Introducing Fault Tolerance into Threshold Password-Authenticated Key Exchange

Ivan Pryvalov
Saarland University
pryvalov@cs.uni-saarland.de

Aniket Kate
MMCI, Saarland University
aniket@mmci.uni-saarland.de

July 30, 2014

Abstract

A threshold password-authenticated key exchange (T-PAKE) protocol allows a set of n servers to collectively authenticate a client with a human-memorizable password such that any subset of size greater than a threshold t can authenticate the client, while smaller subsets of servers learn no information about the password. With its protection against offline dictionary attacks, T-PAKE provides a practical solution for an important real-life problem with password authentication. However, the proposed T-PAKE constructions cannot tolerate any misbehavior—not even a crash—by a participating server during a protocol execution; the protocol has to be re-executed until all participating servers behave correctly. This not only presents a fault management challenge for the servers, but more importantly also leaves the clients frustrated for being denied access even after entering a correct password.

In this work, we present a novel T-PAKE protocol (T-PAKE$_{\mathsf{DKG}}$) which solves the above fault management problem by employing a batched and offline phase of distributed key generation (DKG). T-PAKE$_{\mathsf{DKG}}$ is secure against any malicious behavior from up to any $t < n$ servers under the decisional Diffie–Hellman assumption in the random oracle model, and it ensures protocol completion for $t < n/2$. Moreover, it is efficient ($16n + 7$ exponentiations per client, $20n + 14$ per server), performs explicit authentication in three communication rounds, and requires a significantly lesser number of broadcast rounds compared to previous secure T-PAKE constructions. We have implemented T-PAKE$_{\mathsf{DKG}}$, and have verified its efficiency using micro-benchmark experiments. Our experimental results show that T-PAKE$_{\mathsf{DKG}}$ only introduces a computation overhead of few milliseconds at both the client and the server ends, and it is practical for use in real-life authentication scenarios.

1 Introduction

Password-based authentication is the most widely used authentication mechanism in practice. Therefore, password-authenticated key exchange (PAKE) has been extensively studied in the literature [11, 25–27, 30]. In an untrusted environment, a PAKE protocol establishes a secure and authenticated channel between two participants, typically a client and a server. Here, the client is required to know a human-memorizable keyword, or *password*. In order to verify the client's identity and his knowledge of the password, the server maintains the correct password in a derived form (typically by employing a one-way function) in a so called "password file". However, as passwords are drawn from a relatively small dictionary, when the server gets compromised in a

	MSG06 [32]	BJS+11 [8]	RG06 [21]	**T-PAKE**$_{\mathsf{DKG}}$	**T-PAKE**$^{\mathsf{I}}_{\mathsf{DKG}}$
Server fault tolerance	×	×	×	√	√
# servers for protocol run (out of n)	$t+1$	$t+1$	n	$[t+1, n]$	$[t+1, n]$
Forward secrecy	×	×	√	√	√
# messages sent by *Client*	1(T)[1](B)^2 + 1(B)	1 (T)+ 1	2n	1 (T)+ 1	1
# messages sent by $Server_j$	4 (B)	2	n + 4n (B)	1 + 2 (B)	1 + 2 (B)
Server computation complexity	$O(n)$	$O(1)$	$O(n^2)$	$O(n)$	$O(n)$
Server public keys	√	√	×	√	√
Authentication type	Explicit	Implicit [3]	Implicit	Explicit	Implicit

[1] (T) denotes a trigger (or client-hello) message. [2] (B) denotes a broadcast message. [3] It is vulnerable to an online attack described in Section 1.2.

Table 1: Comparison of T-PAKE protocols

PAKE protocol, the attacker can launch an (offline) *dictionary attack* on the password file to recover the correct passwords eventually. Given the scale and the frequency of the server compromises we are observing on the Internet (e.g., [31, 38]), such attacks present a critical security challenge for online authentication systems.

In order to mitigate the offline dictionary attack, Ford and Kaliski [23] proposed to distribute password verification data among n servers, such that all n servers participate in the protocol and the adversary will have to break into all of them to launch a dictionary attack. MacKenzie *et al.* [32] generalized and formalized this concept to define threshold password-authentication key exchange (T-PAKE), where the password verification data (or the password file) is distributed among n servers such that the client has to run the authentication protocol with any $t+1 \leq n$ of those and an attacker who can compromise up to t servers is not able to launch an offline dictionary attack. Di Raimondo and Gennaro [21] presented a black-box T-PAKE construction for $3t < n$ using the KOY protocol [30], verifiable secret sharing (VSS) [22, 34], and a distributed multiplication protocol [5]. Recently, Baghezandi *et al.* [8] introduced a related concept of password-protected secret sharing (PPSS), and proposed an efficient T-PAKE construction using a PPSS as a black-box. A PPSS scheme allows a client to share a secret among n servers such that it can be reconstructed using a correct password. Although they are interesting theoretically, all these T-PAKE protocols do not satisfy the basic *liveness* property of *fault tolerance*: in all these protocols, if one or more participating servers misbehave, crash or even if their links to the client get broken due to some network failure, the protocol instance fails. The client has to *restart* with a new protocol instance, possibly with a different subset of the servers. Given the frequent server and link failures on the Internet, such an experience can be frustrating for the client; his authentication does not succeed even when he enters a correct password. Moreover, the existing secure T-PAKE constructions [21, 32] are significantly inefficient in terms of communication, which may further aggravate the clients' frustration.

1.1 Our Contributions

In this work, we address the above discussed fault-tolerance and inefficiency problems with T-PAKE. We observed that the root cause of the inability of these protocols to ensure protocol completion in the presence of a fault is the *uncorrelated* randomness employed by the participating servers during a T-PAKE instance. It is possible to solve this problem by correlating their randomness (in a threshold manner) using a distributed key generation (DKG) protocol [24, 34] such that an

adversary compromising up to t servers has no information about it but the randomness from any honest $t+1$ servers can be used to reconstruct the randomness of the rest of the servers.

Based on this observation, we propose an efficient, fault-tolerant T-PAKE protocol: T-PAKE$_{\mathsf{DKG}}$. Our protocol ensures that every T-PAKE execution succeeds without restarting as long as $t+1$ of the participating servers are honest, and thus it guarantees the protocol completion for $n \geq 2t+1$. We achieve this by a secure incorporation of a distributed key generation (DKG) protocol [24] as a black-box, offline mechanism in our construction, and by expecting the servers to execute several DKG instances in a batched fashion in the offline phase. We prove the security of our protocol under the decisional Diffie–Hellman (DDH) assumption in the random oracle model. The protocol is also efficient in terms of communication and requires less communication rounds [1] and broadcasts than in the previous *secure* T-PAKE protocols [21, 32]. Similar to [32], it also maintains security for any pre-defined $t < n$ subset of compromised servers. Moreover, our protocols can be employed to obtain the first fault tolerant constructions for the related PPSS [8] and PASS [17] primitives using a simple transformation suggested in [8, 17].

We also implement our T-PAKE$_{\mathsf{DKG}}$ protocol as well as the employed DKG protocol [24] and test their performance in the LAN environment for the 3-server (i.e., $n = 3$ and $t = 1$), and the 5-server (i.e., $n = 5$ and $t = 2$) settings. Our performance analysis demonstrates the practicality of the T-PAKE$_{\mathsf{DKG}}$ protocol in the real-life password authentication scenarios.

Our T-PAKE$_{\mathsf{DKG}}$ protocol performs two-way explicit authentication in three rounds. We also present another fault-tolerant T-PAKE protocol T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$, which performs implicit authentication in two rounds. Further, we extend our protocols to work in the asynchronous communication setting (no bounds on message transfer delays) for $n \geq 3t+1$ by employing asynchronous reliable broadcast and asynchronous DKG protocols.

1.2 Comparison with the Existing Constructions

In Table 1, we compare provably secure generic (involving $n \geq 2$ servers) T-PAKE constructions by MacKenzie *et al.* [32], Raimondo and Gennaro [21] and Bagherzandi *et al.* [8] with our T-PAKE$_{\mathsf{DKG}}$ and T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ protocols. All previous protocols cannot tolerate any fault, and abort in the presence of one. In contrast to those, our protocols can be instantiated with a subset of the servers of a variable size between $t+1$ and n (i.e., $[t+1, n]$), and they abort only when the number of the servers that follow the protocol drops below $t+1$. Our protocols as well as the protocol [21] provide forward secrecy, while the rest of the protocols do not. We distinguish a *trigger* message (sometimes called a client-hello message), which simply initiates a protocol instance between a client and servers. Our T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ protocol requires only one message directed from a client and provides an implicit authentication, which is the best among the compared protocols. A broadcast channel is used for exchanging messages between servers in a reliable and authenticated manner. In prior constructions, a broadcast channel was assumed to include the client along with all parties. We do *not* find that to be a practical assumption, and restrict a broadcast channel to the servers only. Our protocols require only two broadcasts per instance, which improves on all previous results. We do not include the broadcast instances required for DKG in our calculation as they are performed in an offline and batched manner (with an insignificant amortized overhead). The construction in [21] requires the

[1] We follow a convention about rounds in [32], where a *round* indicates a transfer of a message from a sender to a recipient.

clients to know *only* the password, while the rest of the protocols (including our protocols) require the clients to know the servers public keys.

The PPSS-based T-PAKE construction [8] also has an important problem. As individuals tend to choose simple passwords, it becomes crucial to restrict the adversary's capabilities to online guessing attacks. The resistance against online attacks was discussed in many works, e.g. Boyko *et al.* [14], by placing a limit on a number of unsuccessful authentication attempts. As observed by Camenisch el at. [17] while introducing password-authenticated secret sharing (PASS) in the PPSS protocol, the servers do *not* learn whether the client's password is correct or not; thus, it is impossible for the PPSS servers to block client accounts after a number of failed attempts to authenticate. We find the situation to be even worse: an attacker can efficiently *learn* whether a particular password is correct or not by launching a simple attack assuming that the attacker is able to trigger a PPSS protocol instance multiple times as if she were an honest user. Here, the attacker can simply trigger two instances of PPSS with the same password. If she gets two different reconstructed secrets, then the password was wrong, otherwise with an overwhelming probability the password was correct. We observe that this problem is inherent to the PPSS-based T-PAKE protocol, as the participating servers do not interact with each other in these protocols. We refer to Appendix A for a detailed description of the attack. Finally, we note that the servers can slow down such an attack to a certain extent only by limiting the frequency of *all* login attempts for a given user.

1.3 Application Scenarios

Password authentication protocols are typically used as an entry gate to some functionality that is offered by a server. In a threshold setting, several servers take part in authenticating a client and provide a desired service (such as downloading some protected information or synchronizing over the cloud) there up on. If an attacker compromises less than some predefined number of servers, she has no information about the password and cannot trigger the functionality that is supposed to come after the authentication succeeded.

We consider the following two application scenarios, which are particularly useful for the cloud sourcing environment.

Password-Authenticated Secret Sharing—PASS. As discussed earlier, T-PAKE scheme can be easily turned into PASS [8,17]: here, a client and servers establish secure channels using T-PAKE, and then the client retrieves secret shares over those channels.

Private Authentication. We say that a T-PAKE has a (t, n)-sharing scheme if there are n shares, an adversary is allowed to compromise up to t shares, and the protocol completes if at least $t + 1$ of them follow the protocol honestly. Typically, there are n servers, each server keeps one share and a user has no shares. If, instead of (t, n)-sharing scheme, the user implements a $(n, 2n - t)$ scheme, one can suppress online attacks [8]. In this setting, n servers still keep 1 share each, while the user keeps $n - t$ shares locally, e.g. using a private device. In order to authenticate, at least $t + 1$ servers are required to contribute jointly with $n - t$ users' shares. In particular, it means that the user is not able to authenticate if his device is lost.

1.4 Outline

The rest of the paper is organized as follows. Section 2 describes the system model, the cryptographic assumption, and the cryptographic tools used in our protocol. In Section 3, we present the detailed description of our T-PAKE$_{\mathsf{DKG}}$ protocol, and we perform its security analysis in Section 4. In Section 5, we provide experimental results for our T-PAKE$_{\mathsf{DKG}}$ implementation in the LAN setting. In Section 6, we present a brief overview of our T-PAKE$^{\mathsf{l}}_{\mathsf{DKG}}$ construction, asynchronous versions of our protocols and reduce the size of the required server public keys.

2 Preliminaries

2.1 Model

The goal of the T-PAKE protocol is to perform *distributed authenticated key exchange* (*dake*) between a client with a password and a subset of servers, such that after a protocol run the client would hold different session keys, one per server. For our T-PAKE construction, we use the same system model as in [32] (which extends [10, 30]), with slight relaxation regarding the number of the servers that can participate in a protocol run. We require $2t + 1$ to n servers to participate during a protocol instance, expect that the instance succeeds if at least $t + 1$ servers behave correctly. Moreover, even if t servers are corrupted and collude, the remaining session keys between a client and honest servers should be unknown to anybody else.

Protocol participants. There are two types of protocol participants: clients and servers. Let $ID := Clients \cup Servers$ be a non-empty set of protocol participants. We assume $Servers$ consists of n servers $\{S_1, \ldots, S_n\}$, and that these servers are meant to cooperate in authentication of a client. Each client $C \in Clients$ has a password p_C (denoted by just p later on). We assume that the password is drawn uniformly from the set of possible passwords $Password_C$ (in fact, results can be extended to other password distributions). Clients and servers are modeled as probabilistic polynomial time (PPT) algorithms with an input tape and an output tape. Multiple protocol instances with, possibly, different sets of participants are allowed. We denote by Π_i^U an instance i of participant $U \in ID$.

Communication. We assume that communication channels between a client and servers are public and unauthenticated; the goal of T-PAKE is to establish session keys itself. We assume a broadcast channel among the servers such that a message sent to the broadcast channel is delivered to all servers. Unlike in [21, 32], where the client does also have access to it, our client does not have access to the broadcast channel. Further, we assume pair-wise secure channels between the servers, which are used for distributed key generation before a protocol instance starts.

An adversary $\mathcal{A}$ is allowed to compromise up to t servers and has complete control over the network (except for the secure channels of uncorrupted servers), i.e. she can read, modify, or block any messages sent by parties via public channels and by corrupted servers via private channels. [2]

[2] $\mathcal{A}$ controls the whole network and can launch a DoS attack; however, prevention against these attacks is beyond the scope of this work.

Execution of the protocol. In the real world, a protocol determines how participants behave in response to inputs from their environment. In the formal model, these inputs are provided by the adversary.

Formally, the adversary is a probabilistic algorithm with a distinguished tape. Queries written to this tape are responded to by participants according to the protocol. As defined in [32], the allowed queries are the following:

- `Send`(U, i, M): causes message M to be sent to instance Π_i^U. This instance of participant U runs according to the protocol specification, updates its state appropriately and the output is given to the adversary.
- `Execute`$(C, i, ((S_{j1}, l_{j1}), \ldots, (S_{jk}, l_{jk})))$: causes P to be executed to completion between Π_i^C and $\Pi_{l_{j1}}^{S_{j1}}, \ldots, \Pi_{l_{jk}}^{S_{jk}}$, where k is a number of servers with $2t+1 \leq k \leq n$. This query captures passive eavesdropping of an execution.
- `Reveal`(C, i, S_j): causes the output of the session key sk_{C,S_j}^i held by Π_i^C corresponding to server S_j.
- `Reveal`(S_j, i): causes the output of the session key $sk_{S_j}^i$ held by $\Pi_i^{S_j}$. The `Reveal` queries model possible leakage of session keys due to compromise of a client/server, or cryptanalysis.
- `Test`(C, i, S_j): causes Π_i^C to flip a bit b. If $b = 1$, the adversary is given sk_{C,S_j}^i, otherwise the adversary is given a random key.
- `Test`(S_j, i): causes $\Pi_i^{S_j}$ to flip a bit b. If $b = 1$, the adversary is given $sk_{S_j}^i$, and if $b = 0$ the adversary is given a random key. The adversary is allowed only a single `Test` query (of either type) during the execution of P. The `Test` queries are used to define the protocol security; i.e., whether an adversary can distinguish a true session key from a random key.

Partnering. To reason about leakage of a password the notion of partnering has been introduced. Let *pid*, *sid* and *sk* denote a partner-id, a session-id and a session key, respectively. On accept, a server instance holds *pid*, *sid* and *sk*, and a client instance holds *pid*, *sid* and a set $(sk_{j_1}, \ldots, sk_{j_{2t+1}})$. Let *sid* include a common part of the messages sent by the client instance and a reconstructed shared response sent from $t+1$ or more servers to the client. (Broadcast messages are not part of *sid*, since a client cannot see them.) Then instances Π_i^C holding $(pid, sid, (sk_{j_1}, \ldots, sk_{j_{2t+1}}))$, where $pid = I$ for some $I = \{j_1, \ldots, j_{2t+1}\}$, and $\Pi_{l_j}^{S_j}$ holding (pid', sid', sk) are said to be *partnered* if $j \in I$, $pid' = C$, $sid = sid'$. Furthermore, correctness property requires that $sk_j = sk$.

Freshness. A client instance/server pair (Π_i^C, S_j) is said to be *fresh* if: (1) S_j is not compromised; (2) there has been no `Reveal`(C, i, S_j) query; and (3) if $\Pi_l^{S_j}$ and Π_i^C are partnered, there has been no `Reveal`(S_j, l) query. A server instance $\Pi_i^{S_j}$ is said to be *fresh* if: (1) S_j is not compromised; (2) there has been no `Reveal`(S_j, i) query; and (3) if Π_l^C and $\Pi_i^{S_j}$ are partnered, there has been no `Reveal`(C, l, S_j) query.

Advantage of the adversary. The advantage of the adversary against the protocol P is defined as follows. Let $Succ_P^{dake}(\mathcal{A})$ be the event that $\mathcal{A}$ makes a single Test query directed to some client instance/server pair (Π_i^C, S_j) that is *fresh* and where Π_i^C has terminated, or $\mathcal{A}$ makes a single Test query directed to some server instance $(\Pi_i^{S_j})$ that has terminated and is *fresh*, and eventually $\mathcal{A}$ outputs a bit b', where $b' = b$ for the bit b that was selected in the Test query. The *dake* advantage

of $\mathcal{A}$ attacking P is defined to be

$$Adv_P^{dake}(\mathcal{A}) := 2Pr[Succ_P^{dake}(\mathcal{A})] - 1.$$

Fact 2.1.

$$Pr[Succ_P^{dake}(\mathcal{A})] = Pr[Succ_{P'}^{dake}(\mathcal{A})] + \epsilon \quad \Longleftrightarrow \quad Adv_P^{dake}(\mathcal{A}) = Adv_{P'}^{dake}(\mathcal{A}) + 2\epsilon.$$

Let κ be the security parameter.

Definition 2.2. *Protocol P is a T-PAKE if, for all PPT adversaries $\mathcal{A}$ that make at most n_{in} online attacks and for all dictionary sizes N, there is a negligible function ϵ such that*

$$Adv_P^{dake}(\mathcal{A}) \leq \frac{n_{in}}{N} + \epsilon(\kappa).$$

The above definition ensures that the adversary cannot do significantly better than pure password guessing in online attacks. Formally, an instance Π_U^i presents an online attack if at the time of the $\texttt{Test}(U, i)$ query the following holds: 1) the adversary queried $\texttt{Send}(U, i, *)$; and 2) the adversary queried $\texttt{Reveal}(U, i)$ or $\texttt{Test}(U, i)$.

2.2 Cryptographic Assumption

Let G_q denote a cyclic group of prime order q, where $|q| = \kappa$. Let g be a generator of G_q. The security of our protocols depends upon the decisional Diffie-Hellman (DDH) assumption [12]. For $X = g^x$, $Y = g^y$, the Diffie-Hellman function is defined as $DH(X, Y) = g^{xy}$. Let $\mathcal{A}$ be an algorithm that on input (X, Y, Z) outputs "1" if it believes that $Z = DH(X, Y)$, and "0" otherwise. For any $\mathcal{A}$ running in time Δ,

$$\begin{aligned} Adv_{G_q}^{DDH}(\mathcal{A}) := |Pr\left[(x, y) \leftarrow_R (\mathbb{Z}_q)^2;\ (X, Y, Z) \leftarrow (g^x, g^y, g^{xy}) : A(X, Y, Z) = 1\right] \\ - Pr\left[(x, y, z) \leftarrow_R (\mathbb{Z}_q)^3; (X, Y, Z) \leftarrow (g^x, g^y, g^z) :\ A(X, Y, Z) = 1\right]| \end{aligned}$$

Assumption 1 (The DDH Assumption)**.** *For any PPT adversary $\mathcal{A}$, $Adv_{G_q}^{DDH}(\mathcal{A})$ is negligible in κ.*

2.3 Cryptographic Tools

Non-Interactive Simulation-Sound Zero-Knowledge Proofs—NIZK. We use NIZK proofs in our protocols to assure that the protocol messages are well-formed. An NIZK proof system for language $\mathcal{L}$ is a triple of algorithms consisting of a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and a simulator $\mathcal{S}$. As defined in [8], a proof system is $(T_S, q_P, \epsilon_{ZK}, \epsilon_{SS})$-simulation-sound zero-knowledge, if there is a simulator S running in time T_S which answers up to q_P prover queries on adversary's choice such that statistical difference between the view of an interaction with S and an interaction with the real prover is at most ϵ_{ZK}, and the probability that any adversary interacting with S outputs a correct

proof on a new false statement is at most ϵ_{SS}. The following proof systems are used in our protocol:

$$\begin{aligned}
\mathcal{L}^i_{S1} &= \{(a,b,\bar{a}) \in (G_q)^3 | \exists k \in \mathbb{Z}_q \; s.t. \; (a,b,\bar{a}) = (g^k, (c_p)^k, (\bar{g})^k)\} \\
\mathcal{L}_C &= \{(a,e,c_{\tilde{p}},d_{\tilde{p}},\hat{c}_{\tilde{p}},\hat{d}_{\tilde{p}}) \in (G_q)^6 \mid \exists (r_{\tilde{p}},\tilde{p}) \in (\mathbb{Z}_q)^2 \; s.t. \\
&\qquad (e,c_{\tilde{p}},d_{\tilde{p}},\hat{c}_{\tilde{p}},\hat{d}_{\tilde{p}}) = (a^{r_{\tilde{p}}}, g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}}h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}}(\hat{h})^{\tilde{p}})\} \\
\mathcal{L}^i_{S2} &= \{(z,a,\delta,P) \in (G_q)^4 \mid \exists (k,x) \in (\mathbb{Z}_q)^2 \; s.t. \; (y_i,a,z) = (g^x, g^k, \delta^k P^{-x})\}
\end{aligned}$$

These proof systems are generalizations of Schnorr's proof of discrete logarithm knowledge [35] and Chaum's proof of equality of discrete logarithms [20]; see also [18, 37]. Prover $\mathcal{P}$ essentially proves that appropriate secrets are used while computing values according to the protocol. Auxiliary generators $\bar{g}$, $\hat{g}$, $\hat{y}$, $\hat{h}$ allow to avoid the need for proof of knowledge and enable an efficient simulation of the protocol [8]. These proof systems achieve error bounds $\epsilon_{ZK} = (q_P \cdot q_H)/q$ and $\epsilon_{SS} = q_H/q$, where q_H is an upper bound on the number of adversary's queries to random oracles (hashes). The simulators' running time is the same as that of the provers.

Distributed Key Generation—DKG. The distributed key generation (DKG) allows n servers to generate a random value in a distributed fashion. We refer to the DKG protocol proposed by Gennaro *et al.* [24], which requires at most $n^2 + 5n + 2$ exponentiations per server and has the following properties provided that $n \geq 2t + 1$:
- any subset of honest servers of size $t + 1$ defines the same unique secret key k,
- all honest parties have the same public key $K = g^k$,
- the shared secret k is uniformly distributed at random in $\mathbb{Z}_q$.

The DKG protocol consists of two phases: a secret sharing of two random polynomials by each party via Pedersen's verifiable secret sharing (VSS) [34] and extracting a generated unique value via Feldman's VSS [22]. The protocol requires pair-wise secure and authenticated channels between the servers. Toward security analysis, a simulation of their DKG protocol [24] can be done efficiently; the simulator knows all of the adversary's secret values as the adversary provides enough secret shares of her polynomials to the honest parties (remember that $t < n/2$).

3 The T-PAKE$_{\mathsf{DKG}}$ Protocol

Given the efficiency of the PPSS [8] protocol, we use its password protection and verification techniques towards our protocol construction. PPSS, however, cannot tolerate any fault, and does not allow servers to learn if the password is correct or not. We perform communication among the servers using a broadcast channel to allow them to learn whether the password is correct. In order to tolerate servers faults, we replace independent randomness generation with a DKG-based threshold randomness generation, which results in our T-PAKE$_{\mathsf{DKG}}$ protocol described below.

Setup. Given a generator $g \in G_q$, n servers $\{S_1, S_2, \ldots, S_n\}$ with their respective indices $\{I_1, \ldots, I_n\}$, and a threshold t, the setup protocol generates a shared secret $x \leftarrow \mathbb{Z}_q$ and a public key $y = g^x$ such that each server S_i obtains a secret share $x_i \in \mathbb{Z}_q$ and the corresponding public key $y_i = g^{x_i}$ and any subset of the servers I of size greater than t can reconstruct the secret $x = \sum_{I_i \in I} \lambda_i x_i$, where λ_i are Lagrange interpolation coefficients.

Setup (on public parameters g, q, n, t):
$x \leftarrow_R \mathbb{Z}_q$, $y \leftarrow g^x$, $\{x_i\}_{i=1}^n \xleftarrow{(t+1,n)} SS(x)$, $(h, \hat{g}, \hat{h}, \hat{y}, \bar{g}) \leftarrow (G_q)^5$, $\{y_i \leftarrow g^{x_i}\}_{i=1}^n$.
Client initialization (on public parameters g, y, h and password p): $r_p \leftarrow_R \mathbb{Z}_q$,
$(c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$.
Public information: $g, h, y, \{y_j\}_{j=1}^n, \hat{g}, \hat{h}, \hat{y}, \bar{g}, (c_p, d_p)$.
Private keys: $\{x_i\}_{i=1}^n$. Server S_i has a private key x_i.
Batch offline (before Client Login): $(K := g^k, \{k_i\}_{i=1}^n) \xleftarrow{(t+1,n)} DKG$. Server S_i receives k_i.
Client Login:

C1 (Client C)	Send $(C, I = \langle I_1, \ldots, I_n \rangle)$ to the server S_i for all $I_i \in I$.
S1 (Server S_i)	Compute: $(a_i, b_i, \bar{a}_i) \leftarrow (g^{k_i}, (c_p)^{k_i}, \bar{g}^{k_i})$, $\pi_{1i} \leftarrow \mathsf{Prove}_{S1}((a_i, b_i, \bar{a}_i), k_i)$.
	Broadcast: $(a_i, b_i, \bar{a}_i, \pi_{1i})$, Send $(a_i, b_i, \bar{a}_i, \pi_{1i})$ to the client C.
C2 (Client C)	$I_C \leftarrow \{i \mid \mathsf{Verify}_{S1}((a_i, b_i, \bar{a}_i), (\pi_{1i}))\}_{i \in I}$. If $\lvert I_C \rvert < t+1$ then Abort.
	Pick $(\tilde{x}, r_{\tilde{p}}) \leftarrow_R (\mathbb{Z}_q)^2$. Compute: $\tilde{y} \leftarrow g^{\tilde{x}}$, $\{e_i \leftarrow (a_i)^{r_{\tilde{p}}}\}_{i \in I_C}$,
	$c_\beta \leftarrow \prod_{i \in I_C} (b_i / e_i)^{\lambda_{i, I_C}}$, $(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}})$,
	$K \leftarrow \prod_{i \in I_C} a_i^{\lambda_{i, I_C}}$, $\{\tau_i \leftarrow \langle \tilde{y}, a_i, K \rangle$, $\pi_{2i} \leftarrow \mathsf{Prove}_C((a_i, e_i, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), (r_{\tilde{p}}, \tilde{p}))$,
	$\tilde{y}_i \leftarrow (y_i)^{\tilde{x}}$, $\tilde{k}_i \leftarrow (a_i)^{\tilde{x}}$, $SK_i \leftarrow H_0(\tau_i, \tilde{y}_i, \tilde{k}_i)$ $\}_{i \in I_C}$.
	Send $(\tilde{y}, c_\beta, e_i, (c_{\tilde{p}}, d_{\tilde{p}}), (\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), \pi_{2i})$ to server S_i for all $i \in I_C$.
S2 (Server S_i)	$I_{S_i} \leftarrow \{j \mid \mathsf{Verify}_{S1}((a_j, b_j, \bar{a}_j), (\pi_{1j}))\}_{j \in I}$.
	If $\neg\mathsf{Verify}_C((a_i, e_i, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), (\pi_{2i}))$, then Abort. $\tau_i \leftarrow \langle \tilde{y}, a_i, K \rangle$.
	Compute: $w_i \leftarrow (c_\beta)^{x_i}$, $d_{\beta_i} \leftarrow (d_p / d_{\tilde{p}})^{k_i}$, $z_i \leftarrow d_{\beta_i} / w_i$,
	$\pi_{3i} \leftarrow \mathsf{Prove}_{S2}((z_i, a_i, d_p / d_{\tilde{p}}, c_\beta), (k_i, x_i))$. Broadcast (z_i, π_{3i}).
S3 (Server S_i)	$I'_{S_i} \leftarrow \{j \mid \mathsf{Verify}_{S2}((z_j), (\pi_{3j}))\}_{j \in I_{S_i}}$. If $\lvert I'_{S_i} \rvert < t+1$ then Abort. Compute:
	$\bar{z} \leftarrow \prod_{j \in I'_{S_i}} (z_j)^{\lambda_{j, I'_{S_i}}}$. If $\bar{z} \neq 1$ then Abort.
	Compute: $\tilde{y}_i \leftarrow \tilde{y}^{x_i}$, $\tilde{k}_i \leftarrow \tilde{y}^{k_i}$, $SK_i \leftarrow H_0(\tau_i, \tilde{y}_i, \tilde{k}_i)$.

Figure 1: Protocol: T-PAKE$_{\mathsf{DKG}}$

We can assume that these values are generated by a trusted dealer using Shamir secret sharing $SS(\cdot)$ [36]; however, it can easily be done in a completely distributed fashion using DKG. We also require random generators $h, \hat{g}, \hat{h}, \hat{y}, \bar{g}$ of G_q as additional public parameters. Let $H_0 \leftarrow_R \Omega$ be a random oracle with domain and range defined by the context of its use, where Ω denotes the set of all functions H from $\{0,1\}^*$ to $\{0,1\}^\infty$, as defined in [32]. The adversary does not participate in the setup phase.

Client Initialization. A password p for a client C is transfered to servers as an ElGamal-like encryption $(c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$ with $r_p \leftarrow_R \mathbb{Z}_q$. Notice that no private information is required to generate (c_p, d_p), and we assume that it is known to the adversary.

Client Login. A client C receives as input a set I of n servers and sends a trigger message to each of them. Upon a trigger message, the server S_i broadcasts (and responds separately to the

client C with) $(a_i, b_i, \bar{a}_i) \leftarrow (g^{k_i}, (c_p)^{k_i}, \bar{g}^{k_i})$, while $k_i \leftarrow DKG(I, t+1, n)$ values are kept secret until the protocol completion and are deleted afterwards. Note that the DKG instances are ran as a background process, and the servers keep k_i values ready before a protocol instance starts. The client collects all servers' messages and prepares an answer for each server separately. First, he adds his fresh randomness $r_{\tilde{p}} \leftarrow \mathbb{Z}_q$ to each a_i as $e_i \leftarrow (a_i)^{r_{\tilde{p}}}$. Second, he prepares $c_\beta \leftarrow \prod_{i \in I_C} (b_i/e_i)^{\lambda_{i,I_C}}$ which will be used in threshold-decryption on servers' side, where λ_{i,I_C} denotes a Lagrange coefficient for the server S_i from the set I_C. Third, the client encrypts the password as $(c_{\tilde{p}}, d_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}})$, where $\tilde{p}$ is his (possibly) correct password and of course kept secret. At last, the client generates a fresh key pair $\tilde{x} \leftarrow \mathbb{Z}_q$, $\tilde{y} \leftarrow g^{\tilde{x}}$ for the key exchange with the servers.

Upon receiving a message from the client, each server S_i broadcasts its partial decryption as z_i. The following holds: $z_i = d_{\beta_i}/w_i = (d_p/d_{\tilde{p}})^{k_i} \cdot (c_\beta)^{-x_i} = (y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^{k_i} \cdot (c_\beta)^{-x_i}$. Upon receiving broadcast messages, each server S_i calculates the following value:

$$\begin{aligned}\bar{z} &= \prod_{j \in I'_{S_i}} (z_j)^{\lambda_{j,I'_{S_i}}} = (y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^{\sum_{j \in I'_{S_i}} \lambda_j k_j} \cdot (c_\beta)^{\sum_{j \in I'_{S_i}} (-\lambda_j x_j)} \\ &= (y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^k \cdot (c_\beta)^{-x} = (y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^k \cdot (g^{(r_p - r_{\tilde{p}}) \sum_{j \in I_C} k_j \lambda_j})^{-x} = h^{(p-\tilde{p})k}.\end{aligned}$$

If the password is correct, i.e. $p = \tilde{p}$, then $\bar{z}$ evaluates to 1, otherwise it is a random value. Thus, $\bar{z}$ does not leak any information about the password (except for its correctness), as it includes randomness generated using DKG which is discarded immediately after the T-PAKE instance. Once correctness of the password is verified, each server S_i computes a session key SK_i. The session key is obtained by applying a random oracle hash function H_0 on DH tuples computed using the client's challenge public key $\tilde{y} = g^{\tilde{x}}$, the public key $y_i = g^{x_i}$ and the DKG share $a_i = g^{k_i}$ of server S_i. All communication between the client and the servers as well as all broadcast messages are accompanied by the appropriate NIZK proofs of correct computations (refer to Section 2.3). Note that we do not include $\tilde{y}$ in our NIZK proof as the current proof is sufficient to stop replay attacks. In Figure 1, we present the complete T-PAKE$_{\mathsf{DKG}}$ protocol. It requires two broadcasts, as in the T-PAKE$_{\mathsf{DKG}}$ compromised servers share enough information to the honest servers and d_{β_j} values can be easily simulated as long as $n \geq 2t+1$.

Assuming that the adversary compromises at most t servers, the protocol maintains security as well as the completion guarantee for $n \geq 2t+1$; i.e., to complete a protocol instance, $t+1$ honest servers should be online. For $n < 2t+1$ the protocol is still secure, however its completion guarantee reduces to that of T-PAKE in [32]. As a client may contact fewer than n servers during his T-PAKE authentication instance, during a T-PAKE$_{\mathsf{DKG}}$ execution in a multi-client system, it is important to ensure that the same DKG instance employed by every participating server. Assuming that the employed broadcast primitive sends every message to all (participating and non-participating) servers, this DKG synchronization can be trivially achieved.

Although our T-PAKE$_{\mathsf{DKG}}$ protocol employs the same password protection and verification techniques as the PPSS protocol [8], the two protocols differ significantly. Along with the already discussed conceptual differences, the two constructions also differ technically in the following two ways:

i) In T-PAKE$_{\mathsf{DKG}}$, the servers send z_i values in unencrypted form as these values cannot relieve any information about the password once the k_i values are discarded.

P_0	The original protocol T-PAKE$_{\mathsf{DKG}}$.
P_1	Nonces are distinct.
P_2	ZK proofs are simulated.
P_3	Key exchange is replaced with a perfect key exchange.
P_4	Value $\hat{d_{\tilde{p}}}$ from a client is replaced with a random value.
P_5	Authentication of an honest client is changed so that uncompromised servers compute z_i values without using their secret shares. Also, value $\hat{y}$ is generated with a known discrete log.
P_6	The adversary succeeds if it ever sends a $d_{\tilde{p}}$ value associated with a correct password.
P_7	On any adversary login attempt for C, values d_{β_i} and z_i are replaced with random values.
P_8	Value $d_{\tilde{p}}$ from an honest client is replaced with a random value.
P_9	In initialization procedure, values (c_p, d_p) are replaced with random values.

Figure 2: Informal description of the protocols P_0 through P_9

ii) The reconstruction step of PPSS is not executed by a client; it is rather executed by each server in T-PAKE$_{\mathsf{DKG}}$ to verify the client password.

4 Security Analysis

A secure T-PAKE protocol requires that an adversary cannot determine session keys with significantly better advantage than that of an online dictionary attack. In this section, we prove that T-PAKE$_{\mathsf{DKG}}$ is secure under the DDH assumption in the random oracle model given that the adversary can statically corrupt up to t out of n servers. The security proof goes essentially similar to one of [32].

4.1 Security Proof

Theorem 4.1. *Let P be the protocol T-PAKE$_{\mathsf{DKG}}$ with n servers described in Figure 1, using group G_q, and with a password dictionary of size N (that may be mapped into $\mathbb{Z}_q^*$). Fix an adversary $\mathcal{A}$ that runs in time Δ, makes n_{ex}, n_{re} queries of type* `Execute`, `Reveal`*, respectively, makes n_{ro} queries to the random oracles, and starts at most n_{in} client and server instances. Then for $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$, where Δ_{exp} is the time-cost of a single exponentiation:*

$$Adv_P^{dake}(\mathcal{A}) \leq \frac{n_{in}}{N} + O(n(n_{ex} + n_{in})) \cdot Adv_{G_q}^{DDH}(\Delta') + O\left(\frac{n(n_{in} + nn_{ex})(n_{ex} + n_{in}n_{ro})}{q}\right)$$

Proof. We introduce series of protocols P_0, P_1,...,P_9 (see Figure 2) and show that the difference between the advantage of $\mathcal{A}$ attacking protocols P_i and P_{i+1} is negligible. Below, we give an informal description of the protocol reductions, and later provide the details.

$P_0 \rightarrow P_1$: The probability of a collision of nonces is negligible.

$P_1 \rightarrow P_2$: The simulation of SS-NIZNPs is statistically indistinguishable from the real proofs, except for a simulation error, which is negligible.

$P_2 \to P_3$: This can be shown using a reduction from DDH. On input (X, Y, Z), we plug in random powers of Y for the global public key K generated by DKG, and random powers of X for the clients' $\tilde{y}$ values, and then check H_0 queries for appropriate powers of Z.

$P_3 \to P_4$: This can be shown using a reduction from DDH. On input (X, Y, Z), we plug in random powers of Y for $\hat{y}$, random powers of X for $\hat{c}_{\tilde{p}}$, and $\hat{d}_{\tilde{p}}$ contains Z values. If (X, Y, Z) is a true DH tuple, then all values are of correct form. Otherwise $\hat{d}_{\tilde{p}}$ is a random value.

$P_4 \to P_5$: This is straightforward, since the view of the adversary is indistinguishable in these two protocols.

$P_5 \to P_6$: This is straightforward, since this could only increase the probability of the adversary succeeding. Below we will use the fact that $DLog_{\hat{g}}(\hat{h})$ is known, and that z_i value computed when authenticating a client's $d_{\tilde{p}}$ value by an uncompromised server does not use the secret share x_i of that server.

$P_6 \to P_7$: This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, random powers of X for a_i, and construct d_{β_i} and z_i in such way that they include Z. If (X, Y, Z) is a true DH tuple, then all values are of correct form. Otherwise d_{β_i} and z_i are random values.

$P_7 \to P_8$: This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, random powers of X for $c_{\tilde{p}}$, and construct $d_{\tilde{p}}$ value in such way that it contains Z. If (X, Y, Z) is a true DH tuple, then all values are of correct form. Otherwise $d_{\tilde{p}}$ is a random value.

$P_8 \to P_9$: This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for c_p, X for y, and construct d_p in such way that it contains Z. If (X, Y, Z) is a true DH tuple, then all values are of correct form. Otherwise d_p is a random value.

Let J be a set a corrupted servers. For any uncompromised server i, $J_i = J \cup \{i\}$.

Protocol P_1. Let E be the event that one or more clients generate the same $\tilde{y}$ value in different queries to a client in step **C2** (let's call it E_1), or that DKG for k_i values generates the same global value g^k in two different instances (E_2), or that a server S_i holds the same k_i value in two different instances (E_3), or that in the initialization procedure two servers receive (generate) the same values y_i and y_j (E_4). Let P_1 be a protocol that is identical to P_0 except that if E occurs, the protocol aborts (and thus the adversary fails).

Note that if E does not occur, then it will never be the case that $\mathcal{A}$ makes a $\texttt{Reveal}(C, i, S_j)$ query where Π_i^C generated key $sk^i_{C,S_j} = H_0(\tau_j, \tilde{y}_j, \tilde{k}_j)$ or a $\texttt{Reveal}(S_j, i)$ query where $\Pi_i^{S_j}$ generated key $sk^i_{S_j} = H_0(\tau_j, \tilde{y}_j, \tilde{k}_j)$, and there is another client or server instance that generates a key using $H_0(\tau_j, \tilde{y}_j, \tilde{k}_j)$ that is not a partner to the instance corresponding to the `Reveal` query.

Claim 4.2. *For any adversary $\mathcal{A}$,*

$$Adv^{dake}_{P_0}(\mathcal{A}) \leq Adv^{dake}_{P_1}(\mathcal{A}) + \frac{O(n^2 + (n_{in} + (n+1)n_{ex})^2)}{q}.$$

Proof. The probability that two server keys y_i and y_j are equal is bounded by $P[E_4] \leq \frac{1}{q} + \ldots + \frac{n-1}{q} = \frac{O(n^2)}{q}$. Similarly, $P[E_3] \leq \frac{O((nn_{ex})^2)}{q}$. Here we rely on fact that at least one honest server participates in generating k_i value (although we consider DKG as a black-box). Finally,

$P[E_1] \leq \frac{O((n_{in}+n_{ex})^2)}{q}$ and $P[E_2] \leq \frac{O((n_{in}+n_{ex})^2)}{q}$. □

Protocol P_2. Let P_2 be a protocol that is identical to P_1 except that ZK proofs $(\mathcal{L}^i_{S1}, \mathcal{L}_C, \mathcal{L}^i_{S2})$ are simulated. If simulation of ZK fails, P_2 aborts, and we say the adversary fails.

Claim 4.3. *For any adversary $\mathcal{A}$,*

$$Adv^{dake}_{P_1}(\mathcal{A}) \leq Adv^{dake}_{P_2} + \frac{O((n_{in}+nn_{ex})(n_{ro}+n_{in}+nn_{ex}))}{q}.$$

Proof. P_2 and P_1 are statistically indistinguishable except for the case when simulation of ZK fails. This is bounded by

$$\begin{aligned}
&\textsc{Simerr}_{S1}(n_{ro}, n_{in}+nn_{ex}) + \textsc{Simerr}_C(n_{ro}, n_{in}+n_{ex}) + \textsc{Simerr}_{S2}(n_{ro}, n_{in}+nn_{ex}) \\
&\leq \frac{(n_{in}+nn_{ex})(n_{ro}+n_{in}+nn_{ex})}{q} + \frac{(n_{in}+n_{ex})(n_{ro}+n_{in}+n_{ex})}{q^2} \\
&\quad + \frac{(n_{in}+nn_{ex})(n_{ro}+n_{in}+nn_{ex})}{q^2} \leq \frac{3(n_{in}+nn_{ex})(n_{ro}+n_{in}+nn_{ex})}{q}.
\end{aligned}$$

□

Simulation Soundness. We define Fraud as the event that the adversary is able to provide a valid proof for a statement that does not satisfy the particular relation corresponding to that type of SS-NIZKP. Let E_1, E_2 and E_3 be such events for $\mathcal{L}_{S1}$, $\mathcal{L}_C$, $\mathcal{L}^i_{S2}$. Let $\mathsf{Fraud} = E_1 \vee E_2 \vee E_3$.

Claim 4.4. *For any adversary $\mathcal{A}$ running against P_2,*

$$Pr(\mathsf{Fraud}) = \epsilon_{SS} \leq \frac{3nn_{in}(n_{ro}+1)}{q}.$$

Proof. The proof of Claim 4.4 is similar to the proof of Claim 5.5 in [32]. Let ϵ be the probability that E_1 occurs when $\mathcal{A}$ is running against protocol P_2. We construct an algorithm D that simulates P_2 with the following changes: D guesses which server query will cause E_1 to occur, and on that query D outputs the pair $((a_j, b_j, \bar{a}_j), \pi_{1j})$ associated with that query and halts. D outputs a valid proof that doesn't satisfy $\mathcal{L}_{S1}$ with probability $\frac{\epsilon}{nn_{in}}$. This implies $\textsc{Serr}_{S1}(n_{ro}, n_{in}+n_{ex}) \geq \frac{\epsilon}{nn_{in}}$, where $\textsc{Serr}(n_{ro}, n_{pr})$ is the soundness error of the corresponding ZK protocol given n_{ro} random oracle queries and n_{pr} proof queries. For this protocol, $\textsc{Serr}_{S1} \leq \frac{n_{ro}+1}{q}$, so $Pr[E_1] \leq \frac{nn_{in}(n_{ro}+1)}{q}$.

In a similar way, we can show that $Pr[E_2] \leq \frac{n_{in}(n_{ro}+1)}{q}$, and $Pr[E_3] \leq \frac{nn_{in}(n_{ro}+1)}{q}$. □

Protocol P_3. Let E be the event that $\mathcal{A}$ makes an $H_0(\tau_i, \tilde{y}_i, \tilde{k}_i)$ query for a value $\tilde{k}_i = DH(\tilde{y}, a_i)$ for some a_i belonging to an uncompromised server S_i, and for some $\tilde{y}$ generated in a query to a client C that generated τ_i, where $\tilde{y}_i$ can have any value. Let P_3 be a protocol that is identical to P_2 except if E occurs, the protocol aborts, and we say adversary fails.

Claim 4.5. *For any adversary $\mathcal{A}$ running in time Δ, there is a $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$ such that*

$$Adv_{P_2}^{dake}(\mathcal{A}) \leq Adv_{P_3}^{dake}(\mathcal{A}) + 2Adv_{G_q}^{DDH}(\Delta') + \frac{O(n_{in} + n_{ex} + n_{ro}n)}{q}.$$

Proof. Let ϵ be the probability that E occurs when $\mathcal{A}$ is running against protocol P_2. Then $Pr(Succ_{P_2}^{dake}) \leq Pr(Succ_{P_3}^{dake}(\mathcal{A})) + \epsilon$, and thus by Fact 2.1, $Adv_{P_2}^{dake} \leq Adv_{P_3}^{dake} + 2\epsilon$.

We construct an algorithm D that tries to distinguish between valid DH triples and random triples by running $\mathcal{A}$ on a simulation of the protocol. Given triple (X, Y, Z), D simulates P_2 for $\mathcal{A}$ with the following changes:

1. In the initialization phase, $\bar{g} \leftarrow g^{r_0}$, where $r_0 \leftarrow_R \mathbb{Z}_q$.
2. In the batch offline phase, replace DKG for k_i values with a simulation: $K \leftarrow Y^{\phi}$, where $\phi \leftarrow_R \mathbb{Z}_q$, and for each uncompromised server S_i,
$$a_i \leftarrow \left(\frac{Y^{\phi}}{\prod_{j \in J} g^{k_j \cdot \lambda_{j,J_i}}}\right)^{(\lambda_{i,J_i})^{-1}}.$$
3. In step **S1** in a query to an uncompromised server S_i, use a_i generated in a simulation of DKG, $b_i \leftarrow a_i^{r_p}$, $\bar{a} \leftarrow a_i^{r_0}$.
4. In step **C2** in a query to a client C, $\tilde{y} \leftarrow X^{\psi}$, where $\psi \leftarrow_R \mathbb{Z}_q$. Then for each uncompromised server S_i, $SK_i \leftarrow_R \{0,1\}^{\kappa}$.
5. In step **S2** in a query to an uncompromised server S_i, $d_{\beta_i} \leftarrow \left(\frac{b_i}{e_i}\right)^x$.
6. In step **S3** in a query to an uncompromised server S_i, if $\tilde{y}$ was generated in a query to a client C, $SK \leftarrow SK_i$ for the SK_i value generated by that query.
7. In step **S3** in an H_0 query, if the query is $(\tau_i, \tilde{y}_i, \left(\frac{Z^{\psi\phi}}{\prod_{j \in J} \tilde{y}^{k_j \lambda_{j,J_i}}}\right)^{(\lambda_{i,J_i})^{-1}})$, for τ_i, ψ generated in a query to a client C, ϕ generated in the pre-initialization phase and $\tilde{y}_i$ being any value, D outputs 1 and halts.
8. If $\mathcal{A}$ finishes, D outputs 0 and halts.

If (X, Y, Z) is a DH triple, this simulation is perfectly indistinguishable from P_2 until E occurs, when the simulation halts and D outputs 1. If (X, Y, Z) is a random triple, then D outputs 1 only if $\mathcal{A}$ happens to query H_0 with a third parameter equal to $Z^{\psi\phi}$. This could happen if ψ or ϕ generated by the simulation is zero, or if none of them is zero but the Z value is such that one of the n_{ro} queries made by the adversary to H_0 has the third parameter equal to $Z^{\psi\phi}$. The former probability is at most $\frac{n_{in}+n_{ex}}{q}$, and the latter is at most $\frac{n_{ro}n}{q}$. Let Δ' be the running time of D, and note that $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$. The advantage of D is

$$\begin{aligned} Adv_{G_q}^{DDH}(D) &= Pr[D \text{ outputs } 1|\text{DH triple}] - Pr[D \text{ outputs } 0|\text{DH triple}] \\ &\geq \epsilon - \frac{n_{in} + n_{ex} + n_{ro}n}{q}. \end{aligned}$$

The claim follows from the fact that $Adv^{DDH}_{G_q}(D) \leq Adv^{DDH}_{G_q}(\Delta')$. □

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that in step **C2** in a query to a client C, $\hat{d}_{\tilde{p}} \leftarrow Z_q$.

Claim 4.6. *For any adversary $\mathcal{A}$ running in time Δ, there is a $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$ such that*

$$Adv^{dake}_{P_3}(\mathcal{A}) \leq Adv^{dake}_{P_4}(\mathcal{A}) + 2(n_{in} + n_{ex})\left(Adv^{DDH}_{G_q}(\Delta') + \frac{3nn_{in}(n_{ro}+1)}{q} + \frac{1}{q}\right).$$

Proof. We use a hybrid argument over $n_{in} + n_{ex}$ sessions. For each $l \in [0, n_{in} + n_{ex}]$, we define an intermediate protocol P_3^l which follows P_4 on the first l sessions, and on the remaining sessions it follows P_3. Clearly, $P_3^0 \equiv P_3$ and $P_3^{n_{in}+n_{ex}} \equiv P_4$.

Let's consider sessions l and $l+1$. Assume $Adv^{dake}_{P_3^l} = Adv^{dake}_{P_3^{l+1}} + 2\epsilon$. By Fact 2.1, it follows that $Pr(Succ^{dake}_{P_3^l}(\mathcal{A})) = Pr(Succ^{dake}_{P_3^{l+1}}(\mathcal{A})) + \epsilon$.

We construct an algorithm D that tries to distinguish between valid DH and random triples by running $\mathcal{A}$ on a simulation of the protocol.

Given (X, Y, Z), D simulates P_3^{l+1} with the following changes:

1. In the initialization phase, $\hat{g} \leftarrow g^{r_0}$, $\bar{g} \leftarrow X^{r_1}$, $\hat{y} \leftarrow Y$, where $(r_0, r_1) \leftarrow_R (\mathbb{Z}_q)^2$.
2. In step **C2** in a query to a client C, $c_{\tilde{p}} \leftarrow X^{1/r_0}$, $d_{\tilde{p}} \leftarrow X^{x/r_0} h^{\tilde{p}}$, $\hat{c}_{\tilde{p}} \leftarrow X$, $\hat{d}_{\tilde{p}} \leftarrow Z^{1/r_0} \hat{h}^{\tilde{p}}$, and for each server S_j, $e_j \leftarrow (\bar{a}_j)^{1/(r_0 r_1)}$.
3. If $\mathcal{A}$ succeeds the simulation, D outputs 1, else D outputs 0 and halts.

If (X, Y, Z) is a true DH triple, then D is statistically indistinguishable from P_3^l, because the values from a client are correctly computed, with $r_{\tilde{p}} = a/r_0$ and $\hat{x} = DH(\hat{g}, \hat{y}) = b/r_0$. Note there is a ϵ_{SS} probability that simulation of proofs fails. If (X, Y, Z) is a random triple, then D is statistically indistinguishable from P_3^{l+1}.

Let Δ' be the running time of D, and note that $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$. The advantage of D is

$$\begin{aligned} Adv^{DDH}_{G_q}(D) &= Pr[D \text{ outputs } 1 | \text{DH triple}] - Pr[D \text{ outputs } 1 | \text{random triple}] \\ &\geq Pr[Succ^{dake}_{P_3^l}(\mathcal{A})] - \epsilon_{SS} - Pr[Succ^{dake}_{P_3^{l+1}}(\mathcal{A})] - \frac{1}{q} = \epsilon - \epsilon_{SS} - \frac{1}{q}. \end{aligned}$$

The claim follows from the fact that $Adv^{DDH}_{G_q}(D) \leq Adv^{DDH}_{G_q}(\Delta')$. □

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except for the following:

1. In the initialization phase, $\hat{x} \leftarrow \mathbb{Z}_q$ is generated, and $\hat{y} \leftarrow (\hat{g})^{\hat{x}}$.
2. In step **C2** in a query to a client C, $d_{\tilde{p}} \leftarrow y^{r_{\tilde{p}}} h^p$, and $\hat{d}_{\tilde{p}} \leftarrow (\hat{y})^{r_{\tilde{p}}} (\hat{h})^p$.
3. In step **S2** in a query to an uncompromised server S_i that uses a $d_{\tilde{p}}$ value produced in a client query (i.e., $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}}) = (\hat{h})^p$), $z_i \leftarrow \left(\prod_{j \in J} \left(\frac{d_{\beta_j}}{c_\beta{}^{x_j}}\right)^{\lambda_{j,J_i}}\right)^{-1/\lambda_{i,J_i}}$. Note that this computation does not rely on the secret keys x_i or k_i of uncompromised servers.

Claim 4.7. *For any adversary $\mathcal{A}$,*

$$Adv^{dake}_{P_4}(\mathcal{A}) \leq Adv^{dake}_{P_5}(\mathcal{A}) + (n_{in} + n_{ex}) \cdot \frac{O(nn_{in}n_{ro})}{q}.$$

Proof. We use a hybrid argument over $n_{in} + n_{ex}$ sessions. For each $l \in [0, n_{in} + n_{ex}]$, we define an intermediate protocol P_4^l which follows P_5 on the first l sessions, and on the remaining sessions it follows P_4. Clearly, $P_4^0 \equiv P_4$ and $P_4^{n_{in}+n_{ex}} \equiv P_5$. Let's consider sessions l and $l+1$. We show that if Fraud doesn't occur, P_5 is perfectly indistinguishable from P_4. For this we simply need to show that in a query to an uncompromised server S_j, z_j values computed in P_4^l and P_4^{l+1} will be the same. □

Protocol P_6. Let P_6 be a protocol that is identical to P_5 except that in a query to an uncompromised server S_i for a client C and using parameters $par^* = (\tau_i, e_i, c_\beta, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$ where par^* was never used in a query to a client C, if $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}}) = (\hat{h})^p$, then P_6 stops and we say the adversary succeeds.

Claim 4.8. *For any adversary $\mathcal{A}$, $Adv^{dake}_{P_5}(\mathcal{A}) \leq Adv^{dake}_{P_6}(\mathcal{A})$.*

Proof. By having P_6 stop and saying that an adversary succeeds, we could only increase its probability of success. □

Protocol P_7. Let P_7 be a protocol that is identical to P_6 except for the following:

1. In step **S2** in a query to an uncompromised server S_i, $d_{\beta_i} \leftarrow \mathbb{Z}_q$.
2. In step **S2** in a query to an uncompromised server S_i, that uses values $(c_{\tilde{p}}, d_{\tilde{p}})$ not used in any query to a client C, and that uses a τ' value, compute z_j as follows. If there is a $(\bar{z}^*, \tau')$ pair recorded for this τ', use that $\bar{z}^*$, else choose $\bar{z}^* \leftarrow_R \mathbb{Z}_q$ and record $(\bar{z}^*, \tau')$. Then $z_i \leftarrow \left(\bar{z}^* \cdot \prod_{j \in J} \left(\frac{{c_\beta}^{x_j}}{d_{\beta_j}} \right)^{\lambda_{j,J_i}} \right)^{1/\lambda_{i,J_i}}$.

Claim 4.9. *For any adversary $\mathcal{A}$ running in time Δ, there is a $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$ such that*

$$Adv^{dake}_{P_6}(\mathcal{A}) \leq Adv^{dake}_{P_7}(\mathcal{A}) + nn_{ex} \cdot \left(2Adv^{DDH}_{G_q}(\Delta') + \frac{O(nn_{in}n_{ro})}{q} \right).$$

Proof. We use a hybrid argument over $n_{in} + n_{ex}$ sessions. For each $l \in [0, n_{in} + n_{ex}]$, we define an intermediate protocol which follows P_7 on the first l sessions, and on the remaining sessions it follows P_6. Clearly, $P_6^0 \equiv P_6$ and $P_6^{n_{in}+n_{ex}} \equiv P_7$.

Let's consider sessions l and $l+1$. Assume $Adv^{dake}_{P_6^l} = Adv^{dake}_{P_6^{l+1}} + 2\epsilon$. By Fact 2.1, it follows that $Pr(Succ^{dake}_{P_6^l}(\mathcal{A})) = Pr(Succ^{dake}_{P_6^{l+1}}(\mathcal{A})) + \epsilon$.

We construct an algorithm D that tries to distinguish between valid DH and random triples by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y, Z), D simulates P_6^{l+1} with the following changes:

1. In the initialization phase, $(y, h, \hat{h}, \bar{g}, \hat{g}) \leftarrow (Y, Y^{r_0}, Z^{r_1}, g^{r_2}, Z^{r_3})$, where $r_0, r_1, r_2, r_3 \leftarrow \mathbb{Z}_q$.

2. In the batch offline phase, replace DKG for k_i values with a simulation: $K \leftarrow X$, and for each uncompromised server S_i,
$$a_i \leftarrow \left(\frac{X}{\prod_{j \in J} g^{k_j \cdot \lambda_{j,J_i}}} \right)^{(\lambda_{i,J_i})^{-1}}.$$
3. In step **S1** in a query to an uncompromised server S_i, $(b_i, \bar{a}_i) \leftarrow ({a_i}^{r_p}, {a_i}^{r_2})$.
4. In step **S2** in a query to an uncompromised server S_i,
$$d_{\beta_i} \leftarrow \left(\frac{Z^{r_0 p} \cdot Z^{r_p}}{\prod_{j \in J} {d_p}^{k_j \cdot \lambda_{j,J_i}}} \cdot \frac{\prod_{j \in J} {d_{\tilde{p}}}^{k_j \cdot \lambda_{j,J_i}}}{\left(\hat{d}_{\tilde{p}} / (\hat{c}_{\tilde{p}})^{\hat{x}} \right)^{r_0/r_1} \cdot (\hat{c}_{\tilde{p}})^{1/r_3}} \right)^{(\lambda_{i,J_i})^{-1}}.$$
5. In step **S2** in a query to an uncompromised server S_i,
$$z_i \leftarrow d_{\beta_i} \cdot \left(\frac{\prod_{j \in J} {c_\beta}^{x_j \cdot \lambda_{j,J_i}}}{Z^{r_p} \cdot (\hat{c}_{\tilde{p}})^{-1/r_3}} \right)^{(\lambda_{i,J_i})^{-1}}.$$

We show that if Fraud does not occur, when (X, Y, Z) is a DH triple, then the simulation of the server query (for d_{β_i}) is statistically indistinguishable from the server query (for d_{β_i}) in P_6^l. When (X, Y, Z) is a random triple, then the simulation of the server query (for d_{β_i}) is statistically indistinguishable from the server query in P_6^{l+1}, the statistical difference coming from the $\frac{1}{q}$ probability that (X, Y, Z) is actually a DH triple, and probability that Fraud occurs.

Assume a client C tries to authenticate. Without loss of generality, we may assume that:

$$\begin{aligned}
(c_p, d_p) &= (g^{r_p}, y^{r_p} h^p), \quad (a_j, b_j, \bar{a}_j) = (g^{k_j}, g^{r_p k_j}, \bar{g}^{k_j}), \\
(c_{\tilde{p}}, d_{\tilde{p}}) &= (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}), \quad (\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}) = (\hat{g}^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}}), \quad e_j = g^{k_j r_{\tilde{p}}}, \\
c_\beta &= \prod_{j \in I_C} \left(\frac{b_j}{e_j} \right)^{\lambda_{j,I_C}} = \prod_{j \in I_C} \left(\frac{g^{r_p k_j}}{g^{k_j r_{\tilde{p}}}} \right)^{\lambda_{j,I_C}} \\
&= g^{(r_p - r_{\tilde{p}}) \sum_{j \in I} \lambda_{j,I_C} k_j} = g^{(r_p - r_{\tilde{p}}) k}, \\
d_{\beta_j} &= \left(\frac{d_p}{d_{\tilde{p}}} \right)^{k_j} = \left(\frac{y^{r_p} h^p}{y^{r_{\tilde{p}}} h^{\tilde{p}}} \right)^{k_j} = (y^{r_p - r_{\tilde{p}}} h^{p - \tilde{p}})^{k_j}, \\
z_j &= \frac{d_{\beta_j}}{(c_\beta)^{x_j}} = \frac{(y^{r_p - r_{\tilde{p}}} h^{p - \tilde{p}})^{k_j}}{g^{(r_p - r_{\tilde{p}}) k x_j}}, \\
\bar{z} &= \prod_{j \in I_{S_i}} (z_j)^{\lambda_{j,I_{S_i}}} = \prod_{j \in I_{S_i}} \left(\frac{(y^{r_p - r_{\tilde{p}}} h^{p - \tilde{p}})^{k_j}}{g^{(r_p - r_{\tilde{p}}) k x_j}} \right)^{\lambda_{j,I_{S_i}}} \\
&= \frac{(y^{r_p - r_{\tilde{p}}} h^{p - \tilde{p}})^{\sum_{j \in I_{S_i}} \lambda_{j,I_{S_i}} k_j}}{(g^{(r_p - r_{\tilde{p}}) k})^{\sum_{j \in I_{S_i}} \lambda_{j,I_{S_i}} x_j}} \\
&= \frac{(y^{r_p - r_{\tilde{p}}} h^{p - \tilde{p}})^k}{g^{(r_p - r_{\tilde{p}}) k x}} = \frac{y^{(r_p - r_{\tilde{p}}) k} h^{(p - \tilde{p}) k}}{y^{(r_p - r_{\tilde{p}}) k}} = h^{(p - \tilde{p}) k},
\end{aligned}$$

for some $r_{\tilde{p}}, \{k_j\}_{j \in I}, \tilde{p} \in \mathbb{Z}_q$ and k value generated by DKG.

If Fraud does not occur, when (X, Y, Z) is drawn from the set of DH triples, $d_{\beta_j} = (y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^{k_j}$, $z_j = \dfrac{(y^{r_p - r_{\tilde{p}}} h^{p-\tilde{p}})^{k_j}}{g^{(r_p - r_{\tilde{p}})k x_j}}$ for $j \in J$, and $\bar{z} = Z^{r_0(p-\tilde{p})} = h^{(p-\tilde{p})k}$, as in P_6^l, . When (X, Y, Z) is drawn from the set of random triples, d_i and z_i are random correlated with other servers values, as in P_6^{l+1}.

Let Δ' be the running time for D, and note that $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$. The advantage of D is

$$\begin{aligned} Adv_{G_q}^{DDH}(D) &= Pr[D \text{ outputs } 1|\text{DH triple}] - Pr[D \text{ outputs } 1|\text{random triple}] \\ &\geq Pr[Succ_{P_6^l}^{dake}(\mathcal{A})] - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &- Pr[Succ_{P_6^{l+1}}^{dake}(\mathcal{A})] - \frac{1}{q} - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &= \epsilon - \frac{1}{q} - \frac{6nn_{in}(n_{ro}+1)}{q}. \end{aligned}$$

The claim follows from the fact that $Adv_{G_q}^{DDH}(D) \leq Adv_{G_q}^{DDH}(\Delta')$. □

Protocol P_8. Let P_8 be a protocol that is identical to P_7 except that in a query to a client C, $d_{\tilde{p}} \leftarrow \mathbb{Z}_q$.

Claim 4.10. *For any adversary $\mathcal{A}$ running in time Δ, there is a $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$ such that*

$$Adv_{P_7}^{dake}(\mathcal{A}) \leq Adv_{P_8}^{dake}(\mathcal{A}) + (n_{in} + n_{ex}) \cdot \left(2Adv_{G_q}^{DDH}(\Delta') + \frac{O(nn_{in}n_{ro})}{q} \right).$$

Proof. We use a hybrid argument over $n_{in} + n_{ex}$ sessions. For each $l \in [0, n_{in} + n_{ex}]$, we define an intermadiate protocol which follows P_7 on the first l sessions, and on the remaining sessions it follows P_8. Clearly, $P_7^0 \equiv P_7$ and $P_7^{n_{in}+n_{ex}} \equiv P_8$.

Let's consider sessions l and $l+1$. Assume $Adv_{P_7^l}^{dake} = Adv_{P_7^{l+1}}^{dake} + 2\epsilon$. By Fact 2.1, it follows that $Pr(Succ_{P_7^l}^{dake}(\mathcal{A})) = Pr(Succ_{P_7^{l+1}}^{dake}(\mathcal{A})) + \epsilon$.

We construct an algorithm D that tries to distinguish between DH and random triples. Given triple (X, Y, Z), D simulates P_8^{l+1} with the following changes:

1. In the initialization phase, $(\hat{g}, \bar{g}) \leftarrow (g^{r_0}, g^{r_1})$, $y \leftarrow Y$, and $\hat{y} = (\hat{g})^{\hat{x}}$.
2. In step **C2** in a query to a client C, $c_{\tilde{p}} \leftarrow X$, $d_{\tilde{p}} \leftarrow Z \cdot h^p$, $\hat{c}_{\tilde{p}} \leftarrow X^{r_0}$, $\hat{d}_{\tilde{p}} \leftarrow (\hat{c}_{\tilde{p}})^{\hat{x}} (\hat{h})^p$, and $e_j \leftarrow (\bar{a}_j)^{1/r_0}$.

When (X, Y, Z) is drawn from the set of DH triples, the simulation is statistically indistinguishable from P_7^l, the statistical difference coming from probability that Fraud occurs. When (X, Y, Z) is a random triple, the simulation is statistically indistinguishable from P_7^{l+1}, the statistical difference coming from the probability that Fraud occurs, and $\frac{1}{q}$ the probability that (X, Y, Z) is actually a DH triple.

Let Δ' be the running time for D, and note that $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2 n_{ex})\Delta_{exp})$. The

advantage of D is

$$\begin{aligned} Adv^{DDH}_{G_q}(D) &= Pr[D \text{ outputs } 1|\text{DH triple}] - Pr[D \text{ outputs } 1|\text{random triple}] \\ &\geq Pr[Succ^{dake}_{P_7^l}(\mathcal{A})] - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &- Pr[Succ^{dake}_{P_7^{l+1}}(\mathcal{A})] - \frac{1}{q} - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &= \epsilon - \frac{1}{q} - \frac{6nn_{in}(n_{ro}+1)}{q}. \end{aligned}$$

The claim follows from the fact that $Adv^{DDH}_{G_q}(D) \leq Adv^{DDH}_{G_q}(\Delta')$. □

Protocol P_9. Let P_9 be a protocol that is identical to P_8 except that in the initialization phase, $(c_p, d_p) \leftarrow (\mathbb{Z}_q)^2$.

Claim 4.11. *For any adversary $\mathcal{A}$ running in time Δ, there is a $\Delta' = O(\Delta+(n_{ro}+nn_{in}+n^2n_{ex})\Delta_{exp})$ such that*

$$Adv^{dake}_{P_8}(\mathcal{A}) \leq Adv^{dake}_{P_9}(\mathcal{A}) + 2Adv^{DDH}_{G_q}(\Delta') + \frac{O(nn_{in}n_{ro})}{q}.$$

Proof. We construct an algorithm D that tries to distinguish between DH and random triples. Given triple (X, Y, Z), D simulates P_8 with the following changes in the initialization procedure: $y \leftarrow X$, and $(c_p, d_p) \leftarrow (Y, Z \cdot h^p)$.

When (X, Y, Z) is drawn from the set of DH triples, the simulation is statistically indistinguishable from P_8, the statistical difference coming from the probability that Fraud occurs. When (X, Y, Z) is a random triple, the simulation is statistically indistinguishable from P_9, the statistical difference coming from the probability that Fraud occurs, and $\frac{1}{q}$ the probability that (X, Y, Z) is actually a DH triple.

Let Δ' be the running time for D, and note that $\Delta' = O(\Delta + (n_{ro} + nn_{in} + n^2n_{ex})\Delta_{exp})$. The advantage of D is

$$\begin{aligned} Adv^{DDH}_{G_q}(D) &= Pr[D \text{ outputs } 1|\text{DH triple}] - Pr[D \text{ outputs } 1|\text{random triple}] \\ &\geq Pr[Succ^{dake}_{P_8}(\mathcal{A})] - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &- Pr[Succ^{dake}_{P_9}(\mathcal{A})] - \frac{1}{q} - \frac{3nn_{in}(n_{ro}+1)}{q} \\ &= \epsilon - \frac{1}{q} - \frac{6nn_{in}(n_{ro}+1)}{q}. \end{aligned}$$

The claim follows from the fact that $Adv^{DDH}_{G_q}(D) \leq Adv^{DDH}_{G_q}(\Delta')$. □

In P_9, authentication from an honest client will be accepted and no information about the sessions keys is leaked. Adversaries' attempts will be responded with random values (with probability $\frac{1}{q}$ that these values hit a correct value that matches a correct password), unless it guesses the password correctly. Let E be the event that Fraud occurs, $\mathcal{A}$ succeeds in a password guess (as defined in P_6),

or random values (as defined in P_7) hit correct ones.

$$\begin{aligned} Pr[Succ^{dake}_{P_9}(\mathcal{A})] &\leq Pr[Succ^{dake}_{P_9}(\mathcal{A})|E] \cdot Pr[E] \\ &+ Pr[Succ^{dake}_{P_9}(\mathcal{A})|\neg E] \cdot (1 - Pr[E]) \\ &\leq Pr[E] + \frac{1}{2}(1 - Pr[E]) \\ &= \frac{1}{2} + \frac{Pr[E]}{2} \leq \frac{1}{2} + \frac{n_{in}}{2N} + \frac{n_{in}}{2q} + \frac{3nn_{in}(n_{ro}+1)}{2q}, \end{aligned}$$

which implies $Adv^{dake}_{P_9}(\mathcal{A}) \leq \frac{n_{in}}{N} + \frac{n_{in}}{q} + \frac{3nn_{in}(n_{ro}+1)}{q}$.

The theorem follows from this fact, along with claims 4.2 through 4.11. □

4.2 Forward Secrecy

Unlike the previous T-PAKE protocols [8, 32], our protocol provides *immediate* forward secrecy; i.e., a protocol instance becomes forward secret once the random values used by the honest participants are erased.

Intuitively, immediate forward secrecy comes from the fact that a session key is generated by the DH function involving random elements $(g^{k_i}, g^{\tilde{x}})$ coming both from the server S_i and the client C. To formally prove forward secrecy, we introduce additional `Corrupt` queries to capture a corruption model of the adversary, as it is done in [30]. Three corruption scenarios are possible for T-PAKE:

i) The client password is leaked to the adversary.

ii) The private key share x_i of uncompromised server is leaked to the adversary. It allows to reconstruct x (recall that the adversary is allowed to compromise up to t servers in the base model).

iii) The password verification data (c_p, d_p) at the server S_i is replaced with a value of the adversary's choice.

To capture the scenarios above, we introduce the following oracles: $\texttt{Corrupt}_1(C)$ returns a password p_C for $C \in Clients$ to the adversary; $\texttt{Corrupt}_2(S_i)$ returns x_i belonging to the server S_i; and $\texttt{Corrupt}_3(S_i, C, (c'_p, d'_p))$ replaces the stored (c_p, d_p) on the server S_i with (c'_p, d'_p).

In cases when the adversary uses the `Corrupt` queries to ensure that the partnered client and server instances are associated with the same password verification data, we update the *freshness* and *partnering* definitions.

An additional requirement is added to a notion of *partnering* motivated by $\texttt{Corrupt}_3(S_i, C, (c'_p, d'_p))$ query: (c_p, d_p) verification data stored at server S_i should semantically encode the correct password p of a client C.

A client instance/server pair (Π^C_i, S_j) is said to be *fresh* if: (1) S_j is not compromised; (2) there has been no $\texttt{Reveal}(C, i, S_j)$ query; (3) if $\Pi^{S_j}_l$ and Π^C_i are partnered, there has been no $\texttt{Reveal}(S_j, l)$ query; (4) there has been no $\texttt{Corrupt}_1(C)$ before $\texttt{Send}(C, *)$ query; (5) there has been no $\texttt{Corrupt}_2(S_i)$ or $\texttt{Corrupt}_3(S_i, C, *)$ before $\texttt{Send}(C, *)$ query. A server instance $\Pi^{S_j}_i$ is said to be *fresh* if: (1) S_j is not compromised; (2) there has been no $\texttt{Reveal}(S_j, i)$ query; (3) if Π^C_l and $\Pi^{S_j}_i$ are partnered, there has been no $\texttt{Reveal}(C, l, S_j)$ query or ; (4) if Π^C_l and $\Pi^{S_j}_i$ are partnered, there

has been no $\texttt{Corrupt}_1(C)$ before $\texttt{Send}(S_j, C_l, *)$ query; (5) there has been no $\texttt{Corrupt}_2(S_j)$; and (6) if Π_l^C and $\Pi_i^{S_j}$ are partnered, there has been no $\texttt{Corrupt}_3(S_j, C_l, *)$ before $\texttt{Send}(S_j, C_l, *)$ query. Conditions 4-6 reflect the fact that if the adversary learns client's password, then she can trivially impersonate the client.

Advantage of the adversary is defined exactly as before.

Definition 4.12. *Protocol P is a forward-secure T-PAKE if, for all PPT adversaries $\mathcal{A}$ that make at most n_{in} online attacks and for all dictionary sizes N, there is a negligible function ϵ such that*

$$Adv_P^{dake}(\mathcal{A}) \leq \frac{n_{in}}{N} + \epsilon(\kappa).$$

Theorem 4.13. *T-PAKE$_{\mathsf{DKG}}$ is a forward-secure T-PAKE.*

Proof. The proof of forward secrecy is similar by structure and the most of details to the proof of Theorem 4.1. We will highlight the differences. As in the previous proof, we start with the original protocol P_0 and reduce it to P_9, where (informally) the adversary succeeds only if a guessed password is correct.

We say that instance Π_U^i is associated with U' if either $U = U'$ or $pid_U^i = U'$. Associated instances may (but not necessarily) become partnered. We say a query `Send` in step **C2** to a client C is *corrupted* if the adversary had previously queried $\texttt{Corrupt}_1(C)$. We say a query `Send` in step **S1** or in step **S2** to an uncompromised server S_i is *corrupted* if the adversary had previously queried $\texttt{Corrupt}_2(S_i)$ or $\texttt{Corrupt}_3(S_i, C, *)$, where C is associated with S_i instance. A query `Send` in step **S3** to an uncompromised server S_i is *corrupted* if the corresponding query `Send` in step **S2** to that instance is corrupted. Furthermore, if the adversary has corrupted an uncompromised server S_i using $\texttt{Corrupt}_2(S_i)$, all remaining uncompromised servers belonging to the corresponding protocol instance are assumed to be corrupted too. Note that for any corrupted query `Send` the adversary already "knows" the password used by the instance Π_U^i at the time the query is made, therefore the instance Π_U^i and the remaining instances associated with a client C are no longer fresh.

Let $\epsilon(\kappa)$ denotes a negligible function in κ.

Protocol P_1. We define protocol P_1 in the same way as in the previous proof.

Claim 4.14. *For any adversary $\mathcal{A}$, $Adv_{P_0}^{dake} \leq Adv_{P_1}^{dake} + \epsilon(\kappa)$.*

Proof. The proof proceeds exactly as the proof of Claim 4.2. □

Protocol P_2. We define protocol P_2 in the same way as in the previous proof.

Claim 4.15. *For any adversary $\mathcal{A}$, $Adv_{P_1}^{dake} \leq Adv_{P_2}^{dake} + \epsilon(\kappa)$.*

Proof. Exactly as the proof of Claim 4.3. □

Simulation Soundness. We define Fraud event in the same way as in the previous proof.

Claim 4.16. *For any adversary $\mathcal{A}$ running against P_2, $Pr(\mathsf{Fraud}) = \epsilon_{SS}$ is negligible in κ.*

Proof. Exactly as the proof of Claim 4.4. □

Protocol P_3. We define protocol P_3 in the same way as in the previous proof.

Claim 4.17. *For any adversary $\mathcal{A}$ running in polynomial time, $Adv^{dake}_{P_2} \leq Adv^{dake}_{P_3} + \epsilon(\kappa)$.*

Proof. The proof proceeds similar to the proof of Claim 4.5 with the changes described below. We construct an algorithm D that tries to distinguish between valid DH triples and random triples by running $\mathcal{A}$ on a simulation of the protocol. Given triple (X, Y, Z), D simulates P_2 for $\mathcal{A}$ with the changes described in the proof of Claim 4.5, except for step 6.

In step **S3** in a query to an uncompromised server S_i, if $\tilde{y}$ was generated in a query to a client C and $\bar{z} = 1$, then $SK \leftarrow SK_i$ for the SK_i value generated by that query. The adversary could change the password verification data stored at the server S_i using query $\texttt{Corrupt}_3(S_i, C, (c'_p, d'_p))$. If so, $\bar{z}$ evaluates to 1 only if $\frac{d_p}{(c_p)^x} = \frac{d'_p}{(c'_p)^x}$, otherwise the protocol aborts. □

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that in step **C2** in an non-corrupted query to a client C, $\hat{d}_{\tilde{p}} \leftarrow \mathbb{Z}_q$.

Claim 4.18. *For any adversary $\mathcal{A}$ running in polynomial time, $Adv^{dake}_{P_3} \leq Adv^{dake}_{P_4} + \epsilon(\kappa)$.*

Proof. The proof proceeds exactly as the proof of Claim 4.6. □

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except for the following:

1. In the initialization phase, $\hat{x} \leftarrow \mathbb{Z}_q$ is generated, $\hat{y} \leftarrow (\hat{g})^{\hat{x}}$.
2. In step **C2** in an non-corrupted query to a client C, $d_{\tilde{p}} \leftarrow y^{r_{\tilde{p}}} h^p$, $\hat{d}_{\tilde{p}} \leftarrow (\hat{y})^{r_{\tilde{p}}} (\hat{h})^p$.
3. In step **S2** in an non-corrupted query to an uncompromised server S_i that uses a $d_{\tilde{p}}$ value produced in a client query (i.e., $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}}) = (\hat{h})^p$),

$$z_i \leftarrow \left(\prod_{j \in J} \left(\frac{d_{\beta_j}}{c_\beta{}^{x_j}} \right)^{\lambda_{j,J_i}} \right)^{-1/\lambda_{i,J_i}}.$$

Note that this computation does not rely on the secret keys x_i or k_i of uncompromised servers. Non-corrupted query ensures that the adversary didn't query $\texttt{Corrupt}_3(S_i, C, *)$, and password verification data (c_p, d_p) corresponds to the correct password p of the client C.

Claim 4.19. *For any adversary $\mathcal{A}$, $Adv^{dake}_{P_4} \leq Adv^{dake}_{P_5} + \epsilon(\kappa)$.*

Proof. Exactly the same as the proof of Claim 4.7. □

Protocol P_6. Let P_6 be a protocol that is identical to P_5 except that in an non-corrupted query to an uncompromised server S_i for a client C and using parameters $par^* = (\tau_i, e_i, c_\beta, c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}})$ where par^* was never used in an non-corrupted query to a client C, if $(\hat{d}_{\tilde{p}}/(\hat{c}_{\tilde{p}})^{\hat{x}}) = (\hat{h})^p$, then P_6 stops and we say that $\mathcal{A}$ succeeds.

Claim 4.20. *For any adversary $\mathcal{A}$, $Adv^{dake}_{P_5} \leq Adv^{dake}_{P_6}$.*

Proof. Exactly the same as the proof of Claim 4.8. □

Protocol P_7. Let P_7 be a protocol that is identical to P_6 except for the following:

1. In step **S2** in an non-corrupted query to an uncompromised server S_i, $d_{\beta_i} \leftarrow \mathbb{Z}_q$.
2. In step **S2** in an non-corrupted query to an uncompromised server S_i, that uses values $(c_{\tilde{p}}, d_{\tilde{p}})$ not used in any query to a client C, and that uses a τ' value, compute z_j as follows. If there is a $(\bar{z}^*, \tau')$ pair recorded for this τ', use that $\bar{z}^*$, else choose $\bar{z}^* \leftarrow_R \mathbb{Z}_q$ and record $(\bar{z}^*, \tau')$. Then $z_i \leftarrow \left(\bar{z}^* \cdot \prod_{j \in J} \left(\frac{{c_\beta}^{x_j}}{d_{\beta_j}} \right)^{\lambda_{j,J_i}} \right)^{1/\lambda_{i,J_i}}$.

Claim 4.21. *For any adversary $\mathcal{A}$ running in polynomial time, $Adv^{dake}_{P_6} \leq Adv^{dake}_{P_7} + \epsilon(\kappa)$.*

Proof. The proof is similar to the proof of Claim 4.9. In the same way we construct an algorithm D, but steps 3-5 are applied to non-corrupted queries to an uncompromised server S_i. □

Protocol P_8. Let P_8 be a protocol that is identical to P_7 except that in an uncorrupted query to a client C, $d_{\tilde{p}} \leftarrow \mathbb{Z}_q$.

Claim 4.22. *For any adversary $\mathcal{A}$ running in polynomial time, $Adv^{dake}_{P_7} \leq Adv^{dake}_{P_8} + \epsilon(\kappa)$.*

Proof. The proof is similar to the proof of Claim 4.10. In the same way we construct an algorithm D, but step 2 is applied to a non-corrupted query to a client C. □

Protocol P_9. We define protocol P_9 in the same way as in the previous proof.

Claim 4.23. *For any adversary $\mathcal{A}$ running in polynomial time, $Adv^{dake}_{P_8} \leq Adv^{dake}_{P_9} + \epsilon(\kappa)$.*

Proof. Exactly the same as the proof of Claim 4.11. □

In P_9, authentication from an honest client will be accepted and no information about the session keys is leaked. If the adversary queries $\texttt{Corrupt}_3(S_i, C, (c'_p, d'_p))$ for some uncompromised server S_i where (c'_p, d'_p) corresponds to a wrong password of the client, the protocol instance aborts and the session keys are not generated by uncompromised servers. Adversaries' attempts will be responded with random values, unless she guesses a password correctly. Let E be the event that Fraud occurs, $\mathcal{A}$ succeeds in a password guess (as defined in P_6), or random values (as defines in P_7) hit a correct ones.

$$\begin{aligned} Pr[Succ^{dake}_{P_9}(\mathcal{A})] &\leq Pr[Succ^{dake}_{P_9}(\mathcal{A})|E] \cdot Pr[E] \\ &\quad + Pr[Succ^{dake}_{P_9}(\mathcal{A})|\neg E] \cdot (1 - Pr[E]) \\ &\leq Pr[E] + \frac{1}{2}(1 - Pr[E]) \\ &= \frac{1}{2} + \frac{Pr[E]}{2} \leq \frac{1}{2} + \frac{n_{in}}{2N} + \epsilon(\kappa) \end{aligned}$$

which implies $Adv^{dake}_{P_9}(\mathcal{A}) \leq \frac{n_{in}}{N} + \epsilon(\kappa)$.

The theorem follows from this fact, along with claims 4.14 through 4.23.

□

	Computation Time		Response Time
Setting	Server	Client	Client
$n = 3,\ t = 1$	44 ± 3	35 ± 1	72 ± 3
$n = 5,\ t = 2$	58 ± 5	56 ± 1	99 ± 4

Table 2: The average server and the client computation time (excluding the DKG computation time), and the average client response time for the T-PAKE$_{\mathsf{DKG}}$ protocol (in msec)

	Batch size			
Setting	1	10	10	1000
$n = 3, t = 1$	8 ± 3	62 ± 7	620 ± 20	6080 ± 190
$n = 5, t = 2$	28 ± 6	130 ± 13	1100 ± 40	10900 ± 300

Table 3: Computations time (per server) for the DKG protocol [24] various batch sizes (in msec)

5 Implementation and Performance Analysis

We have implemented our T-PAKE$_{\mathsf{DKG}}$ protocol from Section 3 as well as the DKG protocol [24] (which is required in the batched and offline fashion) and performed micro-benchmarks to verify its performance in the LAN setting. In this section, we discuss our implementation and experiments, and analyze the performance of T-PAKE$_{\mathsf{DKG}}$.

5.1 Implementation and Experiments

Our T-PAKE$_{\mathsf{DKG}}$ implementation is a single-threaded C++ program. It uses the GMP library [1] for all large finite-field computations, and the Relic toolkit [6] for elliptic curve cryptography (ECC) with 128-bit security. It employes the Boost [3] library for performing the network-level communication between servers and clients, and the OpenSSL [4] library for secure communication among the servers required for the DKG protocol. An anonymized version of our implementation is available online [2].

For our experiments, we used six 3.30 GHz (Intel i3) Linux machines with 8 GB RAM connected using a 1 Gbps LAN. We run experiments for the 3-server (i.e., $n = 3$ and $t = 1$), and 5-server (i.e., $n = 5$ and $t = 2$) settings.

In order to determine an average performance, we ran the experiments at least ten times for each parameter set. Our experiments are terminating and conducted via the method of independent replications, where a single replication consists of individual observations corresponding to the time required for every participating T-PAKE server and the client.

5.2 Performance Analysis

The average computation time for a T-PAKE$_{\mathsf{DKG}}$ server instance and a T-PAKE$_{\mathsf{DKG}}$ client instance, and the average response time for a client are available in Table 2. For the three-server setting ($n = 3, t = 1$), computation costs for the server instance and the client instance are respectively 44 ± 3 msec and 35 ± 1 msec, while the average delay a client experiences due to the T-PAKE$_{\mathsf{DKG}}$ execution is 72 ± 3 msec. The client delay is smaller than the combined computation costs for the server and client instances as the server complete their authentication only the client send her final

message. For the five-server setting ($n = 5, t = 2$), computation costs for the server and client instances are respectively 58 ± 5 msec and 56 ± 1 msec, while the average delay for the client is 99 ± 4 msec.

We find that these computations costs (or overheads) for the T-PAKE$_{\mathsf{DKG}}$ protocol can easily be accommodated in the existing password authentication infrastructure. Therefore, we propose the T-PAKE$_{\mathsf{DKG}}$ protocol as a practical password authentication system to tolerate the server compromises and the subsequent offline dictionary attacks.

Notice that the above time measures do not include the DKG instances; we analyze the performance of the DKG instances separately as we expect them to be executed in a batched and off-line fashion. In Table 3, we present the average per server computation time measures for the DKG protocol [24] for the batch sizes 1, 10, 100, and 1000 and for two parameter settings ($n = 3, t = 1$) and ($n = 5, t = 2$). From the table, we expect every DKG instance to take approximately 6 msec for the ($n = 3, t = 1$) setting, and approximately 10 msec for the ($n = 5, t = 2$) setting. Therefore, a DKG instance requires less than 1/5 of time required for a T-PAKE$_{\mathsf{DKG}}$ instance.

This performance of the DKG implementation can be further improved using the batched verification techniques [9] as well as the multi-exponentiations techniques (e.g, using Shamir's trick [33, Algo. 14.88]). In the ECC setting, where group inverses come for free, the number of exponentiations can be further reduced using Avanzi's algorithm [7] based on a sliding windows method. Moreover, performance for the T-PAKE$_{\mathsf{DKG}}$ protocol as well as the DKG protocol can be further improved using the multi-threaded implementation. In the future, we will incorporate these techniques in our implementation.

6 Extensions

In this section, we propose three enhancements for our T-PAKE$_{\mathsf{DKG}}$ protocol. We first improve the three-round T-PAKE$_{\mathsf{DKG}}$ protocol to a two-round T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$, which instead achieves implicit authentication. We then extend our T-PAKE$_{\mathsf{DKG}}$ and T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ protocols to the asynchronous communication setting. Finally, we propose an enhancement to our protocol setup, using which the public information size can be reduced linearly in n.

6.1 The Two-round T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ Protocol

T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ (Figure 3) is a modification of T-PAKE$_{\mathsf{DKG}}$ and provides only *implicit* authentication. The protocol has only two message flows between a client-server pair. The client sends his fresh public key $\tilde{y}$ and a commitment to a password $(c_{\tilde{p}}, d_{\tilde{p}})$ to all the servers. The servers randomize, partially decrypt the message to z_j and verify if the password was correct or not. Upon a successful verification, the servers send their nonces (a_j, b_j, e_j) to the client such that he could derive the session keys. All the messages are supplied with appropriate NIZK proofs. Note that we include $\tilde{y}$ into NIZK proofs to avoid replay attacks. An adversary is allowed to corrupt up to t servers, but she does not learn session keys established between the client and honest servers. The following proof systems are used in the protocol:

$$\mathcal{L}_C = \{(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, \tilde{y}, f_{\tilde{p}}) \in (G_q)^6 \mid \exists (r_{\tilde{p}}, \tilde{p}) \in (\mathbb{Z}_q)^2 \; s.t.$$
$$(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, f_{\tilde{p}}) = (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}}, (\tilde{y})^{r_{\tilde{p}}})\}$$
$$\mathcal{L}^i_{S1} = \{(a, b, c_{\tilde{p}}, e, \bar{a}) \in (G_q)^5 | \; \exists k \in \mathbb{Z}_q \; s.t. \; (a, b, e, \bar{a}) = (g^k, (c_p)^k, (c_{\tilde{p}})^k, (\bar{g})^k)\}$$
$$\mathcal{L}^i_{S2} = \{(z, a, \delta, P) \in (G_q)^4 \mid \exists (k, x) \in (\mathbb{Z}_q)^2 \; s.t. \; (y_i, a, z) = (g^x, g^k, \delta^k P^{-x})\}$$

Informally, the security of T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ is implied from the security of T-PAKE$_{\mathsf{DKG}}$ as the adversary learns the same public messages such as a_i, b_i, e_i here as in T-PAKE$_{\mathsf{DKG}}$. Although generation of e_i and c_β is shifted from the client side to the servers in T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$, the client can verify the well-formedness of these values. The rest of the proof remains almost the same, and therefore is skipped. T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ provides only *implicit* authentication, and using standard techniques (e.g., [10]) an additional round of communication can be added afterwards to perform explicit authentication.

6.2 The Asynchronous Communication Setting

The servers in our T-PAKE$_{\mathsf{DKG}}$ and T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$ protocols employ a broadcast channel to communicate with each other in a reliable and authenticated manner. Consequently, we consider our communication model to be (bounded) synchronous. It is, however, possible to make our protocols work in the completely asynchronous communication setting that does not put any bounds on message transfer delays [16].

This involves replacing the employed broadcast channel by an asynchronous reliable broadcast protocol such as [15], and the employed synchronous DKG protocol [24] by an asynchronous DKG protocol such as [16, 28]. Due to the inherent difficulty of distinguishing between a slow server and a crashed server in the asynchronous communication setting the above asynchronous primitives (and consequently the asynchronous versions of our protocols) ask for $n \geq 3t + 1$ servers instead of the $n \geq 2t + 1$ servers required otherwise. The protocols and their proofs remain exactly the same except that, to ensure the termination, we require $|I_{S_i}| \geq 2t + 1$ such that $|I'_{S_i}|$ will be $\geq t + 1$.

6.3 Reducing the Public Key Size

The current public information that the client needs to obtain and maintain is of the size $O(n\kappa)$ as it contains a public key y_i for every server S_i. It is possible to reduce this public information size to $O(\kappa)$ using a constant-size polynomial commitments protocol [29]. In this case, the public information contains only a (single element) polynomial commitment as the public key for the set of n servers. Every server S_i includes its public key $y_i = g^{x_i}$ and its witness in its first message to the client along with an NIZK proof of knowledge of exponent x_i. The resulting protocol, however, requires the t-strong Diffie–Hellman (t-SDH) assumption [13].

7 Conclusion

T-PAKE protocols provide a practical solution for the offline dictionary attack problem with password authentication by mitigating the risk in a threshold manner. However, we observed

Setup (on public parameters g, q, n, t):
$x \leftarrow_R \mathbb{Z}_q$, $y \leftarrow g^x$, $\{x_i\}_{i=1}^n \xleftarrow{(t+1,n)} SS(x)$, $(h, \hat{g}, \hat{h}, \hat{y}, \bar{g}) \leftarrow (G_q)^5$, $\{y_i \leftarrow g^{x_i}\}_{i=1}^n$.
Client initialization (on public parameters g, y, h and password p): $r_p \leftarrow_R \mathbb{Z}_q$,
$(c_p, d_p) \leftarrow (g^{r_p}, y^{r_p} h^p)$.
Public information: $g, h, y, \{y_j\}_{j=1}^n, \hat{g}, \hat{h}, \hat{y}, \bar{g}, (c_p, d_p)$.
Private keys: $\{x_i\}_{i=1}^n$. Server S_i has a private key x_i.
Batch offline (before Client Login): $(K := g^k, \{k_i\}_{i=1}^n) \xleftarrow{(t+1,n)} DKG$. Server S_i receives k_i.
Client Login

C1 (Client C) Let $I = \langle I_1, \ldots, I_n \rangle$. Pick $(\tilde{x}, r_{\tilde{p}}) \leftarrow_R (\mathbb{Z}_q)^2$.
Compute: $\tilde{y} \leftarrow g^{\tilde{x}}$, $(c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, f_{\tilde{p}}) \leftarrow (g^{r_{\tilde{p}}}, y^{r_{\tilde{p}}} h^{\tilde{p}}, (\hat{g})^{r_{\tilde{p}}}, (\hat{y})^{r_{\tilde{p}}} (\hat{h})^{\tilde{p}}, (\tilde{y})^{r_{\tilde{p}}})$,
$\pi_1 \leftarrow \mathsf{Prove}_C((c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, f_{\tilde{p}}), (r_{\tilde{p}}, \tilde{p}))$.
Send $(C, I, \tilde{y}, (c_{\tilde{p}}, d_{\tilde{p}}), (\hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}), f_{\tilde{p}}, \pi_1)$ to the server S_i for all $I_i \in I$.

S1 (Server S_i) If $\neg\mathsf{Verify}_C((c_{\tilde{p}}, d_{\tilde{p}}, \hat{c}_{\tilde{p}}, \hat{d}_{\tilde{p}}, f_{\tilde{p}}), (\pi_1))$ then Abort.
Compute: $(a_i, b_i, e_i, \bar{a}_i) \leftarrow (g^{k_i}, (c_p)^{k_i}, (c_{\tilde{p}})^{k_i}, \bar{g}^{k_i})$,
$\pi_{2i} \leftarrow \mathsf{Prove}_{S1}((a_i, b_i, e_i, \bar{a}_i), k_i)$.
Broadcast: $(a_i, b_i, e_i, \bar{a}_i, \pi_{2i})$.

S2 (Server S_i) $I_{S_i} \leftarrow \{i \mid \mathsf{Verify}_{S1}((a_i, b_i, e_i, \bar{a}_i), (\pi_{1i}))\}_{i \in I}$.
Compute: $c_\beta \leftarrow \prod_{j \in I_{S_i}} (b_i / e_i)^{\lambda_{j, I_{S_i}}}$, $w_i \leftarrow (c_\beta)^{x_i}$, $d_{\beta_i} \leftarrow (d_p / d_{\tilde{p}})^{k_i}$,
$z_i \leftarrow d_{\beta_i} / w_i$, $\pi_{3i} \leftarrow \mathsf{Prove}_{S2}((z_i, a_i, d_p / d_{\tilde{p}}, c_\beta), (k_i, x_i))$.
Broadcast (z_i, π_{3i}).

S3 (Server S_i) $I'_{S_i} \leftarrow \{j \mid \mathsf{Verify}_{S2}((z_j), (\pi_{3j}))\}_{j \in I_{S_i}}$. If $|I'_{S_i}| < t + 1$ then Abort.
Compute: $\bar{z} \leftarrow \prod_{j \in I'_{S_i}} (z_j)^{\lambda_{j, I'_{S_i}}}$. If $\bar{z} \neq 1$ then Abort.
Let $\tau_i \leftarrow \langle \tilde{y}, a_i, K \rangle$.
Compute: $\tilde{y}_i \leftarrow \tilde{y}^{x_i}$, $\tilde{k}_i \leftarrow \tilde{y}^{k_i}$, $SK_i \leftarrow H_0(\tau_i, \tilde{y}_i, \tilde{k}_i)$.
Send $(a_i, b_i, e_i, \bar{a}_i, \pi_{2i})$ to the client C.

C2 (Client C) $I_C \leftarrow \{i \mid \mathsf{Verify}_{S1}((a_i, b_i, e_i, \bar{a}_i), (\pi_{2i}))\}_{i \in I}$. If $|I_C| < t + 1$ then Abort.
Compute: $K \leftarrow \prod_{i \in I_C} a_i^{\lambda_{i, I_C}}$, $\{\tau_i \leftarrow \langle \tilde{y}, a_i, K \rangle$, $\tilde{y}_i \leftarrow (y_i)^{\tilde{x}}$, $\tilde{k}_i \leftarrow (a_i)^{\tilde{x}}$,
$SK_i \leftarrow H_0(\tau_i, \tilde{y}_i, \tilde{k}_i)\}_{i \in I_C}$

Figure 3: Protocol: T-PAKE$^{\mathsf{I}}_{\mathsf{DKG}}$

that the existing T-PAKE constructions fail the basic liveness requirement of any fault-tolerant communication protocol: none of these protocols can tolerate any misbehavior by one or more participating servers without restarting the protocol and replacing the misbehaving servers. This not only presents a serious fault management challenge for the servers, but can also leave the clients frustrated.

We addressed the fault tolerance problem with T-PAKE (and in turn the associated PASS primitive) by adding a batched and offline phase of DKG. Our resulting protocols ensure the security and guaranteed termination properties for every protocol instance for $n \geq 2t + 1$. We have proved their security under the DDH assumption in the random oracles model. Both protocols

also improve upon the existing T-PAKE protocols in terms of communication and require at least two broadcasts less than other secure T-PAKE constructions. Using the asynchronous broadcast and DKG protocols, they can also work in the asynchronous setting for $n \geq 3t + 1$. Finally, we implemented our T-PAKE$_{\mathsf{DKG}}$ protocol, and verified its performance in the LAN environment for $(n = 3, t = 1)$ and $(n = 5, t = 2)$. We find that our protocol adds a small computation overhead of < 0.1 sec in both settings, and propose it for real-world password authentication system. In the near future, we plan to analyze its performance in the cloud environment.

In this work, we prove the protocol security only against a static attacker. It presents an interesting challenge to prove security against an adaptive attacker with an adaptively secure DKG protocol [19].

References

[1] GMP: The GNU Multiple Precision Arithmetic Library. `http://gmplib.org`.

[2] Our T-PAKE Implementation. `https://sites.google.com/site/tpakefaulttolerant/`.

[3] The Boost C+ Libraries. `http://www.boost.org`.

[4] The OpenSSL Project. `http://www.openssl.org`.

[5] M. Abe. Robust distributed multiplication without interaction. In *CRYPTO*, pages 130–147, 1999.

[6] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[7] R. M. Avanzi. The Complexity of Certain Multi-Exponentiation Techniques in Cryptography. *J. Cryptology*, 18(4):357–373, 2005.

[8] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *CCS'11*, pages 433–444, 2011.

[9] M. Bellare, J. A. Garay, and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *EUROCRYPT*, pages 236–250, 1998.

[10] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.

[11] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Security and Privacy*, pages 72–84, 1992.

[12] D. Boneh. The Decision Diffie-Hellman Problem. In *ANTS*, pages 48–63. 1998.

[13] D. Boneh and X. Boyen. Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *J. Cryptology*, 21(2):149–177, 2008.

[14] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *EUROCRYPT*, pages 156–171, 2000.

[15] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC'84*, pages 154–162, 1984.

[16] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In *CCS'02*, pages 88–97, 2002.

[17] J. Camenisch, A. Lysyanskaya, and G. Neven. Practical yet universally composable two-server password-authenticated secret sharing. In *CCS'12*, pages 525–536, 2012.

[18] J. Camenisch and M. Stadler. *Proof systems for general statements about discrete logarithms.* Citeseer, 1997.

[19] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO*, pages 98–115, 1999.

[20] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[21] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.

[22] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–438, 1987.

[23] W. Ford and B. S. K. Jr. Server-assisted generation of a strong secret from a password. In *WETICE*, pages 176–180, 2000.

[24] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[25] O. Goldreich and Y. Lindell. Session-key generation using human passwords only. *J. Cryptology*, 19(3):241–340, 2006.

[26] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Inf. Syst. Secur.*, 2(3):230–268, 1999.

[27] D. P. Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Comp. Comm. Review*, 26(5):5–26, 1996.

[28] A. Kate and I. Goldberg. Distributed Key Generation for the Internet. In *ICDCS*, pages 119–128, 2009.

[29] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT*, pages 177–194, 2010.

[30] J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *J. ACM*, 57(1), 2009.

[31] Linkedin password file compromise, 2012. `http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised`.

[32] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. *J. Cryptology*, 19(1):27–66, 2006.

[33] A. Menezes, P. C. V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1st edition, 1997.

[34] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[35] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[36] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[37] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.

[38] Yahoo password file compromise, 2012. `http://security.yahoo.com/information-regarding-recent-password-compromise-000000466.html`.

A The Online Attack on the PPSS-based T-PAKE

As suggested in Table 1, due to the already discussed inherent limitations of PPSS, a T-PAKE construction based on the PPSS protocol [8] is not secure against an online password guessing attack. An adversary can attack the protocol as follows:

i) Pick a password candidate $\tilde{p}$ from the set of possible passwords D.

ii) Trigger two instances of the protocol with sId_1 and sId_2 and $\tilde{p}_1 = \tilde{p}_2 = \tilde{p}$.

iii) Upon reconstruction of secrets $\hat{sk}_1$ and $\hat{sk}_2$, stop any further execution of the protocol instances.

iv) If $\hat{sk}_1 \neq \hat{sk}_2$ then the password is wrong, remove $\tilde{p}$ from D and go to step i; else the guessed password is correct $\tilde{p} = p$.

Importantly, in the above scenario, no server can distinguish between an honest client or the attacker.