Threshold Password-Authenticated Key Exchange

Philip MacKenzie*
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974 USA

Thomas Shrimpton†
Deptartment of Computer Science
Portland State University
Portland, OR 97207 USA
teshrim@cs.pdx.edu

Markus Jakobsson‡
School of Informatics
Indiana University at Bloomington
Bloomington, IN 47408 USA
www.markusjakobsson.com

January 7, 2005

Abstract

In most password-authenticated key exchange systems there is a single server storing password verification data. To provide some resilience against server compromise, this data typically takes the form of a one-way function of the password (and possibly a salt, or other public values), rather than the password itself. However, if the server is compromised, this password verification data can be used to perform an offline dictionary attack on the user's password. In this paper we propose an efficient password-authenticated key exchange system involving a set of servers with known public keys, in which a certain threshold of servers must participate in the authentication of a user, and in which the compromise of any fewer than that threshold of servers does not allow an attacker to perform an offline dictionary attack. We prove our system is secure in the random oracle model under the Decision Diffie-Hellman assumption against an attacker that may eavesdrop on, insert, delete, or modify messages between the user and servers, and that compromises fewer than that threshold of servers.

Key words: Password authentication, key exchange, threshold cryptosystems, dictionary attack.

1 Introduction

Many real-world systems today rely on password authentication to verify the identity of a user before allowing that user to perform certain functions, such as setting up a virtual private network or downloading secret information. There are many security concerns associated with password

*Current affiliation: DoCoMo USA Labs, philmac@docomolabs-usa.com
†Work done at Bell Laboratories, Lucent Technologies, and University of California at Davis
‡Work done while at Bell Laboratories, Lucent Technologies, and RSA Laboratories

authentication, due mainly to the fact that most users' passwords are drawn from a relatively small and easily generated dictionary. Thus if information sufficient to verify a password guess is leaked, the password may be found by performing an offline dictionary attack: one can run through a dictionary of possible passwords, testing each one against the leaked information in order to determine the correct password.

When password authentication is performed over a network, one must be especially careful not to allow any leakage of information to one listening in, or even actively attacking, the network. If one assumes the server's public key is known (or at least can be verified) by the user, then performing password authentication after setting up an anonymous secure channel to the server is generally sufficient to prevent leakage of information, as is done in SSH [37] or on the web using SSL [16]. Halevi and Krawczyk [28] give the first protocol of this type that is proven secure. The problem becomes more difficult if the server's public key cannot be verified by the user. Solutions to this problem have been coined *strong password authentication protocols*, and have the property that (informally) the probability of an active attacker (i.e., one that may eavesdrop on, insert, delete, or modify messages on a network) impersonating a user is only negligibly better than a simple on-line guessing attack, consisting of the attacker iteratively guessing passwords and running the authentication protocol. Strong password authentication protocols were proposed by Bellovin and Merritt [5, 6], Jablon [31] and Wu [38], among others. Recently, some protocols were proven secure in the random oracle and/or ideal cipher models[1] (Bellare *et al.* [1], Boyko *et al.* [10] and MacKenzie *et al.* [34]), in the public random string model (Katz *et al.* [33]), and in the standard model[2] (Goldreich and Lindell [26]). However, all of these protocols, even the ones in which the server's public key is known to the user, are vulnerable to server compromise in the sense that compromising the server would allow an attacker to obtain the password verification data on that server (typically some type of one-way function of the password and some public values). This could then be used to perform an offline dictionary attack on the password. To address this issue (without resorting to assumptions like tamper resistance), Ford and Kaliski [22] proposed to distribute the functionality of the server, forcing an attacker to compromise several servers in order to be able to obtain password verification data.[3] Their protocol assumes the servers have known public keys. Note that the main problem is not just to distribute the password verification data, but to distribute the functionality, i.e., to distribute the password verification data such that it can be used for authentication without ever reconstructing the data on any set of servers smaller than a chosen threshold.

While distributed cryptosystems have been studied extensively (and many proven secure) for other cryptographic operations, such as signatures (e.g., [9, 14, 25, 23]), to our knowledge Ford and Kaliski were the first ones to propose a distributed password-authenticated key exchange system. However, they give no proof of security for their system. Jablon [32] extends the system of Ford and Kaliski, most notably to not require the server's public key to be known to the user, but again

[1] In the random oracle model [2], a hash function is modeled as a black box containing an ideal random function. This is not a standard cryptographic assumption. In fact, it is possible for a scheme secure in the random oracle model to be insecure for any real instantiation of the hash function [11]. However, a proof of security in the random oracle model is generally thought to be strong evidence of the practical security of a scheme. (The ideal cipher model is similar to the random oracle model, except that it is a cipher that is modeled as a black box containing a keyed family of independent random permutations and their inverses.)

[2] The protocol for the standard model is only proven secure in the case of non-concurrent executions.

[3] As is well-known in the practice of distributed cryptography, for high security one must be careful to ensure that it is not easy for an attacker to compromise several servers with the same attack, which may be the case, for instance, if they are all running the same operating system.

does not give a proof of security.

Our contributions. In this paper we propose a completely different distributed password authenticated key exchange system and prove it secure in the random oracle model, assuming the hardness of the Decision Diffie-Hellman (DDH) problem [17] (see [8]). Like the system of Ford and Kaliski, we assume the servers have known public keys. However, while the systems of Ford and Kaliski and Jablon require all servers to perform authentication, our system is a k-out-of-n threshold system (for any $1 \leq k \leq n$), where k servers are required for authentication and the compromise of $k-1$ servers does not affect the security of the system. Also, this is the first distributed password-authenticated key exchange system proven secure under any standard cryptographic assumption in any model, including the random oracle model. To be specific, we assume the client may store public data, and our security is against an active attacker that may (statically) compromise any number of servers less than the specified threshold.

Informally, one can succinctly state our main result as a distributed password-authenticated key exchange protocol for which any efficient attacker can do no better than an on-line dictionary attack as long as the DDH problem is hard, and as long as no more than a threshold $k-1$ out of n servers are compromised.

Technically, we achieve our result by storing a semantically-secure encryption of a function of the password at the servers (instead of simply storing a one-way function of the password), and then leveraging off some known solutions for distributing secret decryption keys, such as Feldman verifiable secret sharing [20]. In other words, we transform the problem of distributing password authentication information to the problem of distributing cryptographic keys. However, once we make this transformation, verifying passwords without leaking information becomes much more difficult, requiring intricate manipulations of ElGamal encryptions [19] and careful use of efficient non-interactive zero-knowledge proofs [7]. In particular, we note that one cannot immediately obtain a solution through standard threshold cryptography techniques, since those techniques are not designed to prevent leakage of information when secrets may be chosen from a small set (e.g., when the secrets are passwords).

We note that a threshold password authentication system does not follow from techniques for general secure multi-party computation (e.g., [27]) since we are working in an asynchronous model, allow concurrent executions of protocols, and assume no authenticated channels. (Note in particular that the *goal* of the protocol is for the client to be authenticated.) The only work on general secure multi-party computation in an asynchronous model, and allowing concurrency, assumes authenticated channels [12].

Related work. Subsequent to our work, Di Raimondo and Gennaro [18] presented a distributed password-authenticated key exchange protocol based on the protocol of [33]. Here we highlight the differences from our protocol. Their protocol does not assume the servers have known public keys, whereas ours assumes the servers have known public keys Their protocol is in the public random string model, whereas ours is proven secure in the random oracle model. Their protocol requires a threshold k where $3k < n$, and requires all n servers to be active. Ours allows any threshold $k < n$, and allows only k servers to be active when no malicious behavior occurs.

2 Model

We extend the model of [1] (which builds on [3] and [4], and is also used by [33]). The model of [1] was designed for the problem of authenticated key exchange (ake) between two parties, a client and

a server. The goal was for them to engage in a protocol such that after the protocol was completed, they would each hold a session key that is known to nobody but the two of them. Our model is designed for the problem of *distributed authenticated key exchange* (dake) between a client and k servers. The goal is for them to engage in a protocol such that after the protocol is completed, the client would hold k session keys, one being shared with each server, such that the session key shared between the client and a given server is known to nobody but the two of them, even if up to $k-1$ other servers were to conspire together.

Note that this definition is in some sense optimized for the case when the servers do not misbehave. The client simply contacts any k servers and runs the protocol. If fewer than k servers do not perform the protocol honestly, then at least one uncompromised server will notice this, and the protocol will fail. This problem may be resolved in many ways, the simplest being the client iteratively trying different sets of k servers. Eventually, of course, the system must determine the compromised servers and reset the system, possibly using techniques from proactive security [30, 29]. These issues are beyond the scope of this paper. Here we focus on the basic protocol.

Remark 2.1 The way we have defined dake, the client ends up with k shared keys, while the goal of a standard authenticated key exchange is for the client to end up with a single key shared with a server it wishes to communicate with. There are alternative definitions that would more closely mimic this. However, we feel our definition is more general, since once the client can securely communicate with k servers, it can use this not only to enable secure communication with any other desired server, but to enable any desired cryptographic functionality. For instance, a secure dake protocol allows for secure downloadable credentials, by, e.g., having the servers store an encrypted credentials file with a decryption key stored using a threshold scheme among them, and then having each send a partial decryption of the credentials file to the client, encrypted with the session key it shares with the client. (To deal with compromised servers, one could require each server to also send a zero-knowledge proof that it performed its partial decryption correctly.) Note that the credentials are secure in a threshold sense: fewer than the given threshold of servers are unable to obtain the credentials. Once the client has securely downloaded its credentials (for instance, it could download its certified public key and the associated private key), it can use these credentials to set up secure communication with another server, or perhaps sign messages, or perform other cryptographic operations. Details of these applications are beyond the scope of this paper.

In the following, we will assume some familiarity with the model of [1].

Protocol participants. We have two types of protocol participants: clients and servers. Let $ID \stackrel{\text{def}}{=} \mathit{Clients} \cup \mathit{Servers}$ be a non-empty set of protocol participants, or *principals*.

We assume *Servers* consists of n servers, denoted $\{S_1, \ldots, S_n\}$, and that these servers are meant to cooperate in authenticating a client.[4] Each client $C \in \mathit{Clients}$ has a secret password π_C, and each server $S \in \mathit{Servers}$ has a vector $\pi_S = [\pi_S[C]]_{C \in \mathit{Clients}}$. Entry $\pi_S[C]$ is the *password record*. Let $\mathit{Password}_C$ be a (possibly small) set from which passwords for client C are selected. We will assume that $\pi_C \stackrel{R}{\leftarrow} \mathit{Password}_C$ (but our results easily extend to other password distributions). Clients and servers are modeled as probabilistic poly-time algorithms with an input tape and an output tape.

Execution of the protocol. A protocol P is an algorithm that determines how principals behave in response to inputs from their environment. In the real world, each principal is able to execute

[4] Our model could be extended to have multiple sets of servers, but for clarity of presentation we omit this extension.

P multiple times with different partners, and we model this by allowing an unlimited number of *instances* of each principal. Instance i of principal $U \in ID$ is denoted Π_i^U.

To describe the security of the protocol, we assume there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and thus provides the inputs to instances of principals. (Note in particular that we do not assume the communication between any parties is authenticated or private, even between servers.) We will further assume the network (i.e., $\mathcal{A}$) performs aggregation and broadcast functions.[5] In practice, on a point-to-point network, the protocol implementor would most likely have to implement these functionalities in some way, perhaps using a single intermediate (untrusted) node to aggregate and broadcast messages[6]. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. Queries written to this tape are responded to by principals according to P; the allowed queries are formally defined in [1] and summarized here (with slight modifications for multiple servers):

Send $(U,\ i,\ M)$**:** causes message M to be sent to instance Π_i^U. The instance computes what the protocol says to, state is updated, and the output message is given to $\mathcal{A}$. If this query causes Π_i^U to accept or terminate, this will also be shown to $\mathcal{A}$. To initiate a session between client C and a set of servers, the adversary should send a message containing a set I of k indices of servers in *Servers* to an unused instance of C.

Execute $(C, i, ((S_{j_1}, \ell_{j_1}), \ldots, (S_{j_k}, \ell_{j_k})))$**:** causes P to be executed to completion between Π_i^C (where $C \in$ *Clients*) and $\Pi_{\ell_{j_1}}^{S_{j_1}}, \ldots, \Pi_{\ell_{j_k}}^{S_{j_k}}$, and outputs the transcript of the execution. (This transcript includes all protocol messages, even those from server to server.) This query captures the intuition of a passive adversary who simply eavesdrops on the execution of P.

Reveal $(C,\ i,\ S_j)$**:** causes the output of the session key held by Π_i^C corresponding to server S_j, i.e., sk_{C,S_j}^i.

Reveal $(S_j,\ i)$**:** causes the output of the session key held by $\Pi_i^{S_j}$, i.e., $sk_{S_j}^i$.

Test $(C,\ i,\ S_j)$**:** causes Π_i^C to flip a bit b. If $b = 1$ the session key sk_{C,S_j}^i is output and if $b = 0$ a string drawn uniformly from the space of session keys is output. A Test query (of either type) may be asked at any time during the execution of P, but may only be asked once.

Test $(S_j,\ i)$**:** causes $\Pi_i^{S_j}$ to flip a bit b. If $b = 1$ the session key $sk_{S_j}^i$ is output; otherwise, a string is drawn uniformly from the space of session keys and output. As above, a Test query (of either type) may be asked at any time during the execution of P, but may only be asked once.

The Reveal queries are used to model an adversary who obtains information on session keys in some sessions, and the Test queries are a technical addition to the model that will allow us to determine if an adversary can distinguish a true session key from a random key.

We assume $\mathcal{A}$ may compromise up to $k-1$ servers, and that the choice of these servers is static. In particular, without loss of generality, we may assume the choice is made before initialization, and we may simply assume the adversary has access to the private keys of the compromised servers.

[5]This is more for notational convenience than anything else. In particular, we make no assumptions about synchronicity or any type of distributed consensus.

[6]Note that since $\mathcal{A}$ controls the network and can deny service at any time, we do not concern ourselves with any denial-of-service attacks that this single intermediate node may facilitate.

Partnering. A server instance that accepts holds a partner-id pid, session-id sid, and a session key sk. A client instance that accepts holds a partner-id pid consisting of a set of k server indices, a session-id sid, and a set of k session keys $(sk_{j_1}, \ldots, sk_{j_k})$. Let sid be the concatenation of all messages (or pre-specified compacted representations of the messages) sent and received by the client instance in its communication with the set of servers. (Note that this excludes messages that are sent only between servers, but not to the client. Also, as discussed above, we assume the network performs aggregation and broadcast functions so each server can see the messages communicated between the client and other servers, and thus can construct sid.) Then instances Π_i^C (with $C \in \mathit{Clients}$) holding $(pid, sid, (sk_{j_1}, \ldots, sk_{j_k}))$, where $pid = I$ for some set $I = \{j_1, \ldots, j_k\}$, and $\Pi_{\ell_j}^{S_j}$ (with $S_j \in \mathit{Servers}$) holding (pid', sid', sk) are said to be *partnered* if $j \in I$, $pid' = C$, $sid = sid'$, and $sk_j = sk$. This is basically the so-called "matching conversation" approach to defining partnering, as used in [3, 1].

Freshness. A client instance/server pair (Π_i^C, S_j) is *fresh* if (1) S_j is not compromised, (2) there has been no $\mathsf{Reveal}\,(C,\ i,\ S_j)$ query, and (3) if $\Pi_\ell^{S_j}$ is a partner to Π_i^C, there has been no $\mathsf{Reveal}\,(S_j,\ \ell)$ query. A server instance $\Pi_i^{S_j}$ is *fresh* if (1) S_j is not compromised, (2) there has been no $\mathsf{Reveal}\,(S_j,\ i)$ query, and (3) if Π_ℓ^C is the partner to $\Pi_i^{S_j}$, there has been no $\mathsf{Reveal}\,(C,\ \ell,\ S_j)$ query. Intuitively, the adversary should not be able to distinguish random keys from session keys held by fresh instances.

Advantage of the adversary. We now formally define the distributed authenticated key exchange (dake) advantage of the adversary against protocol P. Let $\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})$ be the event that (1) $\mathcal{A}$ 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 (2) $\mathcal{A}$ 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

$$\mathsf{Adv}_P^{\mathrm{dake}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} 2\Pr\left[\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})\right] - 1.$$

The following fact is easily verified.

Fact 2.2 $\Pr(\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})) = \Pr(\mathsf{Succ}_{P'}^{\mathrm{dake}}(\mathcal{A})) + \epsilon \quad \Longleftrightarrow \quad \mathsf{Adv}_P^{\mathrm{dake}}(\mathcal{A}) = \mathsf{Adv}_{P'}^{\mathrm{dake}}(\mathcal{A}) + 2\epsilon.$

3 Definitions

Let κ be the cryptographic security parameter. Let G_q denote a finite (cyclic) group of order q, where $|q| = \kappa$. Let g be a generator of G_q, and assume it is included in the description of G_q.

Notation. We use $(a, b) \times (c, d)$ to mean elementwise multiplication, i.e., (ac, bd). We use $(a, b)^r$ to mean elementwise exponentiation, i.e., (a^r, b^r). For a tuple V, the notation $V[j]$ means the jth element of V.

We denote by Ω the set of all functions H from $\{0,1\}^*$ to $\{0,1\}^\infty$. This set is provided with a probability measure by saying that a random H from Ω assigns to each $x \in \{0,1\}^*$ a sequence of bits each of which is selected uniformly at random. As shown in [2], this sequence of bits may be used to define the output of H in a specific set, and thus we will assume that we can specify that the output of a random oracle H be interpreted as a (random) element of G_q.[7] Access to any

[7] For instance, this can be easily defined when G_q is a q-order subgroup of $\mathbb{Z}_p^*$, where q and p are prime.

public random oracle $H \in \Omega$ is given to all algorithms; specifically, it is given to the protocol P and the adversary $\mathcal{A}$. Assume that secret session keys are drawn from $\{0,1\}^\kappa$.

A function $f : \mathbb{Z} \to [0,1]$ is negligible if for all $\alpha > 0$ there exists an $\kappa_\alpha > 0$ such that for all $\kappa > \kappa_\alpha$, $f(\kappa) < |\kappa|^{-\alpha}$. All functions we use in this paper will include a security parameter as input, either implicitly or explicitly, and we say that these functions are negligible if they are negligible in the security parameter. (They will be polynomial in all other parameters.)

4 Protocol

In this section we describe our protocol for threshold password-authenticated key exchange. In the next section we prove this protocol is secure under the DDH assumption [8, 17] in the random-oracle model [2].

4.1 Server Setup

Let there be n servers $\{S_i\}_{i\in\{1,2,\ldots,n\}}$. Let (x, y) be the servers' *global* key pair such that $y = g^x$. The servers share the global secret key x using a (k, n)-threshold Feldman secret sharing protocol [20]. Specifically, a polynomial $f(z) = \sum_{j=0}^{k-1} a_j z^j \bmod q$ is chosen with $a_0 \leftarrow x$ and random coefficients $a_j \stackrel{R}{\leftarrow} \mathbb{Z}_q$ for $j > 0$. Then each server S_i gets a secret share $x_i = f(i)$ and a corresponding public share $y_i = g^{x_i}$, $1 \le i \le n$. (In this paper we assume that a trusted dealer generates these shares, but it should be possible to have the servers generate them using a distributed protocol, as in Gennaro *et al.* [24].) In addition, each server S_i independently generates its own *local* key pair (x'_i, y'_i) such that $y'_i = g^{x'_i}$, $1 \le i \le n$. Each server S_i publishes its *local public key* y'_i along with its share of the global public key y_i. Note that we assume that the adversary does not participate in the system setup phase, so all keys are generated honestly. Let $H_0, H_1, H_2, H_3, H_4, H_5, H_6 \stackrel{R}{\leftarrow} \Omega$ be random oracles with domain and range defined by the context of their use. Let $h \leftarrow H_0(y)$ and $h' \leftarrow H_1(y)$ be generators for G_q.

Remark 4.1 We note that in the following protocol the servers are assumed to have stored the $2n+1$ public values y, $\{y'_i\}_{i=1}^n$, and $\{y_i\}_{i=1}^n$. Likewise, the client is assumed to have stored the $n+1$ public values y and $\{y'_i\}_{i=1}^n$. (Alternatively, a trusted certification authority (CA) could certify these values, but we choose to keep our model as simple as possible.)

4.2 Client Setup

A client $C \in \mathit{Clients}$ has a secret password π_C drawn from a set $\mathit{Password}_C$. We assume $\mathit{Password}_C$ can be mapped into $\mathbb{Z}_q^*$, and for the remainder of the paper, we use passwords as if they were elements of $\mathbb{Z}_q^*$. C creates an ElGamal ciphertext E_C of the value $g^{(\pi_C)^{-1}}$, using the servers' global public key y. More precisely, he selects $\alpha \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and computes $E_C \leftarrow (y^\alpha g^{(\pi_C)^{-1}}, g^\alpha)$. He sends E_C to each of the servers S_i, $1 \le i \le n$, who record (C, E_C) in their database. (Alternatively, a trusted CA could be used, but again we choose to keep our model as simple as possible.) We consider E_C to be public information, and in our protocol, we assume that the client knows E_C. The client could simply store E_C, or obtain a (certified) copy of E_C through interaction with the servers. (It should be clear that storing E_C at the client is not the same as storing a shared secret key.) We also assume the adversary does not observe or participate in the system and client setup phases. (Of course, the adversary could learn E_C by corrupting any server. Indeed, this is why we cannot assume E_C is private.)

4.3 Client Login Protocol

A high level description of the protocol is given in Figure 1, and the formal description is given in Appendix B. Our protocol for a client $C \in \mathit{Clients}$ relies on a simulation-sound non-interactive zero-knowledge proof (SS-NIZKP) scheme (see Appendix A for the definition of an SS-NIZKP scheme) $\mathcal{Q} = (\mathsf{Prove}_{\Phi_\mathcal{Q}}, \mathsf{Verify}_{\Phi_\mathcal{Q}}, \mathsf{Sim}_{\Phi_\mathcal{Q}})$ over a language defined by a predicate $\Phi_\mathcal{Q}$ that takes elements of $\{0,1\}^* \times (G_q \times G_q)^3$ and is defined as

$$\Phi_\mathcal{Q}(\tau, E_C, B, V) \stackrel{\text{def}}{=} \exists \beta, \pi, \gamma : \left(B = \left(y^\beta, g^\beta\right) \times (E_C)^\pi \times (g^{-1}, 1)\right) \textbf{ and } (V = (h^\gamma g^\pi, g^\gamma)) .$$

The algorithms $\mathsf{Prove}_{\Phi_\mathcal{Q}}$, $\mathsf{Verify}_{\Phi_\mathcal{Q}}$, and $\mathsf{Sim}_{\Phi_\mathcal{Q}}$ use a random oracle H_3. $\mathsf{Prove}_{\Phi_\mathcal{Q}}$ may be implemented in a standard way as a three-move honest-verifier proof made non-interactive by using the hash function to generate the verifier's random challenge, and having τ be an extra input to the hash function. Other proofs defined below may be implemented similarly.

Here we discuss Figure 1. The client $C \in \mathit{Clients}$ receives a set I of k servers in *Servers* and initiates the protocol with that set, by broadcasting I along with its own identity C. (As stated above, we assume aggregation and broadcast functionalities in the network for the communication between the client and the servers, and among the servers themselves.) In return C receives nonces from the servers in I. The client first generates a *session public key* $\tilde{y}$. The client then "removes" the password from the ciphertext E_C by raising it to π_C and dividing g out of the first element of the tuple, and reblinds the result to form B. The quantity V is then formed to satisfy the predicate $\Phi_\mathcal{Q}$, and an SS-NIZKP σ is created to bind B, V, $\tilde{y}$, and the nonces from the servers. This SS-NIZKP also forces the client to behave properly, and in particular allows a simulator in the proof of security to operate correctly. (The idea is similar to the use of a second encryption to achieve (lunchtime) chosen-ciphertext security in [35].) After verifying the SS-NIZKP, if the client has used the password $\pi = \pi_C$, it will be that $B[1] = y^{\beta+\alpha\pi}$ and $B[2] = g^{\beta+\alpha\pi}$. The servers then run $\mathsf{DistVerify}(\tau, B, V)$ to verify that $\log_g y = \log_{B[2]} B[1]$. Effectively, they are verifying (without decryption) that B is a valid encryption of the plaintext message 1. Each server S_i then computes a session key K_i, which has also been computed by the client.

Intuitively, an honest client in this protocol does not reveal any password information since he simply sends an encryption of 1, along with V, which is an encryption of g^{π_C} under a public key for which no one knows the secret key. However, one must consider the case of an adversary impersonating a client using $\pi \neq \pi_C$, and colluding with up to $k-1$ dishonest servers. Here we must rely on $\mathsf{DistVerify}(\tau, B, V)$ to prevent leakage of information on π_C. We discuss this below.

Efficiency For the following calculations we use the proof constructions of Appendix A. Recall that there are k servers involved in the execution of the protocol. The protocol requires six rounds, where each round is an exchange of messages among some of the participants. All messages are of length proportional to the size of a group element. The client is involved in only the first three rounds, while the servers are involved in all rounds. The client performs $15 + k$ exponentiations, and each server performs $14 + 38k$ exponentiations.

Remark 4.2 These costs are obviously much higher than the Ford-Kaliski scheme, but remember that our protocol is the first to achieve *provable* security (in the random oracle model). Also, the costs may be reasonable for practical implementations with k in the range of 2 to 5.

Remark 4.3 Our protocol does not provide forward security. To achieve forward security, each server S_i would need to generate its Diffie-Hellman values dynamically, instead of simply using y'_i. Then these values would need to be certified somehow by S_i to protect the client against a man-in-the-middle attack. Details are beyond the scope of this paper.

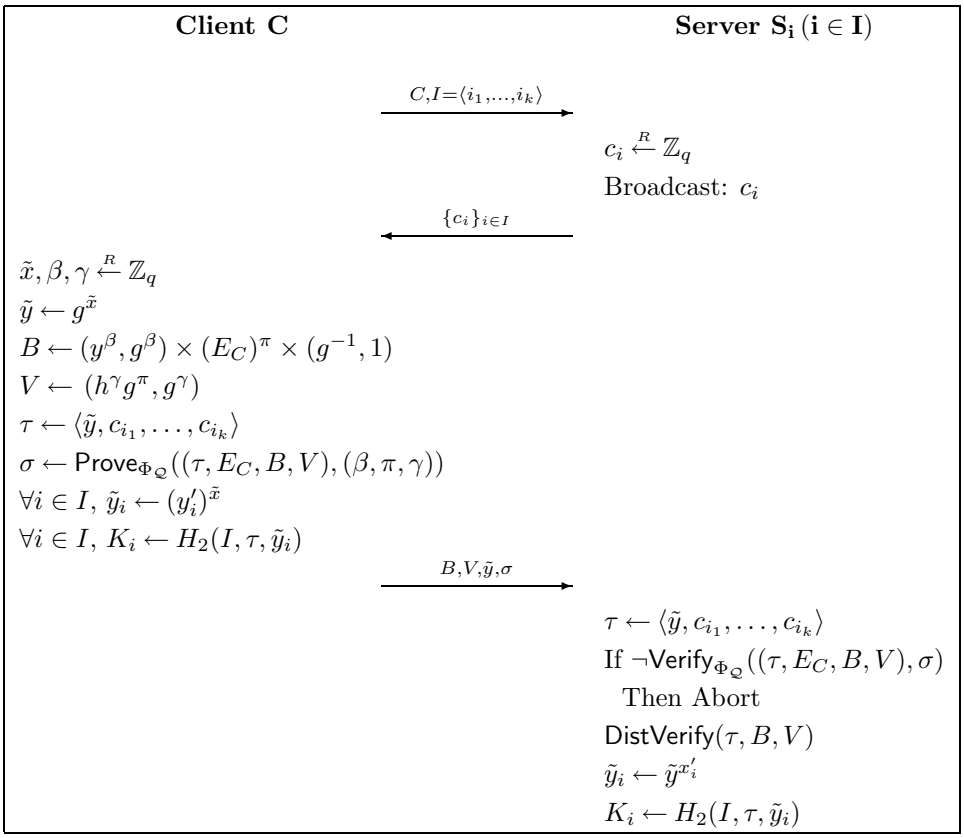

Figure 1: Protocol P.

4.4 The DistVerify Protocol

The DistVerify protocol takes three parameters, τ, B, and V, and is run by the servers $\{S_i\}_{i\in I}$ to verify that $\log_g y = \log_{B[2]} B[1]$, i.e., B is an encryption of 1. The parameter V is used in order to allow a proof of security. The protocol is shown in Figure 2, and uses the standard notation for Lagrange coefficients: $\lambda_{j,I} = \prod_{\ell\in I\setminus\{j\}} \frac{-\ell}{j-\ell} \bmod q$. The basic idea of the protocol is as follows. First the servers distributively compute $(\overline{y}, \overline{g}) \leftarrow B^r \times (y, g)^{r'}$, i.e., they use the (standard) technique which randomizes the quotient $B[1]/(B[2])^x$ if and only if it is not equal to 1. (This is basically how we prevent leakage of password information in the case of an adversary impersonating a client discussed above.) Then they take the second component (i.e., $\overline{g}$) and distributively compute $\overline{g}^x$ using their shared secrets. Finally they verify that $\overline{g}^x = \overline{y}$, implying (with high probability) that $B[1] = (B[2])^x$, and hence B is an encryption of 1.

In more detail, notice that in Step 1, when a server S_i computes B_i (its own randomization of B), it also computes auxiliary encryptions V_i, V'_i, and V''_i, which use the same randomization

value and are computed using V. Similar to V, these auxiliary encryptions are used only in order to allow a proof of security. Finally in Step 1, an SS-NIZKP is computed to force the server to behave properly. (Again, the idea here is similar to the use of a second encryption to achieve (lunchtime) chosen-ciphertext security in [35]. It gives the simulator an alternate way to determine the adversary's behavior, and in particular, allows the simulator to compute decryption shares for honest servers without knowing the decryption key.)

In Step 2, the pair $(\overline{y}, \overline{g})$ is computed, along with a partial computation of $\overline{g}^x$ using the server's individual share of x. However, this partial computation value is not revealed yet. First the server essentially proves that he knows how to perform the partial computation (i.e., that he knows his share of x). This proof is also dependent on τ', which basically includes the important parts of the transcripts of all servers. The value τ' is included so all uncompromised servers can agree on the shared values they are using to compute $(\overline{y}, \overline{g})$ before revealing their partial computations, so as to not leak any information.

In Step 3, once the server receives valid proofs from Step 2, it reveals its partial computation of $\overline{g}^x$ and proves that this computation was performed correctly.

In Step 4, once a server receives partial computations from all servers along with valid proofs, it tests whether $\overline{y} = \overline{g}^x$.

DistVerify uses an SS-NIZKP scheme $\mathcal{R} = (\mathsf{Prove}_{\Phi_\mathcal{R}}, \mathsf{Verify}_{\Phi_\mathcal{R}}, \mathsf{Sim}_{\Phi_\mathcal{R}})$ over a language defined by a predicate $\Phi_\mathcal{R}$ that takes elements of $\mathbb{Z} \times (G_q \times G_q)^6$ and is defined as

$$\begin{aligned}\Phi_\mathcal{R}(i, B, V, B_i, V_i, V_i', V_i'') \stackrel{\text{def}}{=} \; & \exists r_i, r_i', \gamma_i, \gamma_i', \gamma_i'' : B_i = B^{r_i} \times (y, g)^{r_i'} \textbf{ and} \\ & V_i = (h^{\gamma_i} g^{r_i}, g^{\gamma_i}) \textbf{ and } V_i' = (h^{\gamma_i'} (V[1])^{r_i}, g^{\gamma_i'}) \textbf{ and} \\ & V_i'' = (h^{\gamma_i''} (V[2])^{r_i}, g^{\gamma_i''}).\end{aligned}$$

The algorithms $\mathsf{Prove}_{\Phi_\mathcal{R}}$, $\mathsf{Verify}_{\Phi_\mathcal{R}}$, and $\mathsf{Sim}_{\Phi_\mathcal{R}}$ use a random oracle H_4.

DistVerify also uses an SS-NIZKP scheme $\mathcal{S} = (\mathsf{Prove}_{\Phi_\mathcal{S}}, \mathsf{Verify}_{\Phi_\mathcal{S}}, \mathsf{Sim}_{\Phi_\mathcal{S}})$ over a language defined by a predicate $\Phi_\mathcal{S}$ that takes elements of $\mathbb{Z} \times \{0,1\}^* \times G_q \times (G_q \times G_q)$ and is defined as

$$\Phi_\mathcal{S}(i, \tau', C_i, R_i) \stackrel{\text{def}}{=} \exists a_i, \zeta : C_i = g^{a_i} \textbf{ and } R_i = (h^\zeta (h')^{a_i}, g^\zeta).$$

The algorithms $\mathsf{Prove}_{\Phi_\mathcal{S}}$, $\mathsf{Verify}_{\Phi_\mathcal{S}}$, and $\mathsf{Sim}_{\Phi_\mathcal{S}}$ use a random oracle H_5.

Finally, DistVerify uses an SS-NIZKP scheme $\mathcal{T} = (\mathsf{Prove}_{\Phi_\mathcal{T}}, \mathsf{Verify}_{\Phi_\mathcal{T}}, \mathsf{Sim}_{\Phi_\mathcal{T}})$ over a language defined by a predicate $\Phi_\mathcal{T}$ that takes elements of $\mathbb{Z} \times \{0,1\}^* \times G_q \times G_q \times G_q \times (G_q \times G_q)$ and is defined as

$$\Phi_\mathcal{T}(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i) \stackrel{\text{def}}{=} \exists a_i, \zeta : \overline{C}_i = \overline{g}^{a_i} \textbf{ and } C_i = g^{a_i} \textbf{ and } R_i = (h^\zeta (h')^{a_i}, g^\zeta).$$

The algorithms $\mathsf{Prove}_{\Phi_\mathcal{T}}$, $\mathsf{Verify}_{\Phi_\mathcal{T}}$, and $\mathsf{Sim}_{\Phi_\mathcal{T}}$ use a random oracle H_6.

5 Security of the Protocol

Here we state the DDH assumption. Following that we prove that the protocol P is secure, based on the DDH assumption.

Decision Diffie-Hellman. Here we formally state the DDH assumption. For full details, see [8]. Let G_q be as in Section 3, with generator g. For two values $X = g^x$ and $Y = g^y$, let $\mathsf{DH}(X, Y) = g^{xy}$.

Step 1: $r_i, r_i', \gamma_i, \gamma_i', \gamma_i'' \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$B_i \leftarrow B^{r_i} \times (y, g)^{r_i'} \quad V_i \leftarrow (h^{\gamma_i} g^{r_i}, g^{\gamma_i})$
$V_i' \leftarrow (h^{\gamma_i'}(V[1])^{r_i}, g^{\gamma_i'}) \quad V_i'' \leftarrow (h^{\gamma_i''}(V[2])^{r_i}, g^{\gamma_i''})$
$\sigma_i \leftarrow \mathsf{Prove}_{\Phi_{\mathcal{R}}}((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$
Broadcast $(B_i, V_i, V_i', V_i'', \sigma_i)$

Step 2: $\forall j \in I \setminus \{i\}$: Receive $(B_j, V_j, V_j', V_j'', \sigma_j)$
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_{\mathcal{R}}}((j, B, V, B_j, V_j, V_j', V_j''), \sigma_j)$ Then Abort
$(\overline{y}, \overline{g}) \leftarrow \prod_{j \in I} B_j$
$\tau' \leftarrow \langle \tau, B, V, B_{i_1}, \ldots, B_{i_k}, V_{i_1}, \ldots, V_{i_k} \rangle$
$a_i \leftarrow \lambda_{i,I} x_i \quad \overline{C}_i \leftarrow \overline{g}^{a_i} \quad \zeta \stackrel{R}{\leftarrow} \mathbb{Z}_q \quad R_i \leftarrow (h^{\zeta}(h')^{a_i}, g^{\zeta})$
$\forall j \in I : C_j \leftarrow (y_j)^{\lambda_{j,I}}$
$\Gamma_i \leftarrow \mathsf{Prove}_{\Phi_{\mathcal{S}}}((i, \tau', C_i, R_i), (a_i, \zeta))$
Broadcast (R_i, Γ_i)

Step 3: $\forall j \in I \setminus \{i\}$: Receive (R_j, Γ_j)
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_{\mathcal{S}}}((j, \tau', C_j, R_j), \Gamma_j)$ Then Abort
$\Gamma_i' \leftarrow \mathsf{Prove}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a_i, \zeta))$
Broadcast $(\overline{C}_i, \Gamma_i')$

Step 4: $\forall j \in I \setminus \{i\}$: Receive $(\overline{C}_j, \Gamma_j')$
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_{\mathcal{T}}}((j, \tau', \overline{g}, \overline{C}_j, C_j, R_j), \Gamma_j')$ Then Abort
If $\Pi_{j \in I} \overline{C}_j \neq \overline{y}$ Then Abort

Figure 2: Protocol $\mathsf{DistVerify}(\tau, B, V)$ for Server S_i $(i \in I)$.

Let $\mathcal{A}$ be an algorithm that on input (X, Y, Z) outputs "1" if it believes that $Z = \mathsf{DH}(X, Y)$, and "0" otherwise. For any $\mathcal{A}$ running in time t,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{DDH}}_{G_q}(\mathcal{A}) \stackrel{\mathrm{def}}{=} & \Pr\left[x, y \stackrel{R}{\leftarrow} \mathbb{Z}_q; X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^{xy} : \mathcal{A}(X, Y, Z) = 1\right] \\ & - \Pr\left[x, y, z \stackrel{R}{\leftarrow} \mathbb{Z}_q; X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^z : \mathcal{A}(X, Y, Z) = 1\right].\end{aligned}$$

Let $\mathsf{Adv}^{\mathrm{DDH}}_{G_q}(t) = \max_{\mathcal{A}}\left\{\mathsf{Adv}^{\mathrm{DDH}}_{G_q}(\mathcal{A})\right\}$, where the maximum is taken over all adversaries of time complexity at most t. The DDH assumption states that for any probabilistic polynomial-time algorithm $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{DDH}}_{G_q}(\mathcal{A})$ is negligible (in $\kappa = |q|$).

5.1 Protocol *P*

Here we prove that protocol P is secure, in the sense that an adversary attacking the system that compromises fewer than k out of n servers cannot determine session keys with significantly greater advantage than that of an online dictionary attack. Recall that we consider only *static* compromising of servers, i.e., the adversary chooses which servers to compromise before the execution of the system. Let t_{exp} be the time required to perform an exponentiation in G_q.

P_0 The original protocol P.

P_1 The nonces are assumed to be distinct (and thus Reveal queries do not reveal anything that could help in a Test query).

P_2 The Diffie-Hellman key exchange between a client and an uncompromised server is replaced with a perfect key exchange (and thus an adversary that does not succeed in impersonating a client to an uncompromised server does not obtain any information that could help in a Test query).

P_3 Value V from a client, and values V_i, V_i', V_i'', R_i from uncompromised servers, are replaced by random values. The $\mathcal{Q}$-SS-NIZKP σ, and each $\mathcal{R}$-SS-NIZKP σ_i, $\mathcal{S}$-SS-NIZKP Γ_i, and $\mathcal{T}$-SS-NIZKP Γ_i', are constructed using the associated simulators.

P_4 Value B from a client is replaced with a random encryption of 1, and authentication of an honest client is changed so that uncompromised servers compute $\overline{C}_i$ values without using their secret shares. Also, $H_0(y)$ returns a value h with a known discrete log.

P_5 The adversary succeeds if it ever sends a V value associated with the correct password.

P_6 Abort if the adversary creates a new and valid $\mathcal{S}$-SS-NIZKP or $\mathcal{T}$-SS-NIZKP associated with an uncompromised server.

P_7 Value E_C for each client is changed to a random value, and on any adversary login attempt for C, the B_i and $\overline{C}_i$ values from uncompromised servers are replaced with random values (so as to force a failure).

Figure 3: Informal description of protocols P_0 through P_7

Theorem 5.1 *Let P be the protocol described in Figure 1 and Figure 2, (and formally described in Appendix B), 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 t, makes $n_{\rm ex}, n_{\rm re}$ queries of type* Execute, Reveal*, respectively, makes $n_{\rm ro}$ queries directly to the random oracles, and starts at most $n_{\rm in}$ client and server instances. Then for $t' = O(t + (n_{\rm ro} + kn_{\rm in} + k^2 n_{\rm ex})t_{exp})$:*

$$\mathsf{Adv}_P^{\rm dake}(\mathcal{A}) \leq \frac{n_{\rm in}}{N} + O\left(\mathsf{Adv}_{G_q}^{\rm DDH}(t') + \frac{n^2 + kn_{\rm ro}n_{\rm in} + n_{\rm ro}n + (n_{\rm in} + kn_{\rm ex})^2}{q} + \frac{(n_{\rm in} + kn_{\rm ex})(n_{\rm ro} + n_{\rm in} + kn_{\rm ex})}{q^2}\right).$$

Proof: We begin with a sketch of the proof, and later provide the details.

Sketch: Our proof will proceed by introducing a series of protocols $P_0, P_1, \ldots, P_7$ related to P, with $P_0 = P$. In P_7, $\mathcal{A}$ will be reduced to simply "guessing" the correct password π_C. We describe these protocols informally in Figure 3. For each i from 1 to 7, we will prove that the difference between the advantage of $\mathcal{A}$ attacking protocols P_{i-1} and P_i is negligible.

$P_0 \rightarrow P_1$ The probability of a collision of nonces is easily seen to be negligible.

$P_1 \rightarrow P_2$ This can be shown using a standard reduction from DDH. On input (X, Y, Z), we plug in random powers of Y for the servers' local public keys, and random powers of X for the clients' $\tilde{y}$ values, and then check H_2 queries for appropriate powers of Z.

$P_2 \rightarrow P_3$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for $h = H_0(y)$, and we use X and Z to create (randomized) encryptions for all V, V_i, V_i', V_i'', and R_i values. Also, we must factor in the negligible probability of a simulation error in one of the SS-NIZKPs.

$P_3 \rightarrow P_4$ This is straightforward, since the view of the adversary is indistinguishable in these two protocols.

$P_4 \rightarrow P_5$ This is straightforward, since this could only increase the probability of the adversary succeeding. Below we will use the fact that the discrete log of h is known, and that the $\overline{C}_i$ value computed when authenticating a client's B value by an uncompromised server does not use the secret share of that server.

$P_5 \rightarrow P_6$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, simulate the public shares of the uncompromised servers, and let $h' = X$. Given a correct SS-NIZKP for an uncompromised server, we can compute $(h')^x$, where $y = g^x$ (where x is not known). Then we simply check if $Z = (h')^x$.

There is a difficulty now in performing authentication on B values chosen by the adversary, since we do not know the secret shares (the x_i values) for the uncompromised servers. Therefore to perform authentication, we use the fact that discrete log of h is known so we can decrypt all V, V_i, V_i', and V_i'' values, and then use these decryptions to aid in computing the correct value of $\overline{g}^x$ (even though we don't know x). Finally, we generate $\overline{C}_i$ values from uncompromised servers in such a way that the product is $\overline{g}^x$, similar to the way the $\overline{C}_i$ values are computed by uncompromised servers for authentication of a client's B value. Note that the SS-NIZKPs are already being simulated.

$P_6 \rightarrow P_7$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, simulate the public shares of the uncompromised servers, and use X and Z to create (randomized) encryptions for all E_C values. This does not affect authentication using B values generated by clients (since these values are random encryptions of 1 at this point, anyway). The difficulty is in obtaining the correct distribution of $\overline{C}_i$ values while authenticating B values chosen by the adversary. To do this we use X and Z in our creation of the B_i values for uncompromised servers, which leaves $\overline{C}_i$ values correct if (X, Y, Z) is a true DH triple, but has the effect of randomizing the $\overline{C}_i$ values if (X, Y, Z) is a random triple. Again, the decryptions of V, V_i, V_i', and V_i'' are used to aid in computing the true $\overline{g}^x$ value (even though we don't know x) when (X, Y, Z) is a true DH triple, or the appropriate random value, when (X, Y, Z) is a random triple.

One can see that in P_2, an adversary that does not succeed in impersonating a client to an uncompromised server gains negligible advantage in determining a real session key from a random session key. The remainder of the protocols are used to show that an adversary gains negligible advantage

in impersonating a client over a simple online guessing attack. In particular, in P_7 the password is only used to check V values submitted by the adversary attempting to impersonate a client. The theorem follows.

Details: We use the terminology "in a CLIENT ACTION i query to C" to mean "in a Send or Execute query to C that results in the CLIENT ACTION i procedure being executed," and "in a SERVER ACTION i query to S" to mean "in a Send or Execute query to S that results in the SERVER ACTION i procedure being executed." Details of these procedures can be found in the formal specification of the protocol in Appendix B.

We assume without loss of generality that k, n, n_{ro}, and $n_{\text{in}} + n_{\text{ex}}$ are all at least 1. In the following protocols, we let J be the set of indices of compromised servers. Without loss of generality, we assume $|J| = k - 1$. For each uncompromised server S_i, let $J_i = \{i\} \cup J$.

Protocol P_1. Let E be the event that two server keys y_i' and y_j' are the same, or that a server S_i generates the same nonce c_i in SERVER ACTION 1 queries in two different instances, or that one or more clients generate the same $\tilde{y}$ value in different CLIENT ACTION 2 queries. 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 Reveal $(C,\ i,\ S_j)$ query where Π_i^C generated key $sk_{C,S_j}^i = H_2(I, \tau, \tilde{y}_j)$ or a Reveal $(S_j,\ i)$ query where $\Pi_i^{S_j}$ generated key $sk_{S_j}^i = H_2(I, \tau, \tilde{y}_j)$, and there is another client or server instance that generates a key using $H_2(I, \tau, \tilde{y}_j)$ that is not a partner to the instance corresponding to the Reveal query.

Claim 5.2 For any adversary $\mathcal{A}$,

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

Proof: Straightforward. ∎

Protocol P_2. Let E be the event that $\mathcal{A}$ makes an $H_2(I, \tau, \tilde{y}_i)$ query for a value $\tilde{y}_i = \mathsf{DH}(\tilde{y}, y_i')$ for some key y_i' belonging to an uncompromised server S_i, and for some $\tilde{y}$ generated in a CLIENT ACTION 2 query to a client C that generated τ. Let P_2 be a protocol that is identical to P_1 except that if E occurs, the protocol aborts (and thus the adversary fails).

Claim 5.3 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\text{ro}} + kn_{\text{in}} + k^2 n_{\text{ex}})t_{exp})$ such that

$$\mathsf{Adv}_{P_1}^{\text{dake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_2}^{\text{dake}}(\mathcal{A}) + 2\mathsf{Adv}_{G_q}^{\text{DDH}}(t') + \frac{O(n_{\text{in}} + n_{\text{ex}} + n_{\text{ro}} n)}{q}.$$

Proof: Let ϵ be the probability that E occurs when $\mathcal{A}$ is running against protocol P_1. Then $\Pr(\mathsf{Succ}_{P_1}^{\text{dake}}(\mathcal{A})) \leq \Pr(\mathsf{Succ}_{P_2}^{\text{dake}}(\mathcal{A})) + \epsilon$, and thus by Fact 2.2, $\mathsf{Adv}_{P_1}^{\text{dake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_2}^{\text{dake}}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts 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_1 for $\mathcal{A}$ with these changes:

1. In the initialization procedure, for each uncompromised server S_i, replace the normal generation of y'_i with $y'_i \leftarrow Y^{\rho_i}$ where $\rho_i \stackrel{R}{\leftarrow} \mathbb{Z}_q$.

2. In a CLIENT ACTION 2 query to a client C, replace the normal generation of $\tilde{y}$ with $\tilde{y} \leftarrow X^{\psi}$, where $\psi \stackrel{R}{\leftarrow} \mathbb{Z}_q$. Then for each uncompromised server S_i, replace the normal generation of K_i with $K_i \stackrel{R}{\leftarrow} \{0,1\}^{\kappa}$.

3. In a SERVER ACTION 6 query to an uncompromised server S_i, if $\tilde{y}$ was generated in a CLIENT ACTION 2 query to a client C, replace the normal generation of K with $K \leftarrow K_i$ for the K_i value generated by that CLIENT ACTION 2 query.

4. In an H_2 query, if the query is $(I, \tau, Z^{\rho_i \psi})$, for I, τ, and ψ generated in a CLIENT ACTION 2 query to a client C, and any ρ_i generated in the initialization procedure for uncompromised server S_i with $i \in I$, D outputs 1 and halts.

5. If $\mathcal{A}$ finishes, D outputs 0 and halts.

If (X, Y, Z) is drawn from the set of DH triples, this simulation is perfectly indistinguishable from P_1 until E occurs, when the simulation halts and D outputs 1. If (X, Y, Z) is drawn from the set of random triples, then D outputs 1 only if $\mathcal{A}$ happens to query H_2 with a third parameter equal to one of the values tested in the simulation. This could happen if one of the ρ_i or ψ values generated by the simulator is zero or if none of those values are zero but the Z value is such that one of the n_{ro} queries made by the adversary to H_2 has a third parameter equal to $Z^{\rho_i \psi}$. Recalling that n is the total number of servers, and thus an upper bound on the number of uncompromised servers, the former probability is at most $\frac{n+n_{\text{in}}+n_{\text{ex}}}{q}$, and the latter is at most $\frac{n_{\text{ro}} n}{q}$,

Let t' be the running time of D, and note that $t' = O(t + (n_{\text{ro}} + k n_{\text{in}} + k^2 n_{\text{ex}}) t_{exp})$. The advantage of D is

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

The claim follows from the fact that $\mathsf{Adv}^{\text{DDH}}_{G_q}(D) \leq \mathsf{Adv}^{\text{DDH}}_{G_q}(t')$. ∎

Protocol P_3. Let P_3 be a protocol that is identical to P_2 except for the following, where $\mathsf{Simhash}$ and $\mathsf{Simprove}$ refer to the simulated hash functions and simulated provers described in Appendix A.

1. H_3 queries are answered by $\mathsf{Simhash}_{\Phi_{\mathcal{Q}}}$, and in a CLIENT ACTION 2 query to client C, $V \stackrel{R}{\leftarrow} G_q \times G_q$ and σ is constructed using $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$.

2. H_4 queries are answered by $\mathsf{Simhash}_{\Phi_{\mathcal{R}}}$, and in a SERVER ACTION 3 query to an uncompromised server S_i, $V_i, V'_i, V''_i \stackrel{R}{\leftarrow} G_q \times G_q$ and σ_i is constructed using $\mathsf{Simprove}_{\Phi_{\mathcal{R}}}$.

3. H_5 queries are answered by $\mathsf{Simhash}_{\Phi_{\mathcal{S}}}$, and in a SERVER ACTION 4 query to an uncompromised server S_i, $R_i \stackrel{R}{\leftarrow} G_q \times G_q$ and Γ_i is constructed using $\mathsf{Simprove}_{\Phi_{\mathcal{S}}}$.

4. H_6 queries are answered by $\mathsf{Simhash}_{\Phi_\mathcal{T}}$, and in a SERVER ACTION 5 query to an uncompromised server S_i, Γ_i' is constructed using $\mathsf{Simprove}_{\Phi_\mathcal{T}}$.

If $\mathsf{Simprove}_{\Phi_\mathcal{Q}}$, $\mathsf{Simprove}_{\Phi_\mathcal{R}}$, $\mathsf{Simprove}_{\Phi_\mathcal{S}}$, or $\mathsf{Simprove}_{\Phi_\mathcal{T}}$ fails, P_3 aborts.

Claim 5.4 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\text{ro}} + kn_{\text{in}} + k^2 n_{\text{ex}})t_{exp})$ such that

$$\mathsf{Adv}^{\text{dake}}_{P_2}(\mathcal{A}) \leq \mathsf{Adv}^{\text{dake}}_{P_3}(\mathcal{A}) + 2\mathsf{Adv}^{\text{DDH}}_{G_q}(t') + \frac{2}{q} + \frac{O((n_{\text{in}} + kn_{\text{ex}})(n_{\text{ro}} + n_{\text{in}} + kn_{\text{ex}}))}{q^2}.$$

Proof: Assume $\mathsf{Adv}^{\text{dake}}_{P_2}(\mathcal{A}) = \mathsf{Adv}^{\text{dake}}_{P_3}(\mathcal{A}) + 2\epsilon$, and thus by Fact 2.2, $\Pr(\mathsf{Succ}^{\text{dake}}_{P_2}(\mathcal{A})) = \Pr(\mathsf{Succ}^{\text{dake}}_{P_3}(\mathcal{A})) + \epsilon$.

Now we construct an algorithm D that attempts 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_3 for $\mathcal{A}$ with the following changes:

1. A query $H_0(y)$ returns Y. (Recall that $h = H_0(y)$.)
2. In a CLIENT ACTION 2 query to a client C, $V \leftarrow (Z^{\mu_1} Y^{\mu_2} g^{\pi_C}, X^{\mu_1} g^{\mu_2})$, where $\mu_1, \mu_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$. Also, if $\mathsf{Simprove}_{\Phi_\mathcal{Q}}$ fails, D outputs 0 and halts.
3. In a SERVER ACTION 3 query to an uncompromised server S_i using parameters (B, V),

$$\begin{aligned} V_i &\leftarrow (Z^{\mu_1} Y^{\mu_2} g^{r_i}, X^{\mu_1} g^{\mu_2}) \\ V_i' &\leftarrow (Z^{\mu_1'} Y^{\mu_2'} (V[1])^{r_i}, X^{\mu_1'} g^{\mu_2'}), \text{ and} \\ V_i'' &\leftarrow (Z^{\mu_1''} Y^{\mu_2''} (V[2])^{r_i}, X^{\mu_1''} g^{\mu_2''}) \end{aligned}$$

   where $\mu_1, \mu_2, \mu_1', \mu_2', \mu_1'', \mu_2'' \stackrel{R}{\leftarrow} \mathbb{Z}_q$. Also, if $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ fails, D outputs 0 and halts.
4. In a SERVER ACTION 4 query to an uncompromised server S_i

$$R_i \leftarrow (Z^{\mu_1} Y^{\mu_2} (h')^{a_i}, X^{\mu_1} g^{\mu_2})$$

   where $\mu_1, \mu_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$. Also, if $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ fails, D outputs 0 and halts.
5. In a SERVER ACTION 5 query to an uncompromised server S_i, if $\mathsf{Simprove}_{\Phi_\mathcal{T}}$ fails, D outputs 0 and halts.
6. If $\mathcal{A}$ succeeds against this simulation, D outputs 1 and halts, else D outputs 0 and halts.

When (X, Y, Z) is drawn from the set of DH triples, the simulation is statistically indistinguishable from P_2, with difference at most

$$\begin{aligned} &\textsc{Simerr}_\mathcal{Q}(n_{\text{ro}}, n_{\text{in}} + n_{\text{ex}}) + \textsc{Simerr}_\mathcal{R}(n_{\text{ro}}, n_{\text{in}} + kn_{\text{ex}}) + \\ &\textsc{Simerr}_\mathcal{S}(n_{\text{ro}}, n_{\text{in}} + kn_{\text{ex}}) + \textsc{Simerr}_\mathcal{T}(n_{\text{ro}}, n_{\text{in}} + kn_{\text{ex}}) \end{aligned}$$

$$
\begin{aligned}
&\leq \frac{(n_{\text{in}}+n_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+n_{\text{ex}})}{q^3}+\frac{(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^5}+\\
&\quad \frac{(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^2}+\frac{(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^2}\\
&\leq \frac{4(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^2}.
\end{aligned}
$$

When (X,Y,Z) is drawn from the set of random triples, the simulation is statistically indistinguishable from P_3, the statistical difference coming from the $\frac{1}{q}$ probability that (X,Y,Z) is actually a DH triple.

Let t' be the running time for D, and note that $t' = O(t+(n_{\text{ro}}+kn_{\text{in}}+k^2n_{\text{ex}})t_{exp})$. The advantage of D is

$$
\begin{aligned}
\mathsf{Adv}^{\text{DDH}}_{G_q}(D) &= \Pr\left[D \text{ outputs } 1|\text{DH triple}\right]-\Pr\left[D \text{ outputs } 1|\text{random triple}\right]\\
&\geq \Pr\left[\mathsf{Succ}^{\text{dake}}_{P_2}(\mathcal{A})\right]-\frac{4(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^2}-\Pr\left[\mathsf{Succ}^{\text{dake}}_{P_3}(\mathcal{A})\right]-\frac{1}{q}\\
&= \epsilon-\frac{1}{q}-\frac{4(n_{\text{in}}+kn_{\text{ex}})(n_{\text{ro}}+n_{\text{in}}+kn_{\text{ex}})}{q^2},
\end{aligned}
$$

The claim follows from the fact that $\mathsf{Adv}^{\text{DDH}}_{G_q}(D) \leq \mathsf{Adv}^{\text{DDH}}_{G_q}(t')$. ▌

Simulation Soundness. We will define Fraud as the event that the adversary is able to produce a new valid SS-NIZKP (of one of the four types used in the protocol) for a statement that does not satisfy the particular relation corresponding to that type of SS-NIZKP. Formally, let E_1 be the event that $\mathcal{A}$ sends a valid $\mathcal{Q}$-SS-NIZKP σ in a SERVER ACTION 3 query for a string (τ, E_C, B, V) not previously used in a CLIENT ACTION 2 query to client C, where (τ, E_C, B, V) does not satisfy $\Phi_{\mathcal{Q}}$. Let E_2 be the event that $\mathcal{A}$ sends a valid $\mathcal{R}$-SS-NIZKP σ_i for a server S_i in a SERVER ACTION 4 query to any server S_j for a string $(i, B, V, B_i, V_i, V_i', V_i'')$ not previously used in a SERVER ACTION 3 query to server S_i, where $(i, B, V, B_i, V_i, V_i', V_i'')$ does not satisfy $\Phi_{\mathcal{R}}$. Let E_3 be the event that $\mathcal{A}$ sends a valid $\mathcal{S}$-SS-NIZKP Γ_i for a server S_i in a SERVER ACTION 5 query to any server S_j for a string (i, τ', C_i, R_i) that was not previously used in a SERVER ACTION 4 query to server S_i, where (i, τ', C_i, R_i) does not satisfy $\Phi_{\mathcal{S}}$. Let E_4 be the event that $\mathcal{A}$ sends a valid $\mathcal{T}$-SS-NIZKP Γ_i' for a server S_i in a SERVER ACTION 6 query to any server S_j for a string $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$ that was not previously used in a SERVER ACTION 5 query to server S_i, where $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$ does not satisfy $\Phi_{\mathcal{T}}$.

Let $\mathsf{Fraud} = E_1 \vee E_2 \vee E_3 \vee E_4$.

Claim 5.5 For any adversary $\mathcal{A}$ running against P_3,

$$
\Pr(\mathsf{Fraud}) \leq \frac{4kn_{\text{in}}(n_{\text{ro}}+1)}{q}.
$$

Proof: We split the proof into four parts.

Part 1 Let ϵ be the probability that E_1 occurs when $\mathcal{A}$ is running against protocol P_3. We construct an algorithm D that simulates P_3 with the following change: D guesses which SERVER ACTION 3 query will cause E_1 to occur, and on that query D outputs the pair $((\tau, E_C, B, V), \sigma)$ associated with that query and halts. The simulation is indistinguishable from P_3 until D halts. D outputs a valid $\mathcal{Q}$-SS-NIZKP σ for a string (τ, E_C, B, V) that does not satisfy $\Phi_{\mathcal{Q}}$ with probability $\frac{\epsilon}{n_{\text{in}}}$. This implies $\text{SERR}_{\mathcal{Q}}(n_{\text{ro}}, n_{\text{in}} + n_{\text{ex}}) \geq \frac{\epsilon}{n_{\text{in}}}$, where $\text{SERR}_{\mathcal{Q}}(n_{\text{ro}}, n_{\text{pr}})$ is the soundness error of the $\mathcal{Q}$-SS-NIZKP protocol from Appendix A given n_{ro} random oracle queries and n_{pr} proof queries. For this protocol $\text{SERR}_{\mathcal{Q}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$, so $\Pr(E_1) \leq \frac{n_{\text{in}}(n_{\text{ro}}+1)}{q}$.

Part 2 In a similar way, we can show that $\Pr(E_2) \leq \frac{kn_{\text{in}}(n_{\text{ro}}+1)}{q}$. The extra factor of k comes from the fact that D must guess which of the k proofs in a SERVER ACTION 4 query cause E_2 to occur.

Part 3 Similar to Part 2, we can show that $\Pr(E_3) \leq \frac{kn_{\text{in}}(n_{\text{ro}}+1)}{q}$.

Part 4 Similar to Part 3, we can show that $\Pr(E_4) \leq \frac{kn_{\text{in}}(n_{\text{ro}}+1)}{q}$. ∎

In the following proofs we will not use Claim 5.5 directly, but instead use the fact that the bound on $\Pr(\mathsf{Fraud})$ in Claim 5.5 applies to any of the protocols and simulations that we create from this point forward. This is easy to see, since the simulator does not actually do anything but run the protocol and stop at some point to guess a fraudulent proof.

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except for the following.

1. In initialization, $\rho \xleftarrow{R} \mathbb{Z}_q$ is generated, and for y generated in initialization, $H_0(y)$ returns g^ρ. (Recall that $h = H_0(y)$.)
2. In a CLIENT ACTION 2 query to a client C, $B \leftarrow (y^\beta, g^\beta)$.
3. In a SERVER ACTION 4 query to an uncompromised server S_i that uses a B value (received in its associated SERVER ACTION 3 query) produced in a CLIENT ACTION 2 query, $\overline{C}_i \leftarrow \left(\overline{y}\left(\prod_{j\in J} \overline{g}^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{i,J_i}}$. Note that this computation does not rely on the secret shares of uncompromised servers.

Claim 5.6 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\text{dake}}_{P_3}(\mathcal{A}) \leq \mathsf{Adv}^{\text{dake}}_{P_4}(\mathcal{A}) + \frac{O(kn_{\text{in}}n_{\text{ro}})}{q}.$$

Proof: We show that if Fraud does not occur, P_4 is perfectly indistinguishable from P_3. For this we simply need to show that in a SERVER ACTION 4 query to an uncompromised server S_i that uses a B value produced in a CLIENT ACTION 2 query, the $\overline{C}_i$ value computed in P_4 and P_3 will

be the same, as long as Fraud does not occur. To see this, note that in this case, $y = g^x$ implies $\overline{y} = \overline{g}^x$, and thus

$$\begin{aligned}
\overline{C}_i^{\langle P_4 \rangle} &= \left(\overline{y} \left(\prod_{j \in J} \overline{g}^{\lambda_{j,J_i} x_j} \right)^{-1} \right)^{\lambda_{i,I}/\lambda_{i,J_i}} \\
&= \left(\overline{g}^x \left(\prod_{j \in J} \overline{g}^{\lambda_{j,J_i} x_j} \right)^{-1} \right)^{\lambda_{i,I}/\lambda_{i,J_i}} \\
&= \left(\overline{g}^{\lambda_{i,J_i} x_i} \right)^{\lambda_{i,I}/\lambda_{i,J_i}} \\
&= \overline{g}^{\lambda_{i,I} x_i} \\
&= \overline{C}_i^{\langle P_3 \rangle}.
\end{aligned}$$

Now let ϵ be the probability that Fraud occurs in P_3. Then $\Pr(\mathsf{Succ}^{\mathrm{dake}}_{P_3}(\mathcal{A})) \leq \Pr(\mathsf{Succ}^{\mathrm{dake}}_{P_4}(\mathcal{A})) + \epsilon$, and thus by Fact 2.2, $\mathsf{Adv}^{\mathrm{dake}}_{P_3}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{dake}}_{P_4}(\mathcal{A}) + 2\epsilon$. The claim follows from the fact that $\epsilon \leq \frac{4kn_{\mathrm{in}}(n_{\mathrm{ro}}+1)}{q}$. ∎

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except that in a Server Action 3 query to a server S_i, for a client C, and using parameters (τ, E_C, B, V), where (τ, E_C, B, V) was never used in a Client Action 2 query to client C, if $V[1]/(V[2])^{\rho} = g^{\pi_C}$, P_5 stops and we say that $\mathcal{A}$ succeeds.

Claim 5.7 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{dake}}_{P_4}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{dake}}_{P_5}(\mathcal{A})$$

Proof: By having P_5 stop and saying that $\mathcal{A}$ succeeds, we could only increase the probability of success of $\mathcal{A}$. ∎

Protocol P_6. Let E_1 be the event that $\mathcal{A}$ sends a valid $\mathcal{S}$-SS-NIZKP Γ_i for an uncompromised server S_i in a Server Action 5 query to any server S_j for a string (i, τ', C_i, R_i) that was not previously used in a Server Action 4 query to server S_i.

Let E_2 be the event that $\mathcal{A}$ sends a valid $\mathcal{T}$-SS-NIZKP Γ'_i for an uncompromised server S_i in a Server Action 6 query to any server S_j for a string $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$ that was not previously used in a Server Action 5 query to server S_i.

Let $E = E_1 \vee E_2$. Let P_6 be a protocol that is identical to P_5 except that if E occurs, P_6 halts and $\mathcal{A}$ fails.

Claim 5.8 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\mathrm{ro}} + kn_{\mathrm{in}} + k^2 n_{\mathrm{ex}})t_{exp})$ such that

$$\mathsf{Adv}^{\mathrm{dake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{dake}}_{P_6}(\mathcal{A}) + 2\mathsf{Adv}^{\mathrm{DDH}}_{G_q}(t') + \frac{O(kn_{\mathrm{in}}n_{\mathrm{ro}})}{q}.$$

Proof: Let ϵ be the probability that E occurs in P_5. Then $\Pr(\mathsf{Succ}^{\mathrm{dake}}_{P_5}(\mathcal{A})) \leq \Pr(\mathsf{Succ}^{\mathrm{dake}}_{P_6}(\mathcal{A})) + \epsilon$, and thus by Fact 2.2, $\mathsf{Adv}^{\mathrm{dake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{dake}}_{P_6}(\mathcal{A}) + 2\epsilon$.

The intuition behind this proof is that if E occurs, then the adversary must somehow have computed h' raised to the secret share of the uncompromised server. This essentially implies that the adversary performs a Diffie-Hellman computation on h' and the public key y. However, in a reduction argument from DDH, there is a difficulty when trying to simulate the protocol (and in particular the $\overline{C}_i$ values from uncompromised servers), since we do not know the secret shares of the uncompromised servers. Fortunately, the auxiliary values (i.e., V and the V_j, V'_j, and V''_j values from the other servers) give us enough information to simulate the $\overline{C}_i$ values.

Formally, we construct an algorithm D that attempts 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_5 for $\mathcal{A}$ with these changes:

1. In the initialization procedure, $y \leftarrow Y$ and $H_1(y) \leftarrow X$. (Recall that $h' = H_1(y)$.) For each $i \in J$, $x_i \xleftarrow{R} \mathbb{Z}_q$, and for each uncompromised server S_i, $y_i \leftarrow \left(y \left(\prod_{j \in J} g^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{1/\lambda_{i,J_i}}$.

2. In a Server Action 4 query to an uncompromised server S_i that does not use a value B produced in a Client Action 2 query to a client, do the following:

   (a) Let $I_B \subseteq I$ be the set of indices where for $j \in I_B$, the tuple (B_j, V_j, V'_j, V''_j) was not produced by server S_j, and let $I_G = I \setminus I_B$. Use the $B, V, \{j, B_j, V_j, V'_j, V''_j\}_{j \in I}$ values known to S_i to perform the following computation, where $s_\rho(V) = V[1]/(V[2])^\rho$.

   $$\overline{y}^* \leftarrow \left(\prod_{j \in I_G} B_j[1] g^{r_j} (s_\rho(V))^{-r_j/\pi_C}\right) \left(\prod_{j \in I_B} B_j[1] s_\rho(V_j) \left(s_\rho((s_\rho(V'_j), s_\rho(V''_j)))\right)^{-1/\pi_C}\right).$$

   (b) Set $\overline{C}_i \leftarrow \left(\overline{y}^* \left(\prod_{j \in J} \overline{g}^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{i,J_i}}$.

3. In a Server Action 5 query to an uncompromised server, if there is a tuple (R_i, Γ_i) that was not produced by server S_i, S_i is not compromised, and Γ_i is valid, then perform the following computation, where $s_\rho(V) = V[1]/(V[2])^\rho$.

   $$Z^* \leftarrow (s_\rho(R_i))^{\lambda_{i,J_i}(\lambda_{i,I})^{-1}} \left(\prod_{j \in J} X^{\lambda_{j,J_i} x_j}\right).$$

   If $Z = Z^*$, D halts and outputs 1, else D halts and outputs 0.

4. In a Server Action 6 query to an uncompromised server, If there is a tuple $(\overline{C}_i, R_i, \Gamma'_i)$ that was not produced by server S_i, S_i is not compromised, and Γ'_i is valid, then perform the following computation, where $s_\rho(V) = V[1]/(V[2])^\rho$.

   $$Z^* \leftarrow (s_\rho(R_i))^{\lambda_{i,J_i}(\lambda_{i,I})^{-1}} \left(\prod_{j \in J} X^{\lambda_{j,J_i} x_j}\right).$$

If $Z = Z^*$, D halts and outputs 1, else D halts and outputs 0.

5. If $\mathcal{A}$ halts, D halts and outputs 0.

First we show that if Fraud does not occur, then in a SERVER ACTION 4 query to an uncompromised server S_i that does not use a value B computed in a CLIENT ACTION 2 query to a client, $\overline{y}^* = \overline{g}^x$, where $y = g^x$. (Note that x is not necessarily known to D.) Say B is verified against client C. Without loss of generality, we may assume

$$\begin{aligned}
E_C &= (y^\alpha g^{(\pi_C)^{-1}}, g^\alpha), \\
B &= (y,g)^\beta \times (E_C)^\pi \times (g^{-1}, 1) = (y^{\beta+\alpha\pi} g^{\pi(\pi_C)^{-1}-1}, g^{\beta+\alpha\pi}), \\
V &= (h^\gamma g^\pi, g^\gamma), \\
B_j &= B^{r_j} \times (y,g)^{r'_j} = (y^{(\beta+\alpha\pi)r_j + r'_j} g^{r_j(\pi(\pi_C)^{-1}-1)}, g^{(\beta+\alpha\pi)r_j + r'_j}), \text{ and} \\
\overline{g} &= \prod_{j\in I} B_j[2] = \prod_{j\in I} g^{(\beta+\alpha\pi)r_j + r'_j} = g^{(\beta+\alpha\pi)(\sum_{j\in I} r_j) + (\sum_{j\in I} r'_j)},
\end{aligned}$$

for some $\beta, \pi, \gamma, \{r_j\}_{j\in I}, \{r'_j\}_{j\in I} \in \mathbb{Z}_q$. For $j \in I_B$, we may also assume $V_j = (h^{\gamma_j} g^{r_j}, g^{\gamma_j})$, $V'_j = (h^{\gamma'_j}(V[1])^{r_j}, g^{\gamma'_j})$, and $V''_j = (h^{\gamma''_j}(V[2])^{r_j}, g^{\gamma''_j})$, for some $\gamma_j, \gamma'_j, \gamma''_j \in \mathbb{Z}_q$. Then since $h = g^\rho$, $s_\rho(V) = g^\pi$, and for $j \in I_B$, $s_\rho(V_j) = g^{r_j}$, $s_\rho(V'_j) = (V[1])^{r_j}$, $s_\rho(V''_j) = (V[2])^{r_j}$ and $s_\rho((s_\rho(V'_j), s_\rho(V''_j))) = g^{r_j\pi}$.

Thus

$$\begin{aligned}
\overline{y}^* &= \left(\prod_{j\in I_G} B_j[1] g^{r_j} (s_\rho(V))^{-r_j/\pi_C}\right) \left(\prod_{j\in I_B} B_j[1] s_\rho(V_j) \left(s_\rho((s_\rho(V'_j), s_\rho(V''_j)))\right)^{-1/\pi_C}\right) \\
&= \left(\prod_{j\in I_G} B_j[1] g^{r_j} \left(g^{-r_j\pi(\pi_C)^{-1}}\right)\right) \left(\prod_{j\in I_B} B_j[1] g^{r_j} \left(g^{-r_j\pi(\pi_C)^{-1}}\right)\right) \\
&= \prod_{j\in I} B_j[1] g^{r_j(1-\pi(\pi_C)^{-1})} \\
&= \prod_{j\in I} \left(y^{(\beta+\alpha\pi)r_j + r'_j} g^{r_j(\pi(\pi_C)^{-1}-1)}\right) g^{r_j(1-\pi(\pi_C)^{-1})} \\
&= \prod_{j\in I} y^{(\beta+\alpha\pi)r_j + r'_j} \\
&= y^{(\beta+\alpha\pi)(\sum_{j\in I} r_j) + (\sum_{j\in I} r'_j)} \\
&= \overline{g}^x.
\end{aligned}$$

Thus as long as Fraud does not occur, the simulation of the SERVER ACTION 4 query is perfectly indistinguishable from the real SERVER ACTION 4 query in either P_5 or P_6.

Also as long as Fraud does not occur, when (X, Y, Z) is a DH triple, then in both computations of

Z^* above,

$$\begin{aligned} Z^* &= (s_\rho(R_i))^{\lambda_{i,J_i}(\lambda_{i,I})^{-1}} \left(\prod_{j\in J} X^{\lambda_{j,J_i}x_j}\right) \\ &= (X^{\lambda_{i,I}x_i})^{\lambda_{i,J_i}(\lambda_{i,I})^{-1}} \left(\prod_{j\in J} X^{\lambda_{j,J_i}x_j}\right) \\ &= X^{\lambda_{i,J_i}x_i} \left(\prod_{j\in J} X^{\lambda_{j,J_i}x_j}\right) \\ &= \prod_{j\in J_i} X^{\lambda_{j,J_i}x_j} \\ &= X^x \end{aligned}$$

where $y_i = g^{x_i}$ and $y = g^x$, where x and x_i are not necessarily known to D. Thus $Z^* = \mathsf{DH}(X, Y)$.

When (X, Y, Z) is drawn from the set of DH triples, the simulation is statistically indistinguishable from P_5 until E occurs, the statistical difference coming from the probability that Fraud occurs. When E occurs, D outputs 1. When (X, Y, Z) is drawn from the set of random triples, the simulation is statistically indistinguishable from P_5, the statistical difference coming from (1) the probability that Fraud occurs, and (2) the probability that Fraud does not occur but D halts and outputs 1, which is at most $\frac{kn_{\text{in}}}{q}$, since Z is random.

Let t' be the running time for D, and note that $t' = O(t + (n_{\text{ro}} + kn_{\text{in}} + k^2 n_{\text{ex}})t_{exp})$. The advantage of D is

$$\begin{aligned} \mathsf{Adv}^{\text{DDH}}_{G_q}(D) &= \Pr[D \text{ outputs } 1|\text{DH triple}] - \Pr[D \text{ outputs } 1|\text{random triple}] \\ &\geq \epsilon - \frac{4kn_{\text{in}}(n_{\text{ro}}+1)}{q} - \frac{kn_{\text{in}}}{q} - \frac{4kn_{\text{in}}(n_{\text{ro}}+1)}{q}. \end{aligned}$$

The claim follows from the fact that $\mathsf{Adv}^{\text{DDH}}_{G_q}(D) \leq \mathsf{Adv}^{\text{DDH}}_{G_q}(t')$. ∎

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

1. In initialization, for each $C \in \mathit{Clients}$, $E_C \xleftarrow{R} G_q \times G_q$.
2. In a Server Action 3 query to an uncompromised server S_i that uses a value B not used in any Client Action 2 query, $B_i \xleftarrow{R} G_q \times G_q$.
3. In a Server Action 4 query to an uncompromised server S_i that uses a value B not used in any Client Action 2 query, and that uses a τ' value, compute $\overline{C}_i$ as follows. If there is a $(\overline{y}^*, \tau')$ pair recorded for this τ', use that $\overline{y}^*$, else choose $\overline{y}^* \xleftarrow{R} G_q$, and record $(\overline{y}^*, \tau')$. Then $\overline{C}_i \leftarrow \left(\overline{y}^* \left(\prod_{j\in J} \overline{g}^{\lambda_{j,J_i}x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{i,J_i}}$. (The intuition behind this is that with a random $\overline{y}^*$, $\mathcal{A}$ will fail with high probability, as it should.)

Claim 5.9 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\text{ro}} + kn_{\text{in}} + k^2 n_{\text{ex}})t_{exp})$ such that

$$\mathsf{Adv}^{\text{dake}}_{P_6}(\mathcal{A}) \leq \mathsf{Adv}^{\text{dake}}_{P_7}(\mathcal{A}) + 2\mathsf{Adv}^{\text{DDH}}_{G_q}(t') + \frac{O(kn_{\text{in}}n_{\text{ro}} + 1)}{q}.$$

Proof: Assume $\mathsf{Adv}^{\text{dake}}_{P_6}(\mathcal{A}) = \mathsf{Adv}^{\text{dake}}_{P_7}(\mathcal{A}) + 2\epsilon$, and thus by Fact 2.2, $\Pr(\mathsf{Succ}^{\text{dake}}_{P_6}(\mathcal{A})) = \Pr(\mathsf{Succ}^{\text{dake}}_{P_7}(\mathcal{A})) + \epsilon$.

For the intuition behind this proof, consider if P_7 was simply P_6 with E_C computed randomly. (Indeed this would remove the last password dependency from the protocol, which was our original goal.) Then we would try to show that if the adversary can distinguish the two protocols (P_6 and P_7), then the adversary can break the (semantic security of) encryption E_C, and hence DDH. This could be proven using a straightforward reduction from DDH except for the fact that we need to simulate $\overline{C}_i$ values from uncompromised servers who receive B values generated by the adversary. The method in the previous proof cannot be used directly, since it depends explicitly on E_C being valid. To solve this problem, in P_7 we also randomize the B_i values from uncompromised servers, and we compute $\overline{C}_i$ values as in the method in the previous proof, but using a randomly chosen $\overline{y}^*$ value. Our reduction from DDH will then be set up so that the B_i values from uncompromised servers are affected in the same way as E_C, i.e., they are computed honestly for valid Diffie-Hellman triples, but are random for random triples. Similarly, the $\overline{y}^*$ values are computed honestly (i.e., $\overline{y}^* = \overline{g}^x$ as in the previous proof) for valid Diffie-Hellman triples, but are random for random triples (and in particular, are not dependent on the correct form of E_C). Putting this all together, we show that we can perform the simulation in the reduction from DDH, and thus we show that if the adversary can distinguish the two protocols, then DDH can be broken.

Formally, we construct an algorithm D that attempts 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_7 for $\mathcal{A}$ with these changes:

1. In the initialization procedure, $y \leftarrow Y$. For each $i \in J$, $x_i \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and for each uncompromised server S_i, $y_i \leftarrow \left(y \left(\prod_{j \in J} g^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{1/\lambda_{i,J_i}}$.

2. In the initialization of a client C, $E_C \leftarrow (Z^{\mu_1} Y^{\mu_2} g^{(\pi_C)^{-1}}, X^{\mu_1} g^{\mu_2})$ for $\mu_1, \mu_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$.

3. In a Server Action 3 query to an uncompromised server S_i that does not use a value B computed in a Client Action 2 query to a client, D computes $B_i \leftarrow B^{r_i} \times (y, g)^{r'_i} \times (Z, X)^{r''_i}$, for $r_i, r'_i, r''_i \stackrel{R}{\leftarrow} \mathbb{Z}_q$.

4. In a Server Action 4 query to an uncompromised server S_i that does not use a value B produced in a Client Action 2 query to a client, do the following:

   (a) Let $I_B \subseteq I$ be the set of indices where for $j \in I_B$, the tuple (B_j, V_j, V'_j, V''_j) was not produced by server S_j, and let $I_G = I \setminus I_B$. Use the $B, V, \{j, B_j, V_j, V'_j, V''_j\}_{j \in I}$ values

known to S_i to perform the following computation, where $s_\rho(V) = V[1]/(V[2])^\rho$.

$$\overline{y}^* \leftarrow \left(\prod_{j\in I_G} B_j[1] g^{r_j} (s_\rho(V))^{-r_j/\pi_C}\right)\left(\prod_{j\in I_B} B_j[1] s_\rho(V_j)\left(s_\rho((s_\rho(V'_j), s_\rho(V''_j)))\right)^{-1/\pi_C}\right).$$

(b) Set $\overline{C}_i \leftarrow \left(\overline{y}^* \left(\prod_{j\in J} \overline{g}^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{i,J_i}}$.

5. If $\mathcal{A}$ succeeds against this simulation, D outputs 1 and halts, else D outputs 0 and halts.

Similar to the proof of Claim 5.8, if Fraud does not occur, when (X, Y, Z) is a DH triple, $\overline{y}^* = \overline{g}^x$, where $y = g^x$. Thus, when (X, Y, Z) is drawn from the set of DH triples, the simulation is statistically indistinguishable from P_6, the statistical difference coming from the probability that Fraud occurs. When (X, Y, Z) is drawn from the set of random triples, the simulation is statistically indistinguishable from P_7, the statistical difference coming from the $\frac{1}{q}$ probability that (X, Y, Z) is actually a DH triple, and the probability that Fraud occurs.

To see this, consider a non-DH triple (X, Y, Z), and assume Fraud does not occur. Then obviously the distribution of E_C is the same. Consider a (τ, B, V) tuple sent by $\mathcal{A}$ to a set of servers I. The distribution of the B_i value produced by an uncompromised server S_i will be the same as in P_7, i.e., in both B_i will be random in $G_q \times G_q$.

We are left to show that the distribution of $\overline{y}^*$ values produced in P_7 and the simulation are the same. In particular, we need to show that the $\overline{y}^*$ value produced in the simulation for a given τ' is the same for each uncompromised server using τ', is randomly distributed and, if a corresponding $\overline{C}_i$ value is revealed, is independent from any $\overline{y}^*$ value produced in the simulation for any other τ' at any uncompromised server.[8] We argue this as follows. In the simulation, in any Server Action 4 query to an uncompromised server using a given τ', the $\overline{y}^*$ value will be fixed. This is because (1) τ' determines τ, and no τ is ever duplicated (see P_1), implying that each r_j value for $j \in I_G$ can be determined by examining the Server Action 3 query to the instance of S_j corresponding to τ, (2) τ' determines V, (3) τ' determines each B_j and V_j, and for $j \in I_B$, since Fraud does not occur (specifically, for an $\mathcal{R}$-SS-NIZKP not produced in a Server Action 3 query to server S_j), this fixes $s_\rho(V_j)\left(s_\rho((s_\rho(V'_j), s_\rho(V''_j)))\right)^{-1/\pi_C}$. To see that $\overline{y}^*$ is random (and in particular, independent of $\overline{y}$), simply notice that the $\overline{y}^*$ value includes a random factor. Specifically, for an uncompromised server S_i, it includes the factor $g^{r_i}(s_\rho(V))^{-(\pi_C)^{-1} r_i}$, where $s_\rho(V) \neq g^{\pi_C}$ (see P_5) which implies this factor is of the form $(g')^{r_i}$, where $g' \in G_q$ and $g' \neq 1$. But one can see that for a given B_i, all values of r_i are equally likely (since $\log_X Z \neq \log_g y$), so $(g')^{r_i}$ is random.

Note that the above argument implies that the $\overline{y}^*$ value computed by S_i using τ' is independent from any other $\overline{y}^*$ produced by S_i, since the random factor r_i would be independent. Now we claim that if any $\overline{C}_i$ is revealed for a given τ' (say τ'_1) the $\overline{y}^*$ value computed by S_i will be independent of any $\overline{y}^*$ computed at any other uncompromised server S_j for a different τ' (say τ'_2). This is basically because the adversary could not produce a valid $\mathcal{S}$-SS-NIZKP Γ_j for an uncompromised server S_j,

[8] Note that $\overline{y}^*$ is only used in computing $\overline{C}_i$, and if $\overline{C}_i$ is not revealed, then the distribution of $\overline{y}^*$ is irrelevant to the behavior of the adversary.

and a valid Γ_j for τ_1' from every other server is required for S_i to reveal its $\overline{C}_i$ value. So if $\overline{C}_i$ is revealed, then all uncompromised servers in I must have run SERVER ACTION 4 using τ_1'. The claim follows by noting that a $\overline{y}^*$ value computed at an uncompromised server S_j for $\tau_2' \neq \tau_1'$ would be independent from the $\overline{y}^*$ value computed at S_j using τ_1', by the original argument about the independence of $\overline{y}^*$ values computed at the same server.

Let t' be the running time for D, and note that $t' = O(t + (n_{\text{ro}} + kn_{\text{in}} + k^2 n_{\text{ex}})t_{exp})$. The advantage of D is

$$\begin{aligned}
\mathsf{Adv}^{\text{DDH}}_{G_q}(D) &= \Pr[D \text{ outputs } 1 | \text{DH triple}] - \Pr[D \text{ outputs } 1 | \text{random triple}] \\
&\geq \Pr\left[\mathsf{Succ}^{\text{dake}}_{P_6}(\mathcal{A})\right] - \frac{4kn_{\text{in}}(n_{\text{ro}}+1)}{q} - \Pr\left[\mathsf{Succ}^{\text{dake}}_{P_7}(\mathcal{A})\right] - \frac{1}{q} - \frac{4kn_{\text{in}}(n_{\text{ro}}+1)}{q} \\
&= \epsilon - \frac{1}{q} - \frac{8kn_{\text{in}}(n_{\text{ro}}+1)}{q}.
\end{aligned}$$

The claim follows from the fact that $\mathsf{Adv}^{\text{DDH}}_{G_q}(D) \leq \mathsf{Adv}^{\text{DDH}}_{G_q}(t')$. ▌

Now we show that in P_7, unless Fraud occurs, an uncompromised server will accept exactly those tuples (τ, B, V) generated by a client (except with probability $\frac{n_{\text{in}}}{q}$), and thus P_7 gives no information about π_C except for testing for it (as defined in P_5) in a SERVER ACTION 3 query that does not come from a client.

Assume Fraud does not occur. Then say (τ, B, V) is generated by a client. An uncompromised server S_i accepts by testing that $\prod_{i \in I} \overline{C}_i = \overline{y}$. But since Fraud does not occur, every $\overline{C}_i$ produced by $\mathcal{A}$ (with a valid Γ_i') satisfies $\overline{C}_i = \overline{g}^{\lambda_{i,I} x_i}$, and by P_6, it comes from a compromised server. Furthermore, by P_4, every $\overline{C}_i$ produced by an uncompromised server satisfies $\overline{C}_i = \left(\overline{y}\left(\prod_{j \in J} \overline{g}^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{i,J_i}}$. As in the proof of Claim 5.6, $\overline{C}_i = \overline{g}^{\lambda_{i,I} x_i}$, where $\{x_i\}_{i \in \{1,\ldots,n\}}$ is the (k,n)-threshold sharing of the discrete log of $\overline{y}$ over base $\overline{g}$, with the fixed values $\{x_j\}_{j \in J}$, where J is the set of $k-1$ compromised servers. Note that we do not necessarily know x_i for $i \notin J$.

Now let $I_B = I \cap J$ and $I_G = I \setminus I_B$. Then

$$\begin{aligned}
\prod_{i \in I} \overline{C}_i &= \prod_{i \in I_B} \overline{g}^{\lambda_{i,I} x_i} \prod_{i \in I_G} \overline{g}^{\lambda_{i,I} x_i} \\
&= \prod_{i \in I} \overline{g}^{\lambda_{i,I} x_i} \\
&= \overline{y}.
\end{aligned}$$

Thus the tuple (τ, B, V) would be accepted.

Again assume Fraud does not occur. Then say (τ, B, V) is generated by $\mathcal{A}$. Again an uncompromised server S_i accepts by testing that $\prod_{i \in I} \overline{C}_i = \overline{y}$. Also every $\overline{C}_i$ produced by $\mathcal{A}$ (with a valid Γ_i') satisfies $\overline{C}_i = \overline{g}^{\lambda_{i,I} x_i}$ and comes from a compromised server. But now every $\overline{C}_i$ produced by an uncompromised server satisfies $\overline{C}_i = \left(\overline{y}^*\left(\prod_{j \in J} \overline{g}^{\lambda_{j,J_i} x_j}\right)^{-1}\right)^{\lambda_{i,I}/\lambda_{j,J_i}}$, for a random $\overline{y}^*$ (associated with the τ' value used by these servers), independent of $\overline{y}$. Let $\{\overline{x}_i\}_{i \in \{1,\ldots,n\}}$ be a (k,n)-threshold sharing of the discrete log of $\overline{y}^*$ over base $\overline{g}$, such that $\overline{x}_i = x_i$ for $i \in J$, where J is the set of $k-1$ compromised servers. Note that we do not necessarily know $\overline{x}_i$ for $i \notin J$.

Now let $I_B = I \cap J$ and $I_G = I \setminus I_B$. Then

$$\begin{aligned}
\prod_{i\in I} \overline{C}_i &= \prod_{i\in I_B} \overline{g}^{\lambda_{i,I} x_i} \prod_{i\in I_G} \left(\overline{y}^* \left(\prod_{j\in J} \overline{g}^{\lambda_{j,J_i} x_j} \right)^{-1} \right)^{\lambda_{i,I}/\lambda_{i,J_i}} \\
&= \prod_{i\in I_B} \overline{g}^{\lambda_{i,I} x_i} \prod_{i\in I_G} \left(\overline{g}^{\lambda_{i,J_i} \overline{x}_i} \right)^{\lambda_{i,I}/\lambda_{i,J_i}} \\
&= \prod_{i\in I_B} \overline{g}^{\lambda_{i,I} x_i} \prod_{i\in I_G} \overline{g}^{\lambda_{i,I} \overline{x}_i} \\
&= \prod_{i\in I_B} \overline{g}^{\lambda_{i,I} \overline{x}_i} \prod_{i\in I_G} \overline{g}^{\lambda_{i,I} \overline{x}_i} \\
&= \prod_{i\in I} \overline{g}^{\lambda_{i,I} \overline{x}_i} \\
&= \overline{y}^*.
\end{aligned}$$

Thus the tuple (τ, B, V) would be rejected unless $\overline{y}^* = \overline{y}$, which occurs with probability $\frac{1}{q}$.

Let E be the event that Fraud occurs, $\mathcal{A}$ succeeds in a password guess (as defined in P_5), or $\overline{y}^* = \overline{y}$ for some random $\overline{y}^*$ as above.

$$\begin{aligned}
\Pr\left[\mathsf{Succ}^{\mathrm{dake}}_{P_7}(\mathcal{A})\right] &\leq \Pr[E] + \Pr\left[\mathsf{Succ}^{\mathrm{dake}}_{P_7}(\mathcal{A}) | \neg E\right](1 - \Pr[E]) \\
&\leq \Pr[E] + \frac{1}{2}(1 - \Pr[E]) \\
&= \frac{1}{2} + \frac{\Pr(E)}{2} \\
&\leq \frac{1}{2} + \frac{n_{\mathrm{in}}}{2N} + \frac{n_{\mathrm{in}}}{2q} + \frac{2kn_{\mathrm{in}}(n_{\mathrm{ro}}+1)}{q}.
\end{aligned}$$

which implies

$$\mathsf{Adv}^{\mathrm{dake}}_{P_7}(\mathcal{A}) \leq \frac{n_{\mathrm{in}}}{N} + \frac{n_{\mathrm{in}}}{q} + \frac{4kn_{\mathrm{in}}(n_{\mathrm{ro}}+1)}{q}.$$

The theorem follows from this fact, along with claims 5.2 through 5.9. ▌

References

[1] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000* (LNCS 1807), pp. 139–155, 2000.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1st *ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.

[3] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93* (LNCS 773), pp. 232–249, 1993.

[4] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *27th ACM Symposium on the Theory of Computing*, pp. 57–66, 1995.

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

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

[7] M. Blum, P. Feldman and S. Micali. Non-interactive zero-knowledge and its applications. In *20th ACM Symposium on the Theory of Computing*, pp. 103–112, 1988.

[8] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.

[9] C. Boyd. Digital multisignatures. In H. J. Beker and F. C. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, 1986.

[10] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authentication and key exchange using Diffie-Hellman. In *EUROCRYPT 2000* (LNCS 1807), pp. 156–171, 2000.

[11] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *30th ACM Symposium on the Theory of Computing*, pp. 209–218, 1998.

[12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party computation. In *34th ACM Symposium on the Theory of Computing*, 2002.

[13] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94* (LNCS 839), pages 174–187, 1994.

[14] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO '89* (LNCS 435), pages 307–315, 1989.

[15] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO 2001* (LNCS 2139), pp. 566–598, 2001.

[16] T. Dierks and C. Allen. The TLS protocol, version 1.0, IETF RFC 2246, January 1999.

[17] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.

[18] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. In *EUROCRYPT '03* (LNCS 2656), pp. 507–523, 2003. Final version available: `http://www.marioland.it/papers/tpassword.pdf`.

[19] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.

[20] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th IEEE Symp. on Foundations of Computer Science*, pp. 427-437, 1987

[21] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO '86*, pp. 186–194, 1986.

[22] W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the* 5^{th} *IEEE International Workshop on Enterprise Security*, 2000.

[23] Y. Frankel, P. MacKenzie, and M. Yung. Adaptively-secure distributed threshold public key systems. In *European Symposium on Algorithms* (LNCS 1643), pp. 4–27, 1999.

[24] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. The (in)security of distributed key generation in dlog-based cryptosystems. In *EUROCRYPT '99* (LNCS 1592), pp. 295–310, 1999.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *EUROCRYPT '96* (LNCS 1070), pages 354–371, 1996.

[26] O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *CRYPTO 2001* (LNCS 2139), pp. 408–432, 2001.

[27] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[28] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Transactions on Information and Systems Security*, 2(3): 230–268, 1999.

[29] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public-key and signature schemes. In *3rd ACM Conference on Computer and Communications Security*, pp. 100–110, 1996.

[30] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352. Springer-Verlag, 27–31 Aug. 1995.

[31] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.

[32] D. Jablon. Password authentication using multiple servers. In *RSA Conference 2001, Cryptographers' Track* (LNCS 2020), pp. 344–360, 2001.

[33] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001* (LNCS 2045), pp. 475–494, 2001.

[34] P. MacKenzie, S. Patel, and R. Swaminathan. Password authenticated key exchange based on RSA. In *ASIACRYPT 2000*, (LNCS 1976), pp. 599–613, 2000.

[35] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM Symposium on the Theory of Computing*, pp. 427–437, 1990.

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

[37] SSH communications security. `http://www.ssh.fi`, 2001.

[38] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pp. 97–111, 1998.

A Non-interactive zero-knowledge proofs

Our protocols employ a variety of non-interactive zero-knowledge (NIZK) proofs [7]. Here we define their security under the random oracle assumption. Our definition for NIZK proofs is based on [15], which in turn is based on [36]. In particular, we show that our NIZK proofs satisfy simulation-soundness, which turns out to be necessary for the security of our main protocol. Our proof constructions in Sections A.1 through A.4 are based on standard techniques from Cramer et al. [13] and Fiat and Shamir [21].

For a relation R, let $L_R = \{w : (w, v) \in R\}$ be the *language* defined by the relation. For any NP language L, note that there is a natural *witness relation* R containing pairs (w, v) where v is the witness for the membership of w in L, and that $L_R = L$. Recall that κ is a security parameter. Recall Ω is the set of all hash functions. In this section we will assume all $H \in \Omega$ have a range of $\{0,1\}^\kappa$.[9]

- **Zero-knowledge proofs**: A simulation-sound non-interactive zero-knowledge proof system (SS-NIZKP system) for an NP language L, with witness relation R, is a tuple $(\mathcal{P}, \mathcal{V}, \mathsf{Sim})$, where $\mathcal{P}$ is a probabilistic polynomial-time algorithm, $\mathcal{V}$ is a deterministic polynomial-time algorithm, and Sim is a probabilistic polynomial-time protocol for answering both hash queries and prove queries,[10] denoted by $\mathsf{Simhash}$ and $\mathsf{Simprove}$, respectively, satisfying:

  1. Completeness: For all $(w, v) \in R$, for all $H \in \Omega$, $\mathcal{V}^H(w, \mathcal{P}^H(w, v))$ returns TRUE.
  2. Simulation-soundness: There is a negligible function $\textsc{Serr}(\kappa, n_{\text{ro}}, n_{\text{pr}})$ (*soundness error*) such that for all non-uniform probabilistic polynomial-time adversaries $\mathcal{A}$ that make at most n_{ro} hash queries and n_{pr} prove queries, $\Pr[\mathsf{Expt}^{\mathsf{Sim}}_{\mathcal{A}}(\kappa)] \leq \textsc{Serr}(\kappa, n_{\text{ro}}, n_{\text{pr}})$, where experiment $\mathsf{Expt}^{\mathsf{Sim}}_{\mathcal{A}}(\kappa)$ is defined as follows:

     $\mathsf{Expt}^{\mathsf{Sim}}_{\mathcal{A}}(\kappa)$:
     $(w, \sigma) \leftarrow \mathcal{A}^{\mathsf{Simhash},\mathsf{Simprove}(\cdot)}(1^\kappa)$
     Let Q be the list of proofs[11] given by $\mathsf{Simprove}$
     Return TRUE iff $(\sigma \notin Q$ **and** $w \notin L$ **and** $\mathcal{V}^{\mathsf{Simhash}}(w, \sigma) = \text{TRUE})$

     If this experiment returns TRUE for a certain σ, we call σ a *fraudulent proof.*

[9] For our proofs we will actually be working in the group G_q and we will assume the range of all $H \in \Omega$ is $\mathbb{Z}_q$. As discussed previously, this assumption is justified in [2].

[10] Prove queries take a string w as input and produce a proof that there is a v such that $(w, v) \in R$. In general, one cannot guarantee that such a query could be answered. However, if the hash function is also being simulated, then it will be possible to answer these queries, at least in our constructions. While this is not the most general way to define SS-NIZKP, it is sufficient for our purposes.

3. (Unbounded) Zero-knowledge: There is a negligible function $\textsc{Simerr}(\kappa, n_{\rm ro}, n_{\rm pr})$ (*simulation error*) such that $\max_{\mathcal{A}} |\Pr[\mathsf{Expt}_{\mathcal{A}}(\kappa) = 1] - \Pr[\mathsf{Expt}'_{\mathcal{A}}(\kappa) = 1]| \leq \textsc{Simerr}(\kappa, n_{\rm ro}, n_{\rm pr})$, where the maximum is over all non-uniform probabilistic polynomial-time adversaries $\mathcal{A}$ that make at most $n_{\rm ro}$ hash queries and $n_{\rm pr}$ prove queries, and where experiments $\mathsf{Expt}_{\mathcal{A}}(\kappa)$ and $\mathsf{Expt}'_{\mathcal{A}}(\kappa)$ are defined as follows:

$\mathsf{Expt}_{\mathcal{A}}(\kappa)$:	$\mathsf{Expt}'_{\mathcal{A}}(\kappa)$:
$H \xleftarrow{R} \Omega$	$\mathcal{A}^{\mathsf{Simhash},\mathsf{Sim}'(\cdot,\cdot)}(1^\kappa)$
$\mathcal{A}^{H,\mathcal{P}^H(\cdot,\cdot)}(1^\kappa)$	

where $\mathsf{Sim}'(w, v) \stackrel{\rm def}{=} \mathsf{Simprove}(w)$ for $(w, v) \in R$. (If $(w, v) \notin R$, we may assume that both $\mathcal{P}^H(w, v)$ and $\mathsf{Sim}'(w, v)$ abort.)

Remark A.1 Using the zero-knowledge condition, one can see that $\mathsf{Simhash}$ must be computationally indistinguishable from $H \xleftarrow{R} \Omega$. In our proofs, $\mathsf{Simhash}$ will be perfectly indistinguishable from $H \xleftarrow{R} \Omega$.

Remark A.2 The adversary $\mathcal{A}$ used in experiment $\mathsf{Expt}^{\mathsf{Sim}}_{\mathcal{A}}(\kappa)$ defined for simulation soundness may query $\mathsf{Simprove}$ with strings $w \notin L$. For these strings, $\mathsf{Simprove}$ will return a proof σ such that $\mathcal{V}^{\mathsf{Simhash}}(w, \sigma)$ returns TRUE. However, by the definition of simulation-soundness, these proofs will not help $\mathcal{A}$ produce any new valid proofs for any $w \notin L$.

A.1 $\mathcal{Q}$ details

Here we show how to implement the SS-NIZKP $\mathcal{Q} = (\mathsf{Prove}_{\Phi_\mathcal{Q}}, \mathsf{Verify}_{\Phi_\mathcal{Q}}, \mathsf{Sim}_{\Phi_\mathcal{Q}})$ over the language defined by the predicate $\Phi_\mathcal{Q}$, and we prove its security.[12] As discussed above, when this SS-NIZKP is used in protocol P, we let H be H_3.

Recall that

$$\Phi_\mathcal{Q}(\tau, E_C, B, V) \stackrel{\rm def}{=} \exists \beta, \pi, \gamma : \left(B = \left(y^\beta, g^\beta\right) \times (E_C)^\pi \times (g^{-1}, 1)\right) \textbf{ and } (V = (h^\gamma g^\pi, g^\gamma)) .$$

$\mathsf{Prove}_{\Phi_\mathcal{Q}}$, $\mathsf{Verify}_{\Phi_\mathcal{Q}}$, and $\mathsf{Sim}_{\Phi_\mathcal{Q}}$ are implemented in Figures 4, 5, and 6, respectively. Note that $\mathsf{Sim}_{\Phi_\mathcal{Q}}$ uses the standard technique of "backpatching" random oracle queries.

In the following, we include the subscript $\mathcal{Q}$ on the SERR and SIMERR functions, and remove κ from the parameters, since this is already implicit–$\mathcal{Q}$ is defined over the group G_q where $|q| = \kappa$.

Lemma A.3 $\mathcal{Q} = (\mathsf{Prove}_{\Phi_\mathcal{Q}}, \mathsf{Verify}_{\Phi_\mathcal{Q}}, \mathsf{Sim}_{\Phi_\mathcal{Q}})$ is an SS-NIZKP with $\textsc{Serr}_\mathcal{Q}(n_{\rm ro}, n_{\rm pr}) \leq \frac{n_{\rm ro}+1}{q}$, and $\textsc{Simerr}_\mathcal{Q}(n_{\rm ro}, n_{\rm pr}) \leq \frac{n_{\rm pr}(n_{\rm ro}+n_{\rm pr})}{q^3}$.

Proof: *Completeness:* Straightforward.

Simulation soundness: Consider an adversary $\mathcal{A}$ that makes $n_{\rm ro}$ hash queries and $n_{\rm pr}$ $\mathsf{Prove}_{\Phi_\mathcal{Q}}$ queries.

[12] To be completely formal, the predicate $\Phi_\mathcal{Q}$ should also have parameters (G_q, g, y, h), but we choose to be slightly less formal for readability.

$\mu_1, \mu_2, \nu \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$B' \leftarrow (y^{\mu_1}, g^{\mu_1}) \times (E_C)^{\mu_2}$
$V' \leftarrow (h^{\nu} g^{\mu_2}, g^{\nu})$
$e \leftarrow H(\tau, E_C, B, V, B', V')$
$z_1 \leftarrow \beta e + \mu_1 \bmod q$
$z_2 \leftarrow \pi e + \mu_2 \bmod q$
$z_3 \leftarrow \gamma e + \nu \bmod q$

$\sigma \leftarrow (e, z_1, z_2, z_3)$
Return σ

Figure 4: $\mathsf{Prove}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (\beta, \pi, \gamma))$

$B' \leftarrow (y^{z_1}, g^{z_1}) \times (E_C)^{z_2} \times (B \times (g, 1))^{-e}$
$V' \leftarrow (h^{z_3} g^{z_2}, g^{z_3}) \times V^{-e}$

Return TRUE if $e = H(\tau, E_C, B, V, B', V')$

Figure 5: $\mathsf{Verify}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (e, z_1, z_2, z_3))$

$e \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$z_1, z_2, z_3 \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$B' \leftarrow (y^{z_1}, g^{z_1}) \times (E_C)^{z_2} \times (B \times (g, 1))^{-e}$
$V' \leftarrow (h^{z_3} g^{z_2}, g^{z_3}) \times V^{-e}$

$H(\tau, E_C, B, V, B', V') \leftarrow e$

$\sigma \leftarrow (e, z_1, z_2, z_3)$
Return σ

Figure 6: $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}(\tau, E_C, B, V)$: the protocol aborts if backpatching causes the hash function to be inconsistent with a previous hash query. $\mathsf{Simhash}_{\Phi_{\mathcal{Q}}}$ behaves like a normal random oracle, except that it maintains consistency with the backpatching.

Say $\mathcal{A}$ makes a query $(\tau, E_C, B, V, B', V')$ to the random oracle, where (τ, E_C, B, V) does not satisfy $\Phi_{\mathcal{Q}}$ and $(\tau, E_C, B, V, B', V')$ was not backpatched in a $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$ query. Say $B = \left(y^{\beta_1}, g^{\beta_2}\right) \times (E_C)^{\pi} \times (g^{-1}, 1)$, $V = (h^{\gamma} g^{\pi}, g^{\gamma})$, $B' = (y^{\mu_1}, g^{\mu_2}) \times (E_C)^{\mu_3}$, and $V' = (h^{\nu} g^{\mu_3}, g^{\nu})$, for some values $\gamma, \beta_1, \beta_2, \pi, \nu, \mu_1, \mu_2, \mu_3 \in \mathbb{Z}_q$. Let $c_1, c_2, c_3, c_4 \in \mathbb{Z}_q$ be such that $E_C[1] = g^{c_1}$, $E_C[2] = g^{c_2}$, $h = g^{c_3}$, and $y = g^{c_4}$. Then to have a proof $\sigma = (e, z_1, z_2, z_3)$ using this random oracle query such that $\mathsf{Verify}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), \sigma) = \text{TRUE}$, it must be that

$$\begin{aligned}
(c_4\beta_1 + c_1\pi)e + c_4\mu_1 + c_1\mu_3 &= c_4 z_1 + c_1 z_2 \bmod q, \\
(\beta_2 + c_2\pi)e + \mu_2 + c_2\mu_3 &= z_1 + c_2 z_2 \bmod q, \\
(c_3\gamma + \pi)e + c_3\nu + \mu_3 &= c_3 z_3 + z_2 \bmod q, \text{ and} \\
\gamma e + \nu &= z_3 \bmod q.
\end{aligned}$$

However, if (τ, E_C, B, V) does not satisfy $\Phi_{\mathcal{Q}}$, then $\beta_1 \neq \beta_2$, implying $e = \frac{\mu_1 - \mu_2}{\beta_2 - \beta_1} \bmod q$, and thus there is at most one possible hash output that would allow such a proof. Hence there is a $1/q$ probability of this query allowing such a proof.

Note that if $\mathcal{A}$ makes a $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$ query that results in a backpatching on $(\tau, E_C, B, V, B', V')$, then this random oracle query could not be used in a fraudulent proof by $\mathcal{A}$.

Then the probability that $\mathcal{A}$ succeeds is at most $\frac{n_{\text{ro}}+1}{q}$, with the extra $\frac{1}{q}$ probability from $\mathcal{A}$ guessing the output of the random oracle without actually querying it. Thus $\textsc{Serr}_{\mathcal{Q}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$.

Zero-knowledge: The distributions of $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}(\tau, E_C, B, V)$ and $\mathsf{Prove}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (\beta, \pi, \gamma))$ are exactly the same for any $((\tau, E_C, B, V), (\beta, \pi, \gamma))$ in the relation defined by $\Phi_{\mathcal{Q}}$, except for the cases when $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$ aborts. The probability that $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$ aborts is at most the probability of a collision between the new (B', V') pair and any pair that appeared in a previous random oracle query (or previous backpatching). It is easy to see that any $V' \in G_q \times G_q$ is equally likely, and that given a fixed V', and a fixed e, z_2, z_3, there are q values of $B' \in G_q \times G_q$ that are equally likely. Therefore the probability of colliding with a previous pair is at most $\frac{n_{\text{ro}}+n_{\text{pr}}}{q^3}$. Since $\mathsf{Simprove}_{\Phi_{\mathcal{Q}}}$ is queried n_{pr} times, $\textsc{Simerr}_{\mathcal{Q}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^3}$. ∎

A.2 $\mathcal{R}$ details

Here we show how to implement the SS-NIZKP $\mathcal{R} = (\mathsf{Prove}_{\Phi_{\mathcal{R}}}, \mathsf{Verify}_{\Phi_{\mathcal{R}}}, \mathsf{Sim}_{\Phi_{\mathcal{R}}})$ over the language defined by the predicate $\Phi_{\mathcal{R}}$, and we prove its security. When this SS-NIZKP is used in protocol P, we let H be H_4.

Recall that

$$\begin{aligned}
\Phi_{\mathcal{R}}(i, B, V, B_i, V_i, V_i', V_i'') \quad \stackrel{\text{def}}{=} \quad & \exists r_i, r_i', \gamma_i, \gamma_i' : B_i \leftarrow B^{r_i} \times (y, g)^{r_i'} \textbf{ and } V_i \leftarrow (h^{\gamma_i} g^{r_i}, g^{\gamma_i}) \textbf{ and} \\
& V_i' \leftarrow (h^{\gamma_i'}(V[1])^{r_i}, g^{\gamma_i'}) \textbf{ and } V_i'' \leftarrow (h^{\gamma_i''}(V[2])^{r_i}, g^{\gamma_i''}).
\end{aligned}$$

$\mathsf{Prove}_{\Phi_{\mathcal{R}}}$, $\mathsf{Verify}_{\Phi_{\mathcal{R}}}$, and $\mathsf{Sim}_{\Phi_{\mathcal{R}}}$ are implemented in Figures 7, 8, and 9, respectively. Note that $\mathsf{Sim}_{\Phi_{\mathcal{R}}}$ uses the standard technique of "backpatching" random oracle queries.

In the following, we include the subscript $\mathcal{R}$ on the Serr and Simerr functions, and remove κ from the parameters, since this is already implicit–$\mathcal{R}$ is defined over the group G_q where $|q| = \kappa$.

Lemma A.4 $\mathcal{R} = (\mathsf{Prove}_{\Phi_{\mathcal{R}}}, \mathsf{Verify}_{\Phi_{\mathcal{R}}}, \mathsf{Sim}_{\Phi_{\mathcal{R}}})$ is an SS-NIZKP with $\textsc{Serr}_{\mathcal{R}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$, and $\textsc{Simerr}_{\mathcal{R}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^5}$.

$\mu_1, \mu_2, \nu_1, \nu_2, \nu_3 \xleftarrow{R} \mathbb{Z}_q$
$\tilde{B}_i \leftarrow B^{\mu_1} \times (y^{\mu_2}, g^{\mu_2})$
$\tilde{V}_i \leftarrow (h^{\nu_1} g^{\mu_1}, g^{\nu_1})$
$\tilde{V}_i' \leftarrow (h^{\nu_2}(V[1])^{\mu_1}, g^{\nu_2})$
$\tilde{V}_i'' \leftarrow (h^{\nu_3}(V[2])^{\mu_1}, g^{\nu_3})$
$e \leftarrow H(i, B, V, B_i, V_i, V_i', V_i'', \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$
$z_1 \leftarrow r_i e + \mu_1 \bmod q$
$z_2 \leftarrow r_i' e + \mu_2 \bmod q$
$z_3 \leftarrow \gamma_i e + \nu_1 \bmod q$
$z_4 \leftarrow \gamma_i' e + \nu_2 \bmod q$
$z_5 \leftarrow \gamma_i'' e + \nu_3 \bmod q$

$\sigma \leftarrow (e, z_1, z_2, z_3, z_4, z_5)$
Return σ

Figure 7: $\mathsf{Prove}_{\Phi_{\mathcal{R}}}((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$

$\tilde{B}_i \leftarrow B^{z_1} \times (y^{z_2}, g^{z_2}) \times (B_i)^{-e}$
$\tilde{V}_i \leftarrow (h^{z_3} g^{z_1}, g^{z_3}) \times (V_i)^{-e}$
$\tilde{V}_i' \leftarrow (h^{z_4}(V[1])^{z_1}, g^{z_4}) \times (V_i')^{-e}$
$\tilde{V}_i'' \leftarrow (h^{z_5}(V[2])^{z_1}, g^{z_5}) \times (V_i'')^{-e}$

Return TRUE if $e = H(i, B, V, B_i, V_i, V_i', V_i'', \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$

Figure 8: $\mathsf{Verify}_{\Phi_{\mathcal{R}}}((i, B, V, B_i, V_i, V_i', V_i''), (e, z_1, z_2, z_3, z_4, z_5))$

$e \xleftarrow{R} \mathbb{Z}_q$
$z_1, z_2, z_3, z_4, z_5 \xleftarrow{R} \mathbb{Z}_q$
$\tilde{B}_i \leftarrow B^{z_1} \times (y^{z_2}, g^{z_2}) \times (B_i)^{-e}$
$\tilde{V}_i \leftarrow (h^{z_3} g^{z_1}, g^{z_3}) \times (V_i)^{-e}$
$\tilde{V}'_i \leftarrow (h^{z_4} (V[1])^{z_1}, g^{z_4}) \times (V'_i)^{-e}$
$\tilde{V}''_i \leftarrow (h^{z_5} (V[2])^{z_1}, g^{z_5}) \times (V''_i)^{-e}$

$H(i, B, V, B_i, V_i, V'_i, V''_i, \tilde{B}_i, \tilde{V}_i, \tilde{V}'_i, \tilde{V}''_i) \leftarrow e$

$\sigma \leftarrow (e, z_1, z_2, z_3, z_4, z_5)$
Return σ

Figure 9: $\mathsf{Simprove}_{\Phi_\mathcal{R}}(i, B, V, B_i, V_i, V'_i, V''_i)$: the protocol aborts if backpatching causes the hash function to be inconsistent with a previous hash query. $\mathsf{Simhash}_{\Phi_\mathcal{R}}$ behaves like a normal random oracle, except that it maintains consistency with the backpatching.

Proof: *Completeness:* Straightforward.

Simulation soundness: Consider an adversary $\mathcal{A}$ that makes n_{ro} hash queries and n_{pr} $\mathsf{Prove}_{\Phi_\mathcal{R}}$ queries.

Say $\mathcal{A}$ makes a query $(i, B, V, B_i, V_i, V'_i, V''_i, \tilde{B}_i, \tilde{V}_i, \tilde{V}'_i, \tilde{V}''_i)$ to the random oracle, where the tuple $(i, B, V, B_i, V_i, V'_i, V''_i)$ does not satisfy $\Phi_\mathcal{R}$ and $(i, B, V, B_i, V_i, V'_i, V''_i, \tilde{B}_i, \tilde{V}_i, \tilde{V}'_i, \tilde{V}''_i)$ was not backpatched in a $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ query. Say $B_i = B^{r_i} \times \left(y^{r'_i}, g^{r''_i}\right)$, $V_i = (h^{\gamma_1} g^{r_i}, g^{\gamma_1})$, $V'_i = \left(h^{\gamma_2} (V[1])^{\beta_1}, g^{\gamma_2}\right)$, and $V''_i = \left(h^{\gamma_3} (V[2])^{\beta_2}, g^{\gamma_3}\right)$, $\tilde{B}_i = B^{\mu_1} \times (y^{\mu_2}, g^{\mu_3})$, $\tilde{V}_i = (h^{\nu_1} g^{\mu_1}, g^{\nu_1})$, $\tilde{V}'_i = (h^{\nu_2} (V[1])^{\mu_4}, g^{\nu_2})$, and $\tilde{V}''_i = (h^{\nu_3} (V[2])^{\mu_5}, g^{\nu_3})$, for some $r_i, r'_i, r''_i, \gamma_1, \gamma_2, \gamma_3, \beta_1, \beta_2, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \nu_1, \nu_2, \nu_3 \in \mathbb{Z}_q$. Let $c_1, c_2, c_3, c_4 \in \mathbb{Z}_q$ be such that $B[1] \equiv g^{c_1}$, $B[2] \equiv g^{c_2}$, $h \equiv g^{c_3}$, $y \equiv g^{c_4}$, $V[1] \equiv g^{c_5}$, and $V[2] \equiv g^{c_6}$. Then to have a proof $\sigma = (e, z_1, z_2, z_3, z_4, z_5)$ using this random oracle query such that $\mathsf{Verify}_{\Phi_\mathcal{R}}((i, B, V, B_i, V_i, V'_i, V''_i), \sigma) = \text{TRUE}$, it must be that

$$\begin{aligned}
(c_1 r_i + c_4 r'_i)e + c_1\mu_1 + c_4\mu_2 &= c_1 z_1 + c_4 z_2 \bmod q, \\
(c_2 r_i + r''_i)e + c_2\mu_1 + \mu_3 &= c_2 z_1 + z_2 \bmod q, \\
(c_3\gamma_1 + r_i)e + c_3\nu_1 + \mu_1 &= c_3 z_3 + z_1 \bmod q, \\
\gamma_1 e + \nu_1 &= z_3 \bmod q, \\
(c_3\gamma_2 + c_5\beta_1)e + c_3\nu_2 + c_5\mu_4 &= c_3 z_4 + c_5 z_1, \\
\gamma_2 e + \nu_2 &= z_4 \bmod q, \\
(c_3\gamma_3 + c_6\beta_2)e + c_3\nu_3 + c_6\mu_5 &= c_3 z_5 + c_6 z_1 \bmod q, \text{ and} \\
\gamma_3 e + \nu_3 &= z_5 \bmod q.
\end{aligned}$$

However, if $(i, B, V, B_i, V_i, V'_i, V''_i)$ does not satisfy $\Phi_\mathcal{R}$, then either (1) $r'_i \neq r''_i$, implying $e = \frac{\mu_1 - \mu_2}{r''_i - r'_i} \bmod q$, (2) $r'_i = r''_i$ but $\beta_1 \neq r_i$, implying $e = \frac{\mu_4 - \mu_1}{r_i - \beta_1} \bmod q$, or (3) $r'_i = r''_i$ and $\beta_1 = r_i$, but

$\beta_2 \neq r_i$, implying $e = \frac{\mu_5 - \mu_1}{r_i - \beta_2} \bmod q$.[13] Thus there is at most one possible hash output that would allow such a proof. Hence there is a $1/q$ probability of this query allowing such a proof.

Note that if $\mathcal{A}$ makes a $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ query that results in backpatching the random oracle on the input $(i, B, V, B_i, V_i, V_i', V_i'', \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$, then this random oracle query could not be used in a fraudulent proof by $\mathcal{A}$.

Then the probability that $\mathcal{A}$ succeeds is at most $\frac{n_{\text{ro}}+1}{q}$, with the extra $\frac{1}{q}$ probability from $\mathcal{A}$ guessing the output of the random oracle without actually querying it. Thus $\textsc{Serr}_\mathcal{R}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$.

Zero-knowledge: The distributions of

$$\mathsf{Simprove}_{\Phi_\mathcal{R}}(i, B, V, B_i, V_i, V_i', V_i'') \text{ and}$$

$$\mathsf{Prove}_{\Phi_\mathcal{R}}((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$$

are the same for any $((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$ in the relation defined by $\Phi_\mathcal{R}$, except for the cases when $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ aborts. The probability that $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ aborts is at most the probability of a collision between the new $(B, V, \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$ tuple and any tuple that appeared in a previous random oracle query (or previous backpatching). It is easy to see that there are q^5 possible tuple values that are equally likely. Therefore the probability of colliding with a previous pair is at most $\frac{n_{\text{ro}}+n_{\text{pr}}}{q^5}$. Since $\mathsf{Simprove}_{\Phi_\mathcal{R}}$ is queried n_{pr} times, $\textsc{Simerr}_\mathcal{R}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^5}$. ∎

A.3 $\mathcal{S}$ details

Here we show how to implement the SS-NIZKP $\mathcal{S} = (\mathsf{Prove}_{\Phi_\mathcal{S}}, \mathsf{Verify}_{\Phi_\mathcal{S}}, \mathsf{Sim}_{\Phi_\mathcal{S}})$ over the language defined by the predicate $\Phi_\mathcal{S}$, and we prove its security. As discussed above, when this SS-NIZKP is used in protocol P, we let H be H_5.

Recall that

$$\Phi_\mathcal{S}(i, \tau', C_i, R_i) \stackrel{\text{def}}{=} \exists a, \gamma : \left(C_i = g^a \textbf{ and } R_i = (h^\gamma (h')^a, g^\gamma)\right).$$

$\mathsf{Prove}_{\Phi_\mathcal{S}}$, $\mathsf{Verify}_{\Phi_\mathcal{S}}$, and $\mathsf{Sim}_{\Phi_\mathcal{S}}$ are implemented in Figures 10, 11, and 12, respectively. Note that $\mathsf{Sim}_{\Phi_\mathcal{S}}$ uses the standard technique of "backpatching" random oracle queries.

In the following, we include the subscript $\mathcal{S}$ on the Serr and Simerr functions, and remove κ from the parameters, since this is already implicit–$\mathcal{S}$ is defined over the group G_q where $|q| = \kappa$.

Lemma A.5 $\mathcal{S} = (\mathsf{Prove}_{\Phi_\mathcal{S}}, \mathsf{Verify}_{\Phi_\mathcal{S}}, \mathsf{Sim}_{\Phi_\mathcal{S}})$ is an SS-NIZKP with $\textsc{Serr}_\mathcal{S}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$, and $\textsc{Simerr}_\mathcal{S}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^2}$.

Proof: *Completeness:* Straightforward.

Simulation soundness: Consider an adversary $\mathcal{A}$ that makes n_{ro} hash queries and n_{pr} $\mathsf{Prove}_{\Phi_\mathcal{S}}$ queries.

Say $\mathcal{A}$ makes a query $(i, \tau', C_i, R_i, W, R')$ to the random oracle, where (i, τ', C_i, R_i) does not satisfy $\Phi_\mathcal{S}$ and $(i, \tau', C_i, R_i, W, R')$ was not backpatched in a $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ query. Say $C_i = g^a$, $R_i =$

[13] It is also possible that no such e exists that satisfies these equations.

$\mu, \nu \xleftarrow{R} \mathbb{Z}_q$
$W \leftarrow g^{\mu}$
$R' \leftarrow (h^{\nu}(h')^{\mu}, g^{\nu})$
$e \leftarrow H(i, \tau', C_i, R_i, W, R')$
$z_1 \leftarrow ae + \mu \bmod q$
$z_2 \leftarrow \gamma e + \nu \bmod q$

$\Gamma_i \leftarrow (e, z_1, z_2)$
Return Γ_i

Figure 10: $\mathsf{Prove}_{\Phi_{\mathcal{S}}}((i, \tau', C_i, R_i), (a, \gamma))$

$(e, z_1, z_2) \leftarrow \Gamma_i$

$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$W \leftarrow g^{z_1}(C_i)^{-e}$

Verify $e = H(i, \tau', C_i, R_i, W, R')$

Figure 11: $\mathsf{Verify}_{\Phi_{\mathcal{S}}}((i, \tau', C_i, R_i), \Gamma_i)$

$e \xleftarrow{R} \mathbb{Z}_q$
$z_1, z_2 \xleftarrow{R} \mathbb{Z}_q$
$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$W \leftarrow g^{z_1}(C_i)^{-e}$

$H(i, \tau', C_i, R_i, W, R') \leftarrow e$

$\Gamma_i \leftarrow (e, z_1, z_2)$
Return Γ_i

Figure 12: $\mathsf{Simprove}_{\Phi_{\mathcal{S}}}(i, \tau', C_i, R_i)$: the protocol aborts if backpatching causes the hash function to be inconsistent with a previous hash query. $\mathsf{Simhash}_{\Phi_{\mathcal{S}}}$ behaves like a normal random oracle, except that it maintains consistency with the backpatching.

$(h^\zeta(h')^{a'}, g^\zeta)$, $W = g^\mu$, and $R' = (h^\nu(h')^{\mu'}, g^\nu)$, for some $a, a', \gamma, \mu, \mu', \nu \in \mathbb{Z}_q$. Let $c, c' \in \mathbb{Z}_q$ be such that $h = g^c$ and $h' = g^{c'}$. Then to have a proof $\sigma = (e, z_1, z_2)$ using this random oracle query such that $\mathsf{Verify}_{\Phi_\mathcal{S}}((i, \tau', C_i, R_i), \Gamma_i) = \text{TRUE}$, it must be that

$$\begin{aligned} ae + \mu &= z_1 \bmod q, \\ (c\zeta + c'a')e + (c\nu + c'\mu') &= cz_2 + c'z_1 \bmod q, \text{ and} \\ \zeta e + \nu &= z_2 \bmod q. \end{aligned}$$

However, if (i, τ', C_i, R_i) does not satisfy $\Phi_\mathcal{S}$, then $a \neq a'$, implying $e = \frac{\mu - \mu'}{a' - a} \bmod q$, and thus there is at most one possible hash output that would allow such a proof. Hence there is a $1/q$ probability of this query allowing such a proof.

Note that if $\mathcal{A}$ makes a $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ query that results in a backpatching on $(i, \tau', C_i, R_i, W, R')$, then this random oracle query could not be used in a fraudulent proof by $\mathcal{A}$.

Then the probability that $\mathcal{A}$ succeeds is at most $\frac{n_{\text{ro}}+1}{q}$, with the extra $\frac{1}{q}$ probability from $\mathcal{A}$ guessing the output of the random oracle without actually querying it. Thus $\text{SERR}_\mathcal{S}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$.

Zero-knowledge: The distributions of $\mathsf{Simprove}_{\Phi_\mathcal{S}}(i, \tau', C_i, R_i)$ and $\mathsf{Prove}_{\Phi_\mathcal{S}}((i, \tau', C_i, R_i), (a, \gamma))$ are exactly the same for any $((i, \tau', C_i, R_i), (a, \gamma))$ in the relation defined by $\Phi_\mathcal{S}$, except for the cases when $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ aborts. The probability that $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ aborts is at most the probability of a collision between the new W and R' values and any value that appeared in a previous random oracle query (or previous backpatching). It is easy to see that any $W \in G_q$ and any $R'[2] \in G_q$ is equally likely. Therefore the probability of colliding with a previous value is at most $\frac{n_{\text{ro}}+n_{\text{pr}}}{q^2}$. Since $\mathsf{Simprove}_{\Phi_\mathcal{S}}$ is queried n_{pr} times, $\text{SIMERR}_\mathcal{S}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^2}$. ∎

A.4 $\mathcal{T}$ details

Here we show how to implement the SS-NIZKP $\mathcal{T} = (\mathsf{Prove}_{\Phi_\mathcal{T}}, \mathsf{Verify}_{\Phi_\mathcal{T}}, \mathsf{Sim}_{\Phi_\mathcal{T}})$ over the language defined by the predicate $\Phi_\mathcal{T}$, and we prove its security. As discussed above, when this SS-NIZKP is used in protocol P, we let H be H_6.

Recall that

$$\Phi_\mathcal{T}(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i) \stackrel{\text{def}}{=} \exists a, \gamma : \left(\overline{C}_i = \overline{g}^a \textbf{ and } C_i = g^a \textbf{ and } R_i = (h^\gamma(h')^a, g^\gamma)\right).$$

$\mathsf{Prove}_{\Phi_\mathcal{T}}$, $\mathsf{Verify}_{\Phi_\mathcal{T}}$, and $\mathsf{Sim}_{\Phi_\mathcal{T}}$ are implemented in Figures 13, 14, and 15, respectively. Note that $\mathsf{Sim}_{\Phi_\mathcal{T}}$ uses the standard technique of "backpatching" random oracle queries.

In the following, we include the subscript $\mathcal{T}$ on the SERR and SIMERR functions, and remove κ from the parameters, since this is already implicit–$\mathcal{T}$ is defined over the group G_q where $|q| = \kappa$.

Lemma A.6 $\mathcal{T} = (\mathsf{Prove}_{\Phi_\mathcal{T}}, \mathsf{Verify}_{\Phi_\mathcal{T}}, \mathsf{Sim}_{\Phi_\mathcal{T}})$ is an SS-NIZKP with $\text{SERR}_\mathcal{T}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$, and $\text{SIMERR}_\mathcal{T}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^2}$.

Proof: *Completeness:* Straightforward.

Simulation soundness: Consider an adversary $\mathcal{A}$ that makes n_{ro} hash queries and n_{pr} $\mathsf{Prove}_{\Phi_\mathcal{T}}$ queries.

$\mu, \nu \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$\overline{W} \leftarrow \overline{g}^{\mu}$
$W \leftarrow g^{\mu}$
$R' \leftarrow (h^{\nu}(h')^{\mu}, g^{\nu})$
$e \leftarrow H(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$
$z_1 \leftarrow ae + \mu \bmod q$
$z_2 \leftarrow \gamma e + \nu \bmod q$

$\Gamma'_i \leftarrow (e, z_1, z_2)$
Return Γ'_i

Figure 13: $\mathsf{Prove}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a, \gamma))$

$(e, z_1, z_2) \leftarrow \Gamma'_i$

$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$\overline{W} \leftarrow \overline{g}^{z_1}(\overline{C}_i)^{-e}$
$W \leftarrow g^{z_1}(C_i)^{-e}$

Verify $e = H(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$

Figure 14: $\mathsf{Verify}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), \Gamma'_i)$

$e \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$z_1, z_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$
$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$\overline{W} \leftarrow \overline{g}^{z_1}(\overline{C}_i)^{-e}$
$W \leftarrow g^{z_1}(C_i)^{-e}$

$H(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R') \leftarrow e$

$\Gamma'_i \leftarrow (e, z_1, z_2)$
Return Γ'_i

Figure 15: $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$: the protocol aborts if backpatching causes the hash function to be inconsistent with a previous hash query. $\mathsf{Simhash}_{\Phi_{\mathcal{T}}}$ behaves like a normal random oracle, except that it maintains consistency with the backpatching.

Say $\mathcal{A}$ makes a query $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$ to the random oracle, where $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$ does not satisfy $\Phi_{\mathcal{T}}$ and $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, W, R')$ was not backpatched in a $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}$ query. Say $\overline{C}_i = \overline{g}^a$, $C_i = g^{a'}$, $R_i = (h^{\zeta}(h')^{a''}, g^{\zeta})$, $\overline{W} = \overline{g}^{\mu}$, $W = g^{\mu'}$, and $R' = (h^{\nu}(h')^{\mu''}, g^{\nu})$, for some $a, a', a'', \gamma, \mu, \mu', \mu'', \nu \in \mathbb{Z}_q$. Let $c, c', c'' \in \mathbb{Z}_q$ be such that $\overline{g} = g^c$, $h = g^{c'}$, and $h' = g^{c''}$. Then to have a proof $\sigma = (e, z_1, z_2)$ using this random oracle query such that $\mathsf{Verify}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), \Gamma'_i) =$ TRUE, it must be that

$$\begin{aligned} cae + c\mu &= cz_1 \bmod q, \\ a'e + \mu' &= z_1 \bmod q, \\ (c'\zeta + c''a'')e + (c'\nu + c''\mu'') &= c'z_2 + c''z_1 \bmod q, \text{ and} \\ \zeta e + \nu &= z_2 \bmod q. \end{aligned}$$

However, if $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i)$ does not satisfy $\Phi_{\mathcal{T}}$, then either $a \neq a'$, implying $e = \frac{\mu - \mu'}{a' - a} \bmod q$, or $a = a'$ but $a' \neq a''$, implying $e = \frac{\mu' - \mu''}{a'' - a'} \bmod q$.[14] Thus there is at most one possible hash output that would allow such a proof. Hence there is a $1/q$ probability of this query allowing such a proof.

Note that if $\mathcal{A}$ makes a $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}$ query that results in a backpatching on $(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$, then this random oracle query could not be used in a fraudulent proof by $\mathcal{A}$.

Then the probability that $\mathcal{A}$ succeeds is at most $\frac{n_{\text{ro}}+1}{q}$, with the extra $\frac{1}{q}$ probability from $\mathcal{A}$ guessing the output of the random oracle without actually querying it. Thus $\text{SERR}_{\mathcal{T}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{ro}}+1}{q}$.

Zero-knowledge: The distributions of

$$\mathsf{Simprove}_{\Phi_{\mathcal{T}}}(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i) \text{ and}$$

$$\mathsf{Prove}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a, \gamma))$$

are exactly the same for any $((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a, \gamma))$ in the relation defined by $\Phi_{\mathcal{T}}$, except for the cases when $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}$ aborts. The probability that $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}$ aborts is at most the probability of a collision between the new W and R' values and any value that appeared in a previous random oracle query (or previous backpatching). It is easy to see that any $W \in G_q$ and any $R'[2] \in G_q$ is equally likely. Therefore the probability of colliding with a previous value is at most $\frac{n_{\text{ro}}+n_{\text{pr}}}{q^2}$. Since $\mathsf{Simprove}_{\Phi_{\mathcal{T}}}$ is queried n_{pr} times, $\text{SIMERR}_{\mathcal{T}}(n_{\text{ro}}, n_{\text{pr}}) \leq \frac{n_{\text{pr}}(n_{\text{ro}}+n_{\text{pr}})}{q^2}$. ∎

B Formal Specification of the Protocol

Figures 16, 17, and 18 give the formal specification of the protocol, where $\mathsf{index}(U)$ returns the index of $U \in \mathit{Servers}$, and $\{M_i\}_{i \in I'}$ denotes $\langle M_{i_1}, \ldots, M_{i_\ell} \rangle$ where $I' = \{i_1, \ldots, i_\ell\}$ and $i_1 < \cdots < i_\ell$. The Initialization protocol (Figure 16) is run before any queries by the adversary. Note that in Figure 17, $C \leftarrow U$ simply denotes a renaming of U.

[14] It is also possible that no such e exists that satisfies these equations.

$\mathsf{Initialize}(G_q)$ —

Let g be the generator for G_q
$H_0, H_1, H_2, H_3, H_4, H_5 \xleftarrow{R} \Omega$
$x \xleftarrow{R} \mathbb{Z}_q$; $y \leftarrow g^x$ (the global ElGamal key pair)
$a_0 \leftarrow x$; **for** $j \in \{1, \ldots, k-1\}$ **do** $a_j \xleftarrow{R} \mathbb{Z}_q$
let $f(z) = \sum_{j=0}^{k-1} a_j z^j$ (polynomial secret sharing of x)
$h \leftarrow H_0(y)$ ($H_0(\cdot)$ returns a random element of G_q)
$h' \leftarrow H_1(y)$ ($H_1(\cdot)$ returns a random element of G_q)
for $C \in \mathit{Clients}$ **do**
    $\pi_C \xleftarrow{R} \mathit{Password}_C$; $\alpha \xleftarrow{R} \mathbb{Z}_q$; $E_C \leftarrow (y^\alpha g^{(\pi_C)^{-1}}, g^\alpha)$
for $i \in \{1, 2, \ldots, n\}$ **do**
    $x'_i \xleftarrow{R} \mathbb{Z}_q$; $y'_i \leftarrow g^{x'_i}$ (local ElGamal key pairs)
    $x_i \leftarrow f(i)$; $y_i \leftarrow g^{x_i}$ (ith (Feldman) secret/public shares of x)
    distribute to server S_i $(x_i, x'_i, \{E_C\}_{C \in \mathit{Clients}})$
Publish $(y, \{y_i\}_{i \in \{1,2,\ldots,n\}}, \{y'_i\}_{i \in \{1,2,\ldots,n\}})$
for $i \in \mathbb{N}$ and $U \in \mathit{ID}$ **do**
    $\mathit{state}^i_U \leftarrow$ READY
    $\mathit{acc}^i_U \leftarrow \mathit{term}^i_U \leftarrow \mathit{used}^i_U \leftarrow$ FALSE
    $\mathit{sid}^i_U \leftarrow \mathit{pid}^i_U \leftarrow \mathit{sk}^i_U \leftarrow \varepsilon$

Figure 16: Specification of protocol initialization

if $U \in \mathit{Clients}$ **then**
    $\mathit{sid} \leftarrow \mathit{pid} \leftarrow \mathit{sk} \leftarrow \varepsilon$ $\mathit{acc} \leftarrow \mathit{term} \leftarrow$ FALSE $C \leftarrow U$
    if $\mathit{state} =$ READY **then** {CLIENT ACTION 1}
        $\langle I \rangle \leftarrow \mathit{msg\text{-}in}$ where $I = \{i_1, \ldots, i_k\}$ and $i_1 < \cdots < i_k$
        $\mathit{state} \leftarrow \langle I \rangle$ $\mathit{msg\text{-}out} \leftarrow \langle C, I \rangle$
        return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
    elseif $\mathit{state} = \langle I \rangle$ **then** {CLIENT ACTION 2}
        $\langle c_{i_1}, \ldots, c_{i_k} \rangle \leftarrow \mathit{msg\text{-}in}$ where $c_{i_j} \in \{0,1\}^\kappa$ for all $i_j \in I$
        Obtain password π from user
        $\tilde{x}, \beta, \gamma \xleftarrow{R} \mathbb{Z}_q$ $\tilde{y} \leftarrow g^{\tilde{x}}$
        $B \leftarrow (y^\beta, g^\beta) \times (E_C)^\pi \times (g^{-1}, 1)$
        $V \leftarrow (h^\gamma g^\pi, g^\gamma)$
        $\tau \leftarrow \langle \tilde{y}, c_{i_1}, \ldots, c_{i_k} \rangle$
        $\sigma \leftarrow \mathsf{Prove}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (\beta, \pi, \gamma))$
        for $i \in I$ **do**
            $\tilde{y}_i \leftarrow (y'_i)^{\tilde{x}}$ $K_i \leftarrow H_2(I, \tau, \tilde{y}_i)$
        $\mathit{msg\text{-}out} \leftarrow \langle B, V, \tilde{y}, \sigma \rangle$
        $\mathit{acc} \leftarrow \mathit{term} \leftarrow$ TRUE $\mathit{state} \leftarrow$ DONE $\mathit{sk} \leftarrow \langle K_{i_1}, \ldots, K_{i_k} \rangle$
        $\mathit{sid} \leftarrow \langle C, I, \tau, B, V, \sigma \rangle$ $\mathit{pid} \leftarrow I$
        return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$

Figure 17: Specification of protocol (part 1: client side)

if $U \in \mathit{Servers}$ **then**
  $\mathit{sid} \leftarrow \mathit{pid} \leftarrow \mathit{sk} \leftarrow \varepsilon \quad \mathit{acc} \leftarrow \mathit{term} \leftarrow \text{FALSE} \quad i \leftarrow \mathsf{index}(U)$
  if $\mathit{state} = \text{READY}$ **then** {SERVER ACTION 1}
    $\langle C, I\rangle \leftarrow \mathit{msg\text{-}in}$ where $I = \{i_1, \ldots, i_k\}$, $i_1 < \cdots < i_k$, $i \in I$, and $C \in \mathit{Clients}$
    $c_i' \xleftarrow{R} \mathbb{Z}_q \quad \mathit{state} \leftarrow \langle C, I, c_i'\rangle \quad \mathit{msg\text{-}out} \leftarrow \langle i, c_i'\rangle$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  elseif $\mathit{state} = \langle C, I, c_i'\rangle$ **then** {SERVER ACTION 2}
    $\langle c_{i_1}, \ldots, c_{i_k}\rangle \leftarrow \mathit{msg\text{-}in}$ where $c_{i_j} \in \{0,1\}^\kappa$ for all $i_j \in I$
    $c_i \leftarrow c_i' \quad \mathit{msg\text{-}out} \leftarrow \varepsilon \quad \mathit{state} \leftarrow \langle C, I, \{c_j\}_{j\in I}\rangle$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  elseif $\mathit{state} = \langle C, I, \{c_j\}_{j\in I}\rangle$ **then** {SERVER ACTION 3}
    $\langle B, V, \tilde{y}, \sigma\rangle \leftarrow \mathit{msg\text{-}in}$
    $\tau \leftarrow \langle \tilde{y}, c_{i_1}, \ldots, c_{i_k}\rangle$
    if $\mathsf{Verify}_{\Phi_\mathcal{Q}}((\tau, E_C, B, V), \sigma)$ **then**
      $r_i, r_i', \gamma_i, \gamma_i', \gamma_i'' \xleftarrow{R} \mathbb{Z}_q$
      $B_i \leftarrow B^{r_i} \times (y, g)^{r_i'} \quad V_i \leftarrow (h^{\gamma_i} g^{r_i}, g^{\gamma_i})$
      $V_i' \leftarrow (h^{\gamma_i'}(V[1])^{r_i}, g^{\gamma_i'}) \quad V_i'' \leftarrow (h^{\gamma_i''}(V[2])^{r_i}, g^{\gamma_i''})$
      $\sigma_i \leftarrow \mathsf{Prove}_{\Phi_\mathcal{R}}((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$
      $\mathit{msg\text{-}out} \leftarrow \langle i, B_i, V_i, V_i', V_i'', \sigma_i\rangle \quad \mathit{state} \leftarrow \langle C, I, \tau, B, V, B_i, \tilde{y}, \sigma\rangle$
    else $\mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  elseif $\mathit{state} = \langle C, I, \tau, B, V, B_i, \tilde{y}, \sigma\rangle$ **then** {SERVER ACTION 4}
    $\langle Q_{i_1}, \ldots, Q_{i_k}\rangle \leftarrow \mathit{msg\text{-}in}$
    for $j \in I \setminus \{i\}$ **do** $\langle B_j, V_j, V_j', V_j'', \sigma_j\rangle \leftarrow Q_j$
    if $\forall j \in I \setminus \{i\} \ : \ [\mathsf{Verify}_{\Phi_\mathcal{R}}((j, B, V, B_j, V_j, V_j', V_j''), \sigma_j)]$ **then**
      $\tau' \leftarrow \langle \tau, B, V, B_{i_1}, \ldots, B_{i_k}, V_{i_1}, \ldots, V_{i_k}\rangle$
      $(\overline{y}, \overline{g}) \leftarrow \prod_{j\in I} B_j$
      $a_i \leftarrow \lambda_{i,I} x_i \quad \overline{C}_i \leftarrow \overline{g}^{a_i}$
      $\zeta \xleftarrow{R} \mathbb{Z}_q \quad R_i \leftarrow (h^\zeta (h')^{a_i}, g^\zeta)$
      for $j \in I$ **do** $C_j \leftarrow (y_j)^{\lambda_{j,I}}$
      $\Gamma_i \leftarrow \mathsf{Prove}_{\Phi_\mathcal{S}}((i, \tau', C_i, R_i), (a_i, \zeta));$
      $\mathit{msg\text{-}out} \leftarrow \langle i, R_i, \Gamma_i\rangle \quad \mathit{state} \leftarrow \langle C, I, \tau, B, V, \tilde{y}, \sigma, \overline{g}, \overline{y}, R_i, \zeta, \overline{C}_i, \{C_j\}_{j\in I}\rangle$
    else $\mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  elseif $\mathit{state} = \langle C, I, \tau, B, V, \tilde{y}, \sigma, \overline{g}, \overline{y}, R_i, \zeta, \overline{C}_i, \{C_j\}_{j\in I}\rangle$ **then** {SERVER ACTION 5}
    $\langle Q_{i_1}, \ldots, Q_{i_k}\rangle \leftarrow \mathit{msg\text{-}in}$
    for $j \in I \setminus \{i\}$ **do** $\langle R_j, \Gamma_j\rangle \leftarrow Q_j$
    if $\forall j \in I \setminus \{i\} \ : \ [\mathsf{Verify}_{\Phi_\mathcal{S}}((j, \tau', C_j, R_j), \Gamma_j)]$ **then**
      $\Gamma_i' \leftarrow \mathsf{Prove}_{\Phi_\mathcal{T}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a_i, \zeta));$
      $\mathit{msg\text{-}out} \leftarrow \langle i, \overline{C}_i, \Gamma_i'\rangle \quad \mathit{state} \leftarrow \langle C, I, \tau, B, V, \tilde{y}, \sigma, \overline{g}, \overline{y}, \overline{C}_i, \{C_j\}_{j\in I}\rangle$
    else $\mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  elseif $\mathit{state} = \langle C, I, \tau, B, V, \tilde{y}, \sigma, \overline{g}, \overline{y}, \overline{C}_i, \{C_j\}_{j\in I}\rangle$ **then** {SERVER ACTION 6}
    $\langle Q_{i_1}, \ldots, Q_{i_k}\rangle \leftarrow \mathit{msg\text{-}in}$
    for $j \in I \setminus \{i\}$ **do** $\langle \overline{C}_j, \Gamma_j'\rangle \leftarrow Q_j$
    if $\overline{y} = \Pi_{j\in I} \overline{C}_j$ **and** $\forall j \in I \setminus \{i\} \ : \ [\mathsf{Verify}_{\Phi_\mathcal{T}}((j, \tau', \overline{g}, \overline{C}_j, C_j, R_j), \Gamma_j')]$ **then**
      $\tilde{y}_i \leftarrow \tilde{y}^{x_i'} \quad K \leftarrow H_2(I, \tau, \tilde{y}_i)$
      $\mathit{acc} \leftarrow \mathit{term} \leftarrow \text{TRUE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon$
      $\mathit{sk} \leftarrow K \quad \mathit{sid} \leftarrow \langle C, I, \tau, B, V, \sigma\rangle \quad \mathit{pid} \leftarrow \text{C}$
    else $\mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
  else
    $\mathit{msg\text{-}out} \leftarrow \varepsilon$
    return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$

Figure 18: Specification of protocol (part 2: server side)