent information on the messages. $\mathcal{P}_i$ locally stores $(sid, \mathcal{P}_1, \mathcal{P}_2, p_i)$ where $i \in \{1, 2\}$, and outputs $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2)$ to $\mathcal{Z}$. The user $\mathcal{U}$ ends the initialization phase with removing all the internal data.

When a protocol instance begins, each party, firstly, visits $\mathcal{F}_{\text{CRS}}$ to obtain the public parameters $(\mathbb{G}, q, g, h', h)$. Then they encrypt their inputs under the ElGamal encryption scheme, and send them to the specified peers (from Step 1 to Step 3). To capture the asymmetric deployment, $\mathcal{P}_1$, as a front-end server, has to deliver messages between the user $\mathcal{U}$ and the back-end server $\mathcal{P}_2$. Without the PKI assumption, it would not expose the identity of $\mathcal{P}_2$. Specifically, after receiving $(\mathsf{NewSession}, sid, ssid, p)$ from $\mathcal{Z}$, where $p \in \mathbb{G}$ also, the user $\mathcal{U}$ under *username* included in *sid*, encrypts his password attempt p with a randomness r drawn from $\mathbb{Z}_q^*$, so that $(u, v) := (g^r, h'^r p)$ under the public key (g, h'), and sends it to initiate a session with $\mathcal{P}_1$ and an unknown back-end server. When $\mathcal{P}_1$ receives the message, he forwards it to $\mathcal{P}_2$ together with the ciphertext $(u_1, v_1) := (g^{r_1}, h^{r_1} p_1)$ of the password share p_1 with a randomness r_1 drawn from $\mathbb{Z}_q^*$ under (g, h). Similarly, $\mathcal{P}_2$ also computes $(u_2, v_2) := (g^{r_2}, h^{r_2} p_2)$ and replies it to through $\mathcal{P}_1$. These parties naturally store their transcripts and temporary internal states. As can be noticed, $\mathcal{U}$ uses a different public key (g, h') with the servers. It's in consideration for separated reductions of two ElGamal ciphertext in the security proof.

In the next steps, two servers jointly finish threshold decryption for the user's encryption of password. $\mathcal{U}$ selects a randomness s, and computes an abnormal ElGamal ciphertext $(A, B) := (h^s, (v_1 v_2 / p)^s)$ by help of its homomorphic property. Precisely, $(A, B) := (h^s, (h^{(r_1 + r_2)s} (p_1 p_2 / p)^s)$, where the cihpertext is essentially an encryption of the randomized quotient of passwords $(p_1 p_2 / p)^s$, and the secret key is $r_1 + r_2$. In addition, $\mathcal{U}$ is required to generate a labeled SS-NIZK proof π^ℓ that (A, B) are computed correctly, and it prevents from trivially using the identity element "1" of group $\mathbb{G}$ as the quotient of passwords. That is,

$$\pi^\ell := \mathsf{NIZK}\{(r, s) : (u, v) = (g^r, h'^r p) \wedge (A, B) = (h^s, (v_1 v_2 / p)^s)\},$$

where $\ell := (sid, ssid, A, B, u_1, v_1, u_2, v_2)$. Then he hands the message (A, B, π^ℓ) to the front-end servers $\mathcal{P}_1$. Then servers decrypt the ciphertext through cooperation, since each one only owns a part of the secret key. Once receiving the message from $\mathcal{U}$, the server $\mathcal{P}_1$ verifies the proof π^ℓ, and aborts the protocol if it is invalid. In order to prevent unintentional disclosure of the secret information and guarantee that the servers provide the correct decryption operations, $\mathcal{P}_1$ randomizes and partially decrypts the ciphertext from $\mathcal{U}$, and generates the SS-NIZK:

$$\pi_1^{\ell_1} := \mathsf{NIZK}\{(s_1, r_1) : A_1 = A^{s_1} \wedge B_1 = B^{s_1} A^{-r_1 s_1} \wedge u_1 = g^{r_1}\},$$

where $\ell_1 := (sid, ssid, A_1, B_1)$. Next he sends the ciphertext partially decrypted and the proof to $\mathcal{P}_2$, along with the message from $\mathcal{U}$. Upon getting them, the server $\mathcal{P}_2$ verifies the proofs π^ℓ, $\pi_1^{\ell_1}$, and aborts the protocol if either is invalid. $\mathcal{P}_2$ similarly runs and provides a proof:

$$\pi_2^{\ell_2} := \mathsf{NIZK}\{(s_2, r_2) : A_2 = A_1^{s_2} \wedge B_2 = B_1^{s_2} A_1^{-r_2 s_2} \wedge u_2 = g^{r_2}\},$$

where $\ell_2 := (sid, ssid, A_2, B_2)$. Then, he returns (A_2, B_2) and $\pi_2^{\ell_2}$ to $\mathcal{P}_1$. If their passwords do match, the final decryption result is $B_2 = 1$ and it's overwhelmingly a random group element otherwise. Each server outputs succ or fail according to $B_2 = 1$ or not. Finally, $\mathcal{P}_1$ passes the transcripts between two servers (denoted as $\mathsf{trancripts}(\mathcal{P}_1, \mathcal{P}_2)$) in Step 5 and Step 6 to $\mathcal{U}$, which allows him to inform of the result.

Note that we make $\mathcal{U}$ send the ciphertext of the password as an initiation of a new protocol instance. On the surface, this message is of no use in authentication. However, without the first round message, the simulation will get stuck in the security proof, i.e. that the protocol does not realize the $\widehat{\mathcal{F}}_{2\mathrm{PA}}$ in the UC framework. Specially, consider when the adversary impersonates a user. The simulator $\mathcal{S}$ (the adversary in the ideal-world) has to, on behalf of the servers, send (u_1, v_1), (u_2, v_2) before it receives the ciphertext (A, B) from the real-world $\mathcal{A}$ called by $\mathcal{S}$ in the simulation. However, $\mathcal{S}$ cannot extract $\mathcal{A}$'s password attempt and verify whether it provides a matching guess. Namely, the failed simulation could be easily detected, thwarted accordingly—the environment may realize the difference results from ones the functionality outputs. Therefore, the pre-flow message (u, v) encrypted "to the fly" is necessary to help $\mathcal{S}$ get the password attempt of $\mathcal{A}$ via the secret key corresponding to (g, h'). Moreover, a labeled simulation-sound NIZK guarantees that the passwords are consistent, and allows the simulator to give a convincing proof of the false statement with a dummy password.

3.3 Instantiation and Complexity

Here, we concisely provide the instantiations of the proofs in our scheme, and make them non-interactive and simulation-sound by the Fiat-Shamir transformation [18]. It requires the prover to add the open information into the *label*, and includes it as an argument to the random oracle. The following notation is used. $\mathsf{PK}\{(a, b, c) : y = g^a h^b \wedge \widetilde{y} = g^a h^c\}$ denotes a zero-knowledge proof of knowledge of integers a, b, c such that $y = g^a h^b$ and $\widetilde{y} = g^a h^c$ hold, where y, g, h and $\widetilde{y}$ are from $\mathbb{G}$, and known to the verifier. We refer to [7, 10, 8] for details.

The proof π^ℓ by which the user $\mathcal{U}$ proves that (u, v) and (A, B) are associated with the same p is generated in Step 4. Note that $(u, v){:=}(g^r, h'^r p)$ and $(A, B){:=}(h^s, (v_1 v_2/p)^s)$. The passwords encrypted thus are vh'^{-r} and $v_1 v_2 B^{-s^{-1}}$, respectively. Then rewrite the statement $vh'^{-r}{=}v_1 v_2 B^{-s^{-1}}$ into $B{=}h'^{rs}(v_1 v_2/v)^s$. By definition $\beta{:=}rs$, it is trivial to conclude that $1{=}u^s g^{-\beta}$. Hence, π^ℓ can be realized as:

$$\pi^\ell{:=}\mathsf{PK}\{(r, s, \beta){:}u{=}g^r \wedge A{=}h^s \wedge 1{=}u^s g^{-\beta} \wedge B{=}h'^{\beta}(v_1 v_2/v)^s\},$$

where $\beta := rs$ and $\ell := (sid, ssid, A, B, u_1, v_1, u_2, v_2)$. It takes both the prover and the verifier about 4 exponentiations in the group.

The proof $\pi_1^{\ell_1}$ generated by $\mathcal{P}_1$ ensures that the ciphertext (A, B) is honestly randomized with s_1, and partially decrypted with "secret key" r_1 that ever used as the randomness to generate (u_1, v_1). Indeed, the proof just shows that B_1 is the product of B and A_1^{-1} that are raised to s_1 and r_1, respectively. It is not hard to be implemented as:

$$\pi_1^{\ell_1} := \mathsf{PK}\{(s_1, r_1) : A_1{=}A^{s_1} \wedge B_1{=}B^{s_1} A_1^{-r_1} \wedge u_1{=}g^{r_1}\},$$

where $\ell_1 := (sid, hp, h_1, A_1, B_1)$. Observe that $\pi_1^{\ell_1}$ and $\pi_2^{\ell_2}$ are essentially the same except the indices, and then the latter can be made analogously. $\pi_1^{\ell_1}$ costs 3 exponentiations for the prover and the verifier, respectively, so does $\pi_2^{\ell_2}$.

In computation cost respect, the user $\mathcal{U}$ has to compute 14 exponentiations, while each server needs to do 15 exponentiations. The scheme requries 8-round messages altogether, and four of them are transmitted in the relatively safe and efficient internal network.

3.4 Security

THEOREM 1. *Under the DDH assumption and in the random oracle model, if the proofs involved in the schemes are labeled simulation-sound NIZK ones, then the two-server password-based authentication protocol described in Figure 2 securely realizes $\widehat{\mathcal{F}}_{2\mathrm{PA}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-$\mathcal{F}_{\mathrm{SMT}}$-hybrid model.*

We show a concrete analysis of security.

3.4.1 Description of Simulator

In order to make the $\widehat{\mathcal{F}}_{2\mathrm{PA}}$ output the indistinguishable value to $\mathcal{Z}$, $\mathcal{S}$ invokes a copy instance of $\mathcal{A}$ and provides it with a simulated world where $\mathcal{S}$ acts as $\mathcal{Z}$ and other honest parties. $\mathcal{S}$ honestly forwards all messages between $\mathcal{A}$ and $\mathcal{Z}$. Before providing the process of simulation, we put forward some notions for simplification purposes. Say that a message flow is *oracle-generated*, if it comes from an honest party and delivered to the target peer without malicious tempering. Say that it is *adversary-generated* otherwise, that is, either if this message results from a vicious party that could be corrupted or impersonated by $\mathcal{A}$, or if it is sent by an honest party and modified by $\mathcal{A}$. Except for the initialization phase, each party may sustain the active attacks from the adversary, so we take into account several cases. More simulation details are outlined below.

Initialization. Before the initialization phase, with security parameter κ, $\mathcal{S}$ generates the ElGamal public keys (g, h), (g, h') and the corresponding secret keys a and a', satisfying $h := g^a$, $h' := g^{a'}$. Then $\mathcal{S}$ gives $(\mathbb{G}, q, g, h', h)$ to $\mathcal{A}$ as the answers to its queries to $\mathcal{F}_{\mathrm{CRS}}$, and privately stores (a', a). Once receiving $(\mathsf{Initialization}, sid, \mathcal{P}_1, \mathcal{P}_2)$ from $\widehat{\mathcal{F}}_{2\mathrm{PA}}$, $\mathcal{S}$ chooses dummy shares $\overline{p}_1$ $\overline{p}_2$ randomly from $\mathbb{G}$, and then passes them to $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, through $\mathcal{F}_{\mathrm{SMT}}$. The initialization phase ends, after servers store the shares.

Simulating the user. Once the simulator $\mathcal{S}$ receives the query $(\mathsf{NewSession}, sid, ssid, \mathcal{U})$ from $\widehat{\mathcal{F}}_{2\mathrm{PA}}$, it begins to simulate the honest user $\mathcal{U}$. $\mathcal{S}$ computes the ciphertext (u, v) as an encryption of a dummy password attempt $\overline{p}$, and sends the message (u, v) on behalf of $\mathcal{U}$ to the adversary $\mathcal{A}$ who controls the network. Then $\mathcal{S}$ receives a reply (u_1, v_1, u_2, v_2) from $\mathcal{A}$, and one of the following events may happen:

- If neither (u_1, v_1) nor (u_2, v_2) is oracle-generated, then $\mathcal{S}$ decrypts the ciphertexts to obtain the adversary's password $\widehat{p}$ with secret key a (the product of two pieces of plaintexts $\widehat{p} := p_1 p_2$) and sends $(\mathsf{TestPwd}, sid, ssid, \mathcal{U}, \widehat{p})$ to $\widehat{\mathcal{F}}_{2\mathrm{PA}}$. The following steps have two cases:
  - If $\mathcal{S}$ gets "correct guess", it uses $\widehat{p}$ as the true password to simulate the subsequent execution. Later, $\mathcal{U}$ runs the protocol as specified except for generating the simulated proof that (u, v) and (A, B) use the same password. If, in Step 8, the transcripts $\mathcal{U}$ receives from $\mathcal{A}$ are successfully verified, $\mathcal{S}$ outputs $(\mathsf{Result}, sid, ssid, \mathcal{U})$ to $\widehat{\mathcal{F}}_{2\mathrm{PA}}$; otherwise, $\mathcal{S}$ aborts.

  - If the reply is "wrong guess", the simulator randomly chooses two group elements (A, B) as the ciphertext, and sends it with other messages. Upon receiving the final flow from $\mathcal{A}$, $\mathcal{S}$ directly aborts the protocol execution, and outputs $(\mathsf{Result}, sid, ssid, \mathcal{R})$ to $\widehat{\mathcal{F}}_{\mathrm{2PA}}$.

- If one message is oracle-generated, and the other is not, where assume that $\mathcal{P}_2$'s is adversary-generated, without loss of generality, $\mathcal{S}$ needs to handle the following cases:
  - If $\mathcal{A}$ just only impersonates $\mathcal{P}_2$ and sends faked messages, $\mathcal{S}$ aborts upon receiving $(A_2, B_2, \pi_2^{\ell_2})$, since $\mathcal{A}$ hardly generates an encryption of the correct share and provides a convincing proof for a false statement with a non-negligible probability.
  - If the message is produced by the corrupted server $\mathcal{P}_2$, and $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ outputs succ to $\mathcal{U}$ after $\mathcal{S}$ gives the query $(\mathsf{Result}, sid, ssid, \mathcal{U})$, the simulator randomly chooses $s^* \in \mathbb{Z}_q^*$, and sets the ciphertext $(A, B) := (g^{s^*}, (u_1u_2)^{s^*})$, where $u_1u_2 = g^{(r_1+r_2)}$. Then $\mathcal{S}$ sends out (A, B). Afterwards, $\mathcal{P}_1$ generates $(A_1, B_1) := (g^{s_1^*}, u_2^{s_1^*})$, where $u_2 = g^{r_2}$ and $s_1^* \in \mathbb{Z}_q^*$ also.

- If these two messages are oracle-generated from $\mathcal{A}$, then $\mathcal{S}$ continues to use $\overline{p}$ as the password attempt to compute the ciphertexts (A, B), and generate the simulated proof π^ℓ. After obtaining the reply from the adversary, $\mathcal{S}$ eventually outputs $(\mathsf{Result}, sid, ssid, \mathcal{R})$ to the functionality $\widehat{\mathcal{F}}_{\mathrm{2PA}}$.

Simulating both servers. Two honest servers simulated by $\mathcal{S}$ are activated with the incoming message (u, v) from $\mathcal{A}$. If the ciphertexts received by two servers are different, $\mathcal{S}$ aborts until two servers receive the response of $\mathcal{U}$. After ruling out the trivial case, we consider these:

- If (u, v) is adversary-generated, $\mathcal{S}$ extracts $\mathcal{A}$'s attempt $\widehat{p}$ from (u, v) and delivers $(\mathsf{TestPwd}, sid, ssid, \mathcal{P}_1, \mathcal{P}_2, \widehat{p})$ to $\widehat{\mathcal{F}}_{\mathrm{2PA}}$. Upon receiving the response from $\widehat{\mathcal{F}}_{\mathrm{2PA}}$, the simulator will face one of two scenarios:
  - If the reply is "correct guess", $\mathcal{S}$ randomly chooses p_1 from $\mathbb{G}$, and sets $p_2 := \widehat{p}/p_1$. Then, after selecting two random values r_1 and r_2 from $\mathbb{Z}_q^*$, the simulator honestly encrypts them to generate (u_1, v_1) and (u_2, v_2), and sends them to $\mathcal{A}$, respectively. In the subsequent steps, $\mathcal{S}$ simulates the servers according to the prescribed procedure. In the end, $\mathcal{S}$ sends $(\mathsf{Result}, sid, ssid, \mathcal{R})$ to $\widehat{\mathcal{F}}_{\mathrm{2PA}}$, where $\mathcal{R} \in \{\mathcal{P}_1, \mathcal{P}_2\}$.
  - If the reply is "wrong guess", $\mathcal{S}$ replaces the normal ciphertexts (u_1, v_1) and (u_2, v_2) with encryptions of dummy values randomly from $\mathbb{G}$. After receiving messages from $\mathcal{U}$, $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively selects random ciphertexts (A_1, B_1) and (A_2, B_2), makes proofs of the fake statements, and passes them to each other. With a significant probability, $B_2 \neq 1$. Then $\mathcal{S}$ sends $(\mathsf{Result}, sid, ssid, \mathcal{R})$ to $\widehat{\mathcal{F}}_{\mathrm{2PA}}$, where $\mathcal{R} \in \{\mathcal{P}_1, \mathcal{P}_2\}$.

- If this message is oracle-generated and the output from $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ is succ, $\mathcal{S}$ proceeds as if the servers' inputs were $\overline{p}_1$ and $\overline{p}_2$. Especially, $\mathcal{P}_2$ sets $B_2 := 1$, while other messages are randomly chosen, and the proofs are simulated. Then $\mathcal{S}$ sends them to $\mathcal{A}$. If it is fail, $\mathcal{S}$ chooses random elements as ciphertexts, and aborts until it checks that $B_2 \neq 1$.

Simulating one server. Without loss of generality, let $\mathcal{P}_2$ be honest, while the server $\mathcal{P}_1$ is corrupted or impersonated by $\mathcal{A}$. If the simulated message (u_1, v_1) from $\mathcal{P}_1$ is tampered by $\mathcal{A}$, $\mathcal{S}$ sends the random (A_2, B_2) and outputs fail, regardless of whether the password attempt is correct or not. This is because $\mathcal{A}$ correctly guesses a high-entropy share with a negligible probability. If the adversary has corrupted $\mathcal{P}_1$, $\mathcal{S}$ provides $\mathcal{A}$ with the shares $\overline{p}_1$, and just needs to simulate the $\mathcal{P}_2$'s performance with the dummy share $\overline{p}_2$. Thus, we consider the different cases where $\mathcal{P}_1$ has been corrupted:

- If (u, v) from the user is adversary-generated, the operation of $\mathcal{S}$ is similar to the case above, except that if $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ passes "correct guess" to $\mathcal{S}$, it sets $p_2 := \widehat{p}/\overline{p}_1$, and honestly executes the protocol following the instructions.

- If they are oracle-generated, as mentioned previously, $\mathcal{S}$ needs to set $B_2 := 1$ when the passwords match. Conversely, the simulator randomly chooses dummy values A_2 and B_2, if $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ returns "wrong guess".

If $\mathcal{P}_1$ is honest, while $\mathcal{P}_2$ is not, the performance of $\mathcal{S}$ is similar to except that, when $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ sends "correct guess", instead of regular computing (A_1, B_1), the simulator randomly selects s_1^* from $\mathbb{Z}_q^*$, and sets $(A_1, B_1) := (g^{s_1^*}, u_2^{s_1^*})$, where $u_2 = g^{r_2}$ from $\mathcal{P}_2$.

3.4.2 Sequence of Games

Here, via a sequence of games $\mathsf{G_i}$, we will prove that the real-world with the arbitrary $\mathcal{A}$ and the ideal-world with $\widehat{\mathcal{F}}_{\mathrm{2PA}}$ and $\mathcal{S}$ as defined above are indistinguishable in the view of environment $\mathcal{Z}$. This needs to be stressed that, the simulator $\mathcal{S}$ "magically" obtains the inputs of honest parties provided by $\mathcal{Z}$ in the intermediate games, but they are no longer needed at the end of simulation. Following is the sequence of concrete games:

Game $\mathsf{G_0}$: Let $\mathsf{G_0}$ be the real-world game. As we noted above, $\mathcal{S}$ "magically" receives the inputs, and honestly runs the protocol instance.

Game $\mathsf{G_1}$: It is identical to $\mathsf{G_0}$, except that we change the generation of public parameters in the protocol. More specifically, on one hand, the common random strings are replaced with the simulated ones, and $\mathcal{S}$ knows the secret keys. On the other hand, whenever the honest parties perform the zero-knowledge proofs, $\mathcal{S}$ provides the simulated ones instead. The indistinguishability between them follows from the zero-knowledge properties of the proof system.

Game $\mathsf{G_2}$: $\mathcal{S}$ replaces (u, v) from the honest $\mathcal{U}$ by an encryption of the dummy password attempt $\overline{p}$. Two games are indistinguishable due to the semantic security of IND-CPA encryption schemes.

Game $\mathsf{G_3}$: If (u, v) is not oracle-generated, $\mathcal{S}$ extracts the password attempt $\widehat{p}$ using the secret key. In particular, if it is a "wrong guess", $\mathcal{S}$ halts the protocol until $\mathcal{P}_2$ and $\mathcal{P}_1$ receives $(A, B, \pi^\ell, A_1, B_1, \pi_1^{\ell_1})$ and $(A_2, B_2, \pi_2^{\ell_2})$, respectively. The major difference between $\mathsf{G_2}$ and $\mathsf{G_3}$ is that the servers verify the password attempt depending on (u, v) instead of (A, B). This change is observable, if in $\mathsf{G_3}$, $\mathcal{S}$ aborts the protocol instance in virtue of "wrong guess", while in $\mathsf{G_2}$, $\mathcal{A}$ provides (A, B) containing the "correct guess". Namely, (u, v) and (A, B) are encryptions of different values. However, it happens with negligible probability. The simulation-soundness of NIZKs implies that $\mathcal{A}$ hardly gives a convincing proof π^ℓ, which ensures two games are indistinguishable.

Upon receiving $(\mathsf{NewKey}, sid, ssid, \mathcal{R}, sk^*)$ **from** $\mathcal{S}$**, where** $\mathcal{R} \in \{\mathcal{U}, \mathcal{P}_1, \mathcal{P}_2\}$**:**
If there is a stored NewSession record for $\mathcal{R} \in \{\mathcal{U}, \mathcal{P}_1\}$, then:
• If it is compromised, or either of two records are corrupted, then $\mathcal{F}$ outputs $(\mathsf{NewKey}, sid, ssid, sk^*)$ to $\mathcal{R}$.
• When it is labeled as interrupted, or the passwords are not matching with the other record, then $\mathcal{F}$ sends a public delayed output $(\mathsf{NewKey}, sid, ssid, \mathsf{fail})$ to $\mathcal{R}$, if $\mathcal{R} = \mathcal{P}_1$.
• If this record is fresh, the other record with $p' = p$ is also fresh, and a key sk has been sent, then $\mathcal{F}$ outputs $(\mathsf{NewKey}, sid, ssid, sk)$ to $\mathcal{R}$.
• In any other case, $\mathcal{F}$ randomly chooses a new key sk, and outputs $(\mathsf{NewKey}, sid, ssid, sk)$ to $\mathcal{R}$.
If there is a stored record for $\mathcal{R} = \mathcal{P}_2$, then:
• If his record is interrupted, or the other one (if it exists) is labeled as interrupted or compromised, or $p' \neq p$, then $\mathcal{F}$ gives a public delayed output $(\mathsf{NewKey}, sid, ssid, \mathsf{fail})$ to $\mathcal{R}$.
• In other cases, $\mathcal{F}$ gives a public delayed output $(\mathsf{NewKey}, sid, ssid, \mathsf{succ})$ to $\mathcal{R}$.
Either way, $\mathcal{F}$ marks this record completed.

Figure 3: The Newkey query of $\mathcal{F}_{\text{2PAKE}}$

Game G_4: It follows G_3 above. If (u, v) is adversary-generated and an encryption of a non-matching password, $\mathcal{S}$ computes (u_1, v_1) and (u_2, v_2) as encryptions of dummy shares, rather than the correct ones. As between G_1 and G_2, G_4 is indistinguishable from G_3 by the semantic security of the ElGamal encryption scheme.

Game G_5: Compared with G_4, $\mathcal{S}$ goes further. When the ciphertext (u, v) is not oracle-generated and $\mathcal{F}$'s response is "wrong guess", $\mathcal{S}$ chooses (A_1, B_1) and (A_2, B_2) at random from $\mathbb{G}^2$. Here, we only prove that the gap brought by change of the former is indistinguishable under DDH assumption, and the latter is nearly similar.

First, define a hybrid Mix just like G_4 except when (u, v) is adversary-generated and an encryption of a non-matching password, $\mathcal{S}$ just randomly chooses the ciphertext (A_1, B_1). Between G_4 and Mix, we run a series of hybrid games, and $\mathsf{G}_{4,i}$ are defined as follows. In the first i queries, the ciphertexts are random as well as in the Mix, and the values are real for the rest of queries. $\mathcal{S}$ can construct such a distinguisher $\mathcal{D}$ that it can solve the DDH problem by the help of $\mathcal{Z}$ who manages to distinguish between $\mathsf{G}_{4,i-1}$ and $\mathsf{G}_{4,i}$.

Concretely, $\mathcal{D}$ obtains a tuple (A, B, X, Y) that is either randomly sampled or a DDH tuple, i.e. $X := A^{s_1}$ and $Y := B^{s_1}$. In $\mathsf{G}_{4,j}$, where $j = i - 1$ or $j = i$, $\mathcal{D}$ sets $(A_1, B_1) := (X, YX^{-r_1})$ and makes a simulated proof. If this tuple is random, it is the hybrid $\mathsf{G}_{4,i-1}$, and it is the hybrid $\mathsf{G}_{4,i}$, otherwise. Finally, $\mathcal{D}$ outputs whatever $\mathcal{Z}$ outputs.

Thus, we believe that the advantage of $\mathcal{D}$ in the DDH problem is close to the advantage of $\mathcal{Z}$'s distinguishing these two hybrids. Further, G_4 and G_5 are indistinguishable.

Game G_6: Here, we introduce the modifications to the previous game. When (u, v) is oracle-generated and it corresponds a fail result, $\mathcal{S}$ randomly chooses (A_1, B_1) and (A_2, B_2), and two servers abort the protocol after checking $B_2 \neq 1$. As G_5, the DDH assumption ensures the changes are not observable with overwhelming probability. If the passwords match, $\mathcal{P}_1$ randomly selects s_1^* from $\mathbb{Z}_q^*$, and sets the ciphertext $(A_1, B_1) := (g^{s_1^*}, u_2^{s_1^*})$, where $u_2 = g^{r_2}$, while $\mathcal{P}_2$ chooses A_2 at random and sets $B_2 := 1$. This syntactic change has no effect on the distributions of outputs.

Game G_7: The modification is as follows. Once the uncorrupted servers receive the oracle-generated message (u, v), The correct pieces of password are no longer of use, $\mathcal{S}$ computes (u_1, v_1), (u_2, v_2) as encryptions of dummy shares $\overline{p}_1$, $\overline{p}_2$. The indistinguishability follows from the IND-CPA security of the ElGamal encryption scheme.

Game G_8: From this game on, if the ciphertexts (u_1, v_1), (u_2, v_2) and (u, v) are all oracle-generated, $\mathcal{S}$ randomly chooses the dummy ciphertext (A, B). Note the fact that the honestly forwarded ciphertexts (u_1, v_1), (u_2, v_2) are not encryptions of correct shares. Hence, the DDH assumption guarantees the indistinguishability between G_7 and G_8.

Game G_9: From this game on, if both (u_1, v_1) and (u_2, v_2) are not adversary-generated, and the passwords do not match, $\mathcal{S}$ randomly chooses two group elements from $\mathbb{G}$ as (A, B). In contrast, if the passwords are equal, the simulator randomly chooses s^*, and sets the ciphertext $(A, B) := (g^{s^*}, (u_1 u_2)^{s^*})$, where $u_1 u_2 = g^{(r_1 + r_2)}$. This change is nearly undetected, following the DDH assumption also.

Game G_{10}: When neither (u_1, v_1) nor (u_2, v_2) sent to the honest user is oracle-generated, $\mathcal{S}$ extracts $\widehat{p} := p_1 p_2$. If it is a matching one, $\mathcal{S}$ honestly acts as $\mathcal{U}$ with $\widehat{p}$. Otherwise, (A, B) is set as random values. Similarly, the DDH assumption ensures the indistinguishability between G_9 and G_{10}.

Now, we observe that the ideal-world is identical to G_{10}, except that $\mathcal{S}$ no longer owns the specified inputs from $\mathcal{Z}$, and have capabilities to query $\widehat{\mathcal{F}}_{\text{2PA}}$ to model the active attacks of $\mathcal{A}$ instead. Apparently, the gap is locally syntactic as well. Thus, the proof of theorem is completed.

4. 2PAKE

We suggest the first formal notions of the ideal functionality of two-server password-authenticated key exchange. It follows the definition of 2PA except that the Result query is placed with the Newkey query depicted in Figure 3. Naturally, $\mathcal{F}_{\text{2PAKE}}$ inherits the entity authentication of the user from $\mathcal{F}_{\text{2PA}}$. In our model, the front-end server $\mathcal{P}_1$ can share a session key with the user only if the latter owns an eligible password attempt, while $\mathcal{P}_2$, as the back-end server, just plays a role in assisting $\mathcal{P}_1$ in authenticating $\mathcal{U}$ rather establishing another with him. Hence, he only obtains the result of authentication (succ or fail). It is similar to the $\mathcal{F}_{\mathsf{pwKE}}$ of [12], in that if one of NewSession records is corrupted the adversary is given the ability to fully determine the resulting session key, after passing the query $(\mathsf{NewKey}, sid, ssid, \mathcal{R}, sk^*)$ to $\mathcal{F}_{\text{2PAKE}}$. It's more of a con-

	$\mathcal{U}(p)$	$\mathcal{P}_1(p_1)$	$\mathcal{P}_2(p_2)$
Step 4.	$s \leftarrow \mathbb{Z}_q^*; A := h^s; B := (v_1v_2/p)^s$ $hk := (\eta, \theta) \leftarrow \mathbb{Z}_q^{*2}; hp := g^\eta h^\theta$ $H := (u_1u_2)^\eta (v_1v_2/p)^\theta; sk := H$		
	$\xrightarrow{hp, A, B, \pi^\ell}$		
Step 5.		if π^ℓ fails, abort; $s_1 \leftarrow \mathbb{Z}_q^*$ $A_1 := A^{s_1}; B_1 := B^{s_1} A^{-r_1 s_1}$ $w_1 \leftarrow \mathbb{Z}_q^*; h_1 := g^{w_1}; H_1 := hp^{r_1}$	
		$\xrightarrow{hp, A, B, \pi^\ell, h_1, A_1, B_1, \pi_1^{\ell_1}}$	
Step 6.			if π^ℓ or $\pi_1^{\ell_1}$ fails, abort $s_2 \leftarrow \mathbb{Z}_q^*; A_2 := A_1^{s_2}$ $B_2 := B_1^{s_2} A_1^{-r_2 s_2}$ if $B_2 \neq 1$, abort; $H_2 := hp^{r_2}$ $t_2 \leftarrow \mathbb{Z}_q^*; U_2 := g^{t_2}; V_2 := h_1^{t_2} H_2$
		$\xleftarrow{A_2, B_2, U_2, V_2, \pi_2^{\ell_2}}$	
Step 7.		if $\pi_2^{\ell_2}$ fails or $B_2 \neq 1$, abort $H_2 := V_2/U_2^{w_1}; sk := H_1H_2$	

Figure 4: Step 4 to Step 7 of 2PAKE

venience for the simulation in the security proof than an important improvement, compared with $\mathcal{F}_{\text{2PA}}$.

Next, we extend the 2PA protocol to a 2PAKE scheme to UC-realize $\mathcal{F}_{\text{2PAKE}}$. It follows the same initialization phase and Step 1 to Step 3 (cf. Figure 2) of our 2PA protocol scheme in Section 3, and we propose the detailed description of the other steps in Figure 4. Here, we use the smooth projective hash function as a building block. Once $\mathcal{U}$ receives the ElGamal ciphertexts of password shares from $\mathcal{P}_1$ and $\mathcal{P}_2$, he produces the abnormal ciphertext (A, B) as before. Moreover, he uniformly samples $hk := (\eta, \theta)$ from $\mathbb{Z}_q^{*2}$ as the hash key and generates the projection key $hp := g^\eta h^\theta$. Then $\mathcal{U}$ computes the hash value on the multiplicity of the received ciphertexts as the final session key $sk := \mathsf{Hash}(hk, u_1u_2, v_1v_2, p) = (u_1u_2)^\eta(v_1v_2/p)^\theta$. He passes the projected key hp to $\mathcal{P}_1$ together with the ciphertext (A, B) and the labeled SS-NIZK π^ℓ. Its label $\ell := (sid, ssid, hp, A, B)$ contains hp so that the adversary cannot replace it with a malicious value to predict the session key from the simulation-soundness property. $\mathcal{P}_1$ not only finishes partial decryption and provides $\pi_1^{\ell_1}$, but also computes $H_1 := \mathsf{ProjHash}(hp, u_1u_2, v_1v_2, p_1, r_1) = hp^{r_1}$ as a session key material and temporary public key (g, h_1) that is bound with his password share by the label, i.e. $\ell_1 := (sid, ssid, hp, h_1, A_1, B_2)$. Similarly, $\mathcal{P}_2$ runs as in in the 2PA protocol but computes the session key material $H_2 := \mathsf{ProjHash}(hp, u_1u_2, v_1v_2, p_2, r_2) = hp^{r_2}$ and transmits it encrypted under (g, h_1) to $\mathcal{P}_1$, if $B_2 = 1$. The proof $\pi_2^{\ell_2}$ needs an additional guarantee that $\mathcal{P}_2$'s valid computation on ciphertexts of H_2. It can be expressed as followed:

$$\pi_2^{\ell_2} := \mathsf{NIZK}\{(s_2, r_2, t_2) : A_2 = A_1^{s_2} \wedge B_2 = B_1^{s_2} A_1^{-r_2 s_2} \wedge u_2 = g^{r_2} \wedge U_2 = g^{t_2} \wedge V_2 = h_1^{t_2} hp^{r_2}\},$$

where $\ell_2 := (sid, ssid, hp, h_1, A_2, B_2, U_2, V_2)$. From the correctness property of SPHF, when $\mathcal{U}$ possesses the correct password, he will share the same sk with $\mathcal{P}_1$. Conversely, if the passwords are not matching, the smoothness property of SPHF makes the values calculated by the user independent from the product of ones the servers obtain statistically.

When our protocol is instantiated using the Fiat-Shamir transformation [18] to obtain SS-NIZK proofs in the random oracle model, $\mathcal{U}$ has to compute 12 exponentiations, while each server needs to do 20 exponentiations. Besides that, we can draw a conclusion below, and have to provide the proof in the full paper for the limitation of space:

THEOREM 2. *Under the DDH assumption and in the random oracle model, if the proofs involved are labeled SS-NIZK ones, then the two-server password-authenticated key exchange securely realizes $\widehat{\mathcal{F}}_{\text{2PAKE}}$ in the $\mathcal{F}_{\text{CRS}}$-$\mathcal{F}_{\text{SMT}}$-hybrid model.*

5. 2PASS

The functionality of two-server password-authenticated secret sharing looks like the two-server version of Camenisch et al.'s in [8], and more concrete descriptions are omitted. For the sake of comparisons with previous schemes [10, 8], we draw on the definitions of TPASS in the [8] (for $t = 1$, and $n = 2$) and related terminology with some slight changes.

Here, we apply our 2PA scheme to 2PASS. However, for compatibility with [8]'s ideal functionality, we abandon the asymmetric architecture and the *sid* consists of two servers' identities. During the initialization phase (corresponding to [8]'s Setup protocol), $\mathcal{U}$ computes shares of password p' and secret k' from $\mathbb{G}$ so that $p' := p_1p_2$ and $k' := k_1k_2$, where p_1, k_1 are uniformly sampled from $\mathbb{G}$. Then he stores (p_1, k_1) and (p_2, k_2) on $\mathcal{P}_1$, $\mathcal{P}_2$, respectively, via $\mathcal{F}_{\text{SMT}}$. Step 1 to Step 3 of 2PASS are also inherited from our 2PA scheme, so we skip them and directly describe Step 4 to Step 8 in Figure 5. The hash key hk, the projective key hp and the hash value H generated by $\mathcal{U}$ are identical to ones in 2PAKE above. Additionally, he runs the key generation algorithm $\mathsf{KG}(\cdot)$ of the CCA2 public encryption scheme to get the key pair $(PK, SK) \leftarrow \mathsf{KG}(1^\kappa)$. Then hp and pk are bound with the encryption of p via the label of π^ℓ. $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively decrypt the ciphertext (A, B). If the passwords do not

	$\mathcal{U}(p)$		$\mathcal{P}_1(p_1, k_1)$		$\mathcal{P}_2(p_2, k_2)$
Step 4.	$s \leftarrow \mathbb{Z}_q^*; A := h^s; B := (v_1 v_2/p)^s$ $hk := (\eta, \theta) \leftarrow \mathbb{Z}_q^{*2}; hp := g^\eta h^\theta$ $H := (u_1 u_2)^\eta (v_1 v_2/p)^\theta$ $(PK, SK) \leftarrow \mathsf{KG}(1^\kappa)$				
		$\xrightarrow{PK, hp, A, B, \pi^\ell}$			
Step 5.			if π^ℓ fails, abort; $s_1 \leftarrow \mathbb{Z}_q^*$ $A_1 := A^{s_1}; B_1 := B^{s_1} A^{-r_1 s_1}$ $H_1 := hp^{r_1}$		
				$\xrightarrow{PK, hp, A, B, \pi^\ell, , h_1, A_1, B_1, \pi_1^{\ell_1}}$	
Step 6.					if π^ℓ or $\pi_1^{\ell_1}$ fails, abort $s_2 \leftarrow \mathbb{Z}_q^*; A_2 := A_1^{s_2}$ $B_2 := B_1^{s_2} A_1^{-r_2 s_2}$ if $B_2 \neq 1$, abort; $H_2 := hp^{r_2}$ $E_2 := \mathsf{Enc}_{PK}^{L_2}(k_2, H_2)$
				$\xleftarrow{A_2, B_2, E_2, \pi_2^{\ell_2}}$	
Step 7.			if $\pi_2^{\ell_2}$ fails or $B_2 \neq 1$, abort $E_1 := \mathsf{Enc}_{PK}^{L_1}(k_1, H_1)$		
		$\xleftarrow{E_1, E_2}$			

Step 8.	$(k_i, H_i) := \mathsf{Dec}_{SK}^{L_i}(E_i), i = \{1, 2\}$
	if $H \neq H_1 H_2$, abort
	else, output $k := k_1 k_2$				

Figure 5: Step 4 **to** Step 8 **of 2PASS**

match, servers will abort subsequent performances. Otherwise, they encrypt secret shares and hash values under the CCA2-secure public-key encryption scheme using the user's *PK* and the *sid* and *ssid* as labels, so that the recipient will reconstruct the secret. Thanks to the homomorphism of the ElGamal encryption and the SPHF on the ciphertexts, $\mathcal{U}$ can always check whether servers possess correct password shares after decrypting the ciphertexts in Step 8. The proof of the following theorem, is given in the full paper.

THEOREM 3. *Under the DDH assumption and in the random oracle model, if* (KG, Enc, Dec) *is a labeled CCA2-secure public-key encryption scheme, and the proofs involved are labeled SS-NIZK ones, then the protocol above securely realizes* $\widehat{\mathcal{F}}_{\text{2PASS}}$ *in the* $\mathcal{F}_{\text{CRS}}$*-*$\mathcal{F}_{\text{SMT}}$*-hybrid model.*

The labeled CCA2-secure encryption scheme is instantiated using the ElGamal encryption scheme with the Fujisaki-Okamoto transformation [20], while the labeled SS-NIZK is as above. Compared with previous UC-secure PASS schemes, ours relieves the cost of computation in the server side, each participant needs to do 18 exponentiations.

6. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their beneficial comments. This work is supported by the National Natural Science Foundation of China (No.U1536205, 61502527, 61170278) and the National Basic Research Program of China (No.2013CB338003).

7. REFERENCES

[1] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 433–444. ACM, 2011.

[2] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology—Eurocrypt 2000*, pages 139–155. Springer, 2000.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[4] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology—Eurocrypt 2000*, pages 156–171. Springer, 2000.

[5] J. G. Brainard, A. Juels, B. Kaliski, and M. Szydlo. A new two-server approach for authentication with short secrets. In *USENIX Security*, volume 3, pages 201–214, 2003.

[6] J. Camenisch, R. R. Enderlein, and G. Neven. Two-server password-authenticated secret sharing uc-secure against transient corruptions. In *Public-Key Cryptography—PKC 2015*, pages 283–307. Springer, 2015.

[7] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In *Advances in Cryptology—EUROCRYPT 2009*, pages 425–442. Springer, 2009.

[8] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In *Advances in Cryptology—CRYPTO 2014*, pages

256–275. Springer, 2014.

[9] J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 182–194. ACM, 2015.

[10] J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server password-authenticated secret sharing. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 525–536. ACM, 2012.

[11] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[12] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology—Eurocrypt 2005*, pages 404–421. Springer, 2005.

[13] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 494–503. ACM, 2002.

[14] R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology—Crypto 2003*, pages 265–281. Springer, 2003.

[15] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology—Eurocrypt 2002*, pages 45–64. Springer, 2002.

[16] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology—Crypto 2001*, pages 566–598. Springer, 2001.

[17] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1985.

[18] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1987.

[19] W. Ford and B. S. Kaliski Jr. Server-assisted generation of a strong secret from a password. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000.(WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, pages 176–180. IEEE, 2000.

[20] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology—CRYPTO'99*, volume 1666, pages 537–554. Springer, 1999.

[21] Gemalto. 2014 year of mega breaches & identity theft: Findings from the 2014 breach level index, 2015.

[22] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology—CRYPTO 2006*, pages 142–159. Springer, 2006.

[23] A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 516–525. ACM, 2010.

[24] D. P. Jablon. Password authentication using multiple servers. In *Topics in Cryptology—CT-RSA 2001*, pages 344–360. Springer, 2001.

[25] S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *Advances in Cryptology—ASIACRYPT 2014*, pages 233–253. Springer, 2014.

[26] J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. In *Applied Cryptography and Network Security*, pages 1–16. Springer, 2005.

[27] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on rsa. In *Advances in Cryptology—Asiacrypt 2000*, pages 599–613. Springer, 2000.

[28] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *Advances in Cryptology—CRYPTO 2002*, pages 385–400. Springer, 2002.

[29] D. Reisinger. No password is safe from this new 25-gpu computer cluster. http://www.cnet.com/news/no-password-is-safe-from-this-new-25-gpu-computer-cluster/, 2012.

[30] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 543–553. IEEE, 1999.

[31] D. Wang, G. Jian, X. Huang, and P. Wang. Zipf's law in passwords. Cryptology ePrint Archive, Report 2014/631, 2014. http://eprint.iacr.org/2014/631.pdf.

[32] D. Wang and P. Wang. Two birds with one stone: Two-factor authentication with security beyond conventional bound. IEEE Trans. Depend. Secur. Comput, 2016. In press, available at http://t.cn/RGCaI8f.

[33] Z. Whittaker. These companies lost your data in 2015's biggest hacks, breaches. http://www.zdnet.com/pictures/worst-largest-security-data-breaches-2015/, 2015.

[34] Y. Yang, R. H. Deng, and F. Bao. A practical password-based two-server authentication and key exchange system. *Dependable and Secure Computing, IEEE Transactions on*, 3(2):105–114, 2006.

[35] X. Yi, F. Hao, and E. Bertino. Id-based two-server password-authenticated key exchange. In *Computer Security—ESORICS 2014*, pages 257–276. Springer, 2014.

[36] X. Yi, S. Ling, and H. Wang. Efficient two-server password-only authenticated key exchange. *Parallel and Distributed Systems, IEEE Transactions on*, 24(9):1773–1782, 2013.