Security Characterization of J-PAKE and its Variants

Michel Abdalla[1,2], Manuel Barbosa[3], Peter B. Rønne[4], Peter Y.A. Ryan[4], and Petra Šala[1,4]

[1] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
michel.abdalla@ens.fr
[2] INRIA, Paris, France
[3] FCUP and INESC TEC, Porto, Portugal
mbb@fc.up.pt
[4] University of Luxembourg, Luxembourg
firstname.lastname@uni.lu

Abstract. The J-PAKE protocol is a Password Authenticated Key Establishment protocol whose security rests on Diffie-Hellman key establishment and Non-Interactive Zero Knowledge proofs. It has seen widespread deployment and has previously been proven secure, including forward secrecy, in a game-based model. In this paper we show that this earlier proof can be re-cast in the Universal Composability framework, thus yielding a stronger result. We also investigate the extension of such proofs to a significantly more efficient variant of the original J-PAKE, that drops the second round Non-Interactive Zero-Knowledge proofs, that we call sJ-PAKE. Adapting the proofs to this light-weight variant proves highly-non trivial, and requires novel proof strategies and the introduction of the algebraic group model. This means that J-PAKE implementations can be made more efficient by simply deleting parts of the code while retaining security under stronger assumptions. We also investigate the security of two further new variants that combine the efficiency gains of dropping the second round NIZK proofs with the gains achieved by two earlier, lightweight variants: RO-J-PAKE and CRS-J-PAKE. The earlier variants replaced the second Diffie-Hellman terms from each party by either a hash term or a CRS term, thus removing the need for half of the NIZK proofs in the first round. The efficiency and security assumptions of these variants are compared.

1 Introduction

The possibility of establishing confidential channels between remote entities without having to securely distribute key material in advance was proposed by Diffie and Hellman, [DH76]. However, mechanisms such as Diffie-Hellman key establishment still require some mechanism to authenticate the parties participating in the protocol to enable the parties to be sure who is at the other end of the newly established, secure channel, and to avoid man-in-the-middle attacks. Various ways of incorporating authentication to such protocols have been proposed, including digitally signing messages or incorporating long term key material into the computation of the session key, as in the MTI class of protocols, [MTI86]. However, these assume some way for the parties to be confident of the public keys associated with the other parties, thus requiring either some form of PKI or a web of trust etc. Another possibility is the use of out-of-band channels, authenticated by some other means such as line of sight or distance bounding, to confirm the identity of parties involved in the protocol.

A further possibility, and the subject of this paper, is to have the parties share a (typically low entropy) secret in advance and using this to bootstrap the authenticity of subsequently established session keys. This idea first appears in [Mer82] and is usually referred to as Password Authenticated Key Establishment (PAKE). The key challenge in designing a PAKE is to ensure that an attacker cannot derive sufficient information to execute an offline dictionary attack, i.e. never gets enough to (tractably) confirm or deny in an offline computation a guess at the password. Such attackers might be passive, simply eavesdropping, or active, masquerading as one or more of the parties. These two attack models are actually incomparable. This is essential as the passwords are typically low entropy and hence vulnerable to brute force attack. Of course, online guesses are unavoidable, but such attacks are detectable by the honest parties and can be throttled by for example limiting the number of tries.

J-PAKE introduces a novel paradigm in the design of PAKEs, inspired by the elegant algebraic cancellation construction of the anonymous vote protocol proposed by Hao and Zelinsky, [HZ06]. Aside from the basic Diffie-Hellman construction, J-PAKE relies only on ZK proofs of knowledge. We note in passing that these ZK proofs could be replaced by commitments, but at the cost of extra rounds. J-PAKE is included in ISO/IEC 11770-4 (2017) and is one of the most used PAKE protocols in practice. For this reason, it is beneficial to have J-PAKE proven in *Universally composable* [Can01] model. Moreover, we offer a simplified version that saves some communication and computation cost which is also beneficial, as the current users can just take the existing implementation and do a minimal change to it and have the same efficiency for the lower cost.

It is sometimes argued that passwords will die out and replaced by other mechanisms for authentication, but there is little evidence of this to date. Passwords remain the foremost mechanism for authentication, sometimes in conjunction with other mechanisms in multi-factor approaches. In particular the use of PAKEs in access control greatly improves the security of password based authenticated channel by protecting the passwords in transit, thus arguably giving passwords a new lease of life.

1.1 Previous work

Although, there has been an abundance of PAKE protocols, see [BM92, Wu98, Jab96, JR13, KOY01, Har15, AP05, HR10, TWMP07, BR94, Mac02, BRRS18, HL19], in the last two decades, we mention just a representative few, relevant to our work. Starting with EKE by Bellovin and Merritt [BM92] and SPEKE [Jab96] which were seminal but shown prone to attacks. Popularity of PAKE protocols grew as prominent complexity-theoretic security models ([BPR00, AFP05, BMP00, CK01]) precisely defined guidelines to prove PAKE protocols secure. For instance, PAK and PPK [Mac02], SPAKE2 [AP05], J-PAKE [HR10] are proven secure in *Indistiguishability-based models* [AFP05, BPR00]. Even though Indistiguishability -based models are considered to be the weakest security proofs models for PAKE(s), they still represent a high-standard bar that a certain PAKE protocol needs to reach to be taken into consideration for practice use. A stronger model that makes no assumptions on the passwords and captures more real-world scenarios is *Universally composable* (UC) model designed by Canetti et al. [CK01]. SPAKE2+ [Sho20] and OPAQUE [JKX18] are some

of the more recent protocols proven secure in the UC framework, which guarantees security under arbitrary protocol composition. Furthermore, one of the benefits of UC is that it provides security for strong asymmetric PAKEs when converted from symmetric PAKEs [GMR06]. The practice has shown that having strong asymmetric PAKEs is of greatest importance when considering PAKEs for standardization because they are resilient to *server-compromise* in the client-server setting. Until recently, only protocols that could administrate the scenario where an adversary conducts an online attack against the session by making a password guess *before* that session is completed could be proven UC secure. More precisely, PAKE protocols in which the password could be *extracted* from the adversary's message before both parties hold the session key were proven in the UC framework. For the same reason, we prove J-PAKE secure in the UC setting. Finally, protocols SPAKE2 [AP05], SPEKE and protocols based on SPEKE [HS14, Jab97, Mac01], such as TBSPEKE [PW17] and CPace [HL19] were the first ones proven secure in *Universally Composable relaxed model* (UC-relaxed) [ABB+20]. UC relaxed model captures the same security requirements as UC model, but allows the adversary, while making an online attack on the session, to make its password guess *after* that session terminates. More concretely, protocols that have one message that corresponds to *perfectly hiding* commitment to the password could be proven secure in UC -relaxed. While security proofs are desirable, they are not vital for the protocol to be deployed in practice. A prime example of that is SRP [Wu98] which is currently the most used PAKE protocol. In contrast to SRP, which does not have a full proof of security, J-PAKE has a formal proof [ABM15]. Perhaps due to its simplicity and freedom from patents, J-PAKE has seen significant real-world deployment. Its applications include the Thread protocol (in IoT) [Thr16], Pale Moon [Pal16] and OpenSSL [Ope16] library. Early steps to make J-PAKE more efficient yielded two other variations, CRS-J-PAKE and RO-J-PAKE [LST16] which were proven secure in a similar fashion to J-PAKE [ABM15]. CRS-J-PAKE and RO-J-PAKE showed a true potential to be considered for deployment. However, the additional assumptions such as common reference string and *full domain hash* were cumbersome for the implementation. Widely-world deployment of J-PAKE encouraged us to search for its lighter versions, which do not need much change in the implementation. For this reason, this paper seeks further efficiency improvements of J-PAKE by reducing the use of zero-knowledge proofs.

1.2 Our contribution

Our starting point is the optimization of J-PAKE shown in Figure 1, which we call sJ-PAKE. In comparison to the original design, sJ-PAKE drops the zero knowledge proofs for α and β in the original protocol and introduces a hashing step to derive the shared key. We show that further simplifications that naively omit the zero-knowledge proofs in the first round of the protocol result in insecure protocols, and that introducing a hashing step is also necessary, as the protocol is vulnerable to a related key attack without it.

We then consider the question of formally proving that sJ-PAKE is secure. Our results show that the proposed optimization leads to a protocol that is surprisingly hard to prove secure and requires completely different techniques than those used in the original proof of J-PAKE.

To highlight this fact, we first show that the use of zero-knowledge proofs of knowledge (ZK-PoK) in J-PAKE actually allows to establish a much stronger result than that given in [ABM15]: with small modifications to the proof, and assuming the same form of ZK-PoK protocol, we show that J-PAKE is actually secure in the Universal Composability framework with respect to the standard ideal functionality for PAKE protocols.[5]

In contrast, we could not find a close adaptation of the original proof of J-PAKE that applies to sJ-PAKE. We give a proof that sJ-PAKE achieves game-based security and provides perfect forward secrecy in the random oracle model, using proof techniques similar to those used for SPAKE2 [AB19]. However, in contrast to SPAKE2, the proof must be carried out in the algebraic group model, even in the case of weak forward secrecy. Intuitively, the reason for this is that SPAKE2 fixes a common reference string that is totally

[5] In [ABM15] the underlying ZK PoK protocol is instantiated with the Schnorr protocol, which is proved to satisfy the required form of extractability in the algebraic group model. We do not investigate how this can be done in the UC setting, as the goal of our UC proof is only to demonstrate how strongly the original proof for J-PAKE relies on the ZK-PoK component.

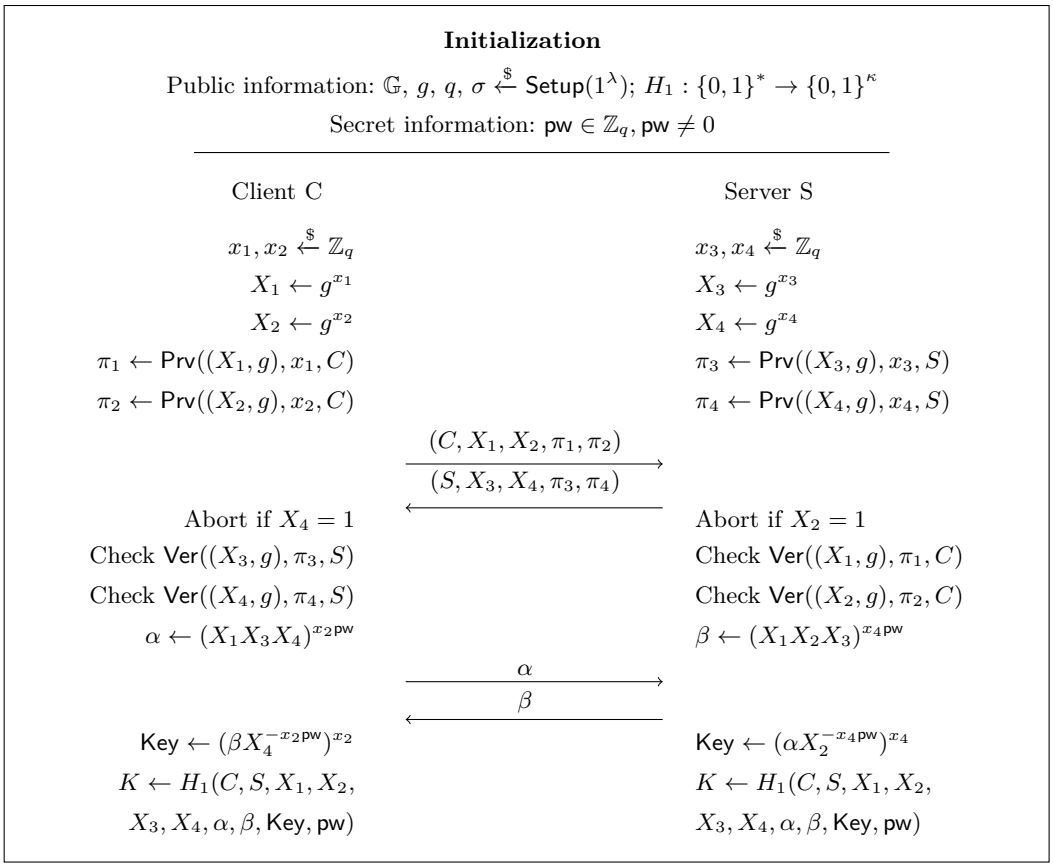

Fig. 1: The sJ-PAKE protocol. For comparison, the J-PAKE is given in Fig. 6.

out of the attacker's control in which it is possible to embed a hard problem instance, whereas in sJ-PAKE, the adversary has some additional power in choosing the group elements that are used to compute the secret key.

1.3 Organization of the paper

The rest of the paper is organized as follows. In section 2, we describe evolution from J-PAKE to sJ-PAKE. In section 3 we cover preliminaries where we give a list of definitions and security assumptions useful for the proofs. Then in section 4 we lay out the UC proof of J-PAKE, followed by the description of the model in section 5. We layout the game based proof of sJ-PAKE in section 6. We conclude the paper in section 7, where we compare J-PAKE and sJ-PAKE to all its variants and give a brief conclusion. In addition, we supply proof details on UC and sJ-PAKE in Supplementary material in Sec. A and B. Finally, we provide game-based proofs for sRO-J-PAKE and sCRS-J-PAKE as Supplementary material in Sec. D.

2 From J-PAKE to sJ-PAKE

The initial idea of our sJ-PAKE protocol was to slightly modify J-PAKE protocol by omitting NIZK proofs in the second round and prove sJ-PAKE secure in the same fashion as Abdalla et al. in [ABM15]. Note that removing the first round NIZK proofs leads to simple offline dictionary attacks in which the receiver incorporates X_1^{-1} in X_3 or X_4 resulting in X_1 cancelling out in the computation of α. However, omitting

NIZKs in the second round and computing $K = \mathsf{Key}$ as in J-PAKE, leads to so called *Related key attack* [AP05], where the adversary acts as a *man-in-the-middle* and succeeds in inducing the parties to hold a keys with a known relationship. This scenario is not desired, as an adversary can easily create different sessions with related key values. In Fig. 2 we show the attack, which we describe below.

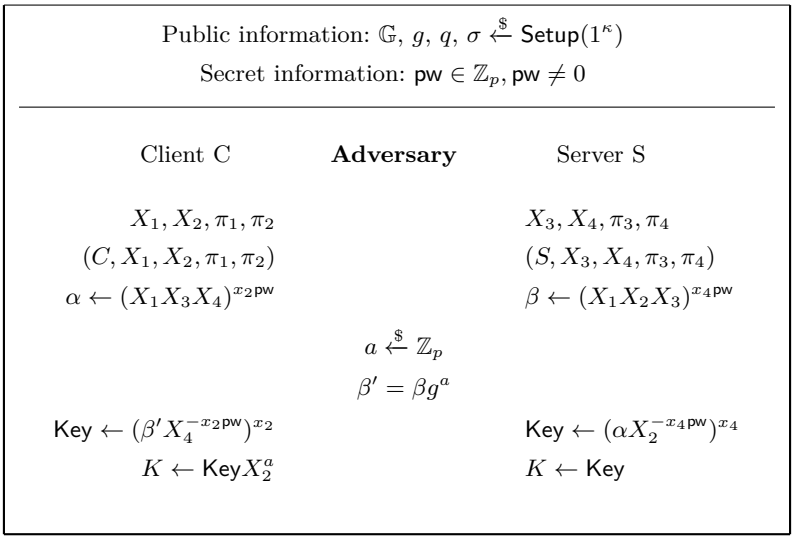

Fig. 2: Related key attack if $K = \mathsf{Key}$.

Out of simplicity, we demonstrate the attack between one client π_C instance and one server instance π_S. The first round goes in the same manner as in J-PAKE, with an adversary just forwarding the messages. In the second round, π_C computes α and sends it to π_S who also computes β and sends it to π_C. Then, because there is no proof of knowledge for β, the adversary steps in: intercepts β from π_S and computes its own β' as $\beta' = \beta g^a$, for some $a \xleftarrow{\$} \mathbb{Z}_p$ and sends it to π_C. Then, π_C receives β', computes Key as usual and gets $K_C = \mathsf{Key} X_2^a$, while π_S computes its session key and gets $K_S = \mathsf{Key}$. Notice that session keys are linked, and if the adversary tests one instance, it could obtain its session key K and all the adversary needs to do is to compute the other party's session key as $K_S = \frac{K_C}{X_2^a}$. Furthermore, that attack works also for the other party (with resp. to α). Therefore, to prove sJ-PAKE secure, we introduce the hashing step to derive the session key, which is in any case good practice.

2.1 Variants of sJ-PAKE

After J-PAKE was proven secure, two of its variants were proposed, CRS-J-PAKE and RO-J-PAKE. Furthermore, CRS-J-PAKE and RO-J-PAKE were proven secure analogously as their big brother in [LST16]. The significant difference between them is that both, CRS-J-PAKE and RO-J-PAKE, omit one computation in the first round on each side, for a client-side $X_1 = g^{x_1}$ and for a server-side $X_3 = g^{x_3}$. This implies that two ZK-PoK are dropped, π_1 and π_3. This idea stems from the observation that x_1 and x_3 are not in fact required to compute α, β and K. However, we cannot simply drop the x_1 and x_3 as it is essential for the security to have additional terms in the exponents of α, β and K that are unknown to the parties (other than the x_2 unknown to S and x_4 unknown to C). Thus the X_1 and X_3 are replaced by terms computed as hashes or as a single CRS term. Fortunately, we can derive variants for sJ-PAKE, sCRS-J-PAKE (Fig. 9) and sRO-J-PAKE (Fig. 7) in the same manner. We describe these variants in detail in Supplementary material in Sec. D where we also provide their security proof. We compare the efficiency with sJ-PAKE in section 7. Keeping NIZK proofs in two rounds made both CRS-J-PAKE and RO-J-PAKE provable in the

same fashion as [ABM15]. This is not the case with sJ-PAKE as π_α and π_β are omitted in the second round, which makes its proof very different from [ABM15] and [LST16].

3 Preliminaries

3.1 Notation

We use calligraphic letters to denote adversaries, typically $\mathcal{A}$. We write $d \xleftarrow{\$} D$ for sampling uniformly at random from set D and $|D|$ to denote the number of elements in D. Let $\{0,1\}^*$ denote the bit string of arbitrary length while $\{0,1\}^l$ stands for those of length l. When we sample elements from $\mathbb{Z}_q$, it is understood that they are viewed as integers in [0 ... q-1], and all operations on these are performed *mod* q. Finally, let κ denote the security parameter, and $negl(\kappa)$ denote a negligible function.

3.2 Security Assumptions

We now state the security assumptions that will be used in this paper. We assume that we have a cyclic group $\mathbb{G}$ of prime order q which is generated by g and which is the output of a public algorithm $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(\kappa)$ with κ being the security parameter.

Definition 3.1. *Computational Diffie-Hellman (CDH) Problem. Given g^x, g^y, compute g^{xy}, where $\{g^x, g^y\} \in \mathbb{G}$. Let the advantage of an algorithm $\mathcal{A}$ in solving the CDH problem be:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{A}) = \Pr\left[x \xleftarrow{\$} \mathbb{Z}_q : g^{xy} = \mathcal{A}(g^x, g^y)\right].$$

Under the *CDH assumption* there exist sequences of cyclic groups $\mathbb{G}$ indexed by κ s.t. $\forall\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CDH}}(\mathcal{A})$ is a negligible function.

Definition 3.2. *Computational Square Diffie-Hellman (CqSDH) Problem. Given g^x compute g^{x^2}, where $\{g^x, g^{x^2}\} \in \mathbb{G}$. Let the advantage of an algorithm $\mathcal{A}$ in solving the CSqDH problem be:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CSqDH}}(\mathcal{A}) = \Pr\left[x \xleftarrow{\$} \mathbb{Z}_q : g^{x^2} = \mathcal{A}(g^x)\right].$$

Under the *CSqDH assumption* there exist sequences of cyclic groups $\mathbb{G}$ indexed by κ s.t. $\forall\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CSqDH}}(\mathcal{A})$ is a negligible function.

Definition 3.3. *Computational Triple Group Diffie-Hellman (CTGDH) Problem. Given g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, where $g^{xy} = DH(g^x, g^y)$, $g^{xz} = DH(g^x, g^z)$, $g^{yz} = DH(g^y, g^z)$, compute g^{xyz}, where $\{g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, g^{xyz}\} \in \mathbb{G}$. Let the advantage of an algorithm $\mathcal{A}$ in solving the CTGDH problem be:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CTGDH}}(\mathcal{A}) = \Pr\left[(x,y,z) \xleftarrow{\$} \mathbb{Z}_q^3 : g^{xyz} = \mathcal{A}(g^x, g^y, g^z, g^{xy}, g^{yz}, g^{xz})\right].$$

Under the *CTGDH assumption* there exist sequences of cyclic groups $\mathbb{G}$ indexed by κ s.t. $\forall\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}_{\mathbb{G}}^{\mathtt{CTGDH}}(\mathcal{A})$ is a negligible function.

Definition 3.4. *Decisional Diffie-Hellman (DDH) Problem. Given (g^x, g^y, g^z), distinguish whether $z = xy$ or $z = r$, where r is is randomly chosen from $\mathbb{Z}_q$ and $\{g^x, g^y, g^z\} \in \mathbb{G}$. Let the advantage of a algorithm $\mathcal{A}$ in solving DDH problem be:*

$$\mathrm{Adv}_{\mathbb{G}}^{\mathtt{DDH}}(\mathcal{A}) \quad = \quad |\Pr\left[(x,y) \xleftarrow{\$} \mathbb{Z}_q^2 : 1 = \mathcal{A}(g^x, g^y, g^{xy})\right] \quad - \quad \Pr\left[(x,y,r) \xleftarrow{\$} \mathbb{Z}_q^3 : 1 = \mathcal{A}(g^x, g^y, g^r)\right]|.$$

Under *DDH assumption* there exist sequences of cyclic groups $\mathbb{G}$, indexed by a security parameter κ, such that for all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{A})$ is a negligible function.

Definition 3.5. *Decisional Square Diffie-Hellman (DSqDH) Problem. Given (g^x, g^z), distinguish whether $z = x^2$ or $z = r$, where r is randomly chosen from $\mathbb{Z}_q$ and $\{g^x, g^z\} \in \mathbb{G}$. Let the advantage of a algorithm $\mathcal{A}$ in solving DSqDH problem be:*

$$\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathbb{G}}(\mathcal{A}) \quad = \quad |\Pr\left[x \xleftarrow{\$} \mathbb{Z}_q : 1 = \mathcal{A}(g^x, g^{x^2})\right] \quad - \quad \Pr\left[(x,r) \xleftarrow{\$} \mathbb{Z}_q^2 : 1 = \mathcal{A}(g^x, g^r)\right]|.$$

Under *DSqDH assumption* there exist sequences of cyclic groups $\mathbb{G}$, indexed by a security parameter κ, such that for all $\mathcal{A}$ running in time t polynomial in κ, $\mathrm{Adv}^{\mathtt{DSqDH}}(\mathcal{A})$ is a negligible function.

The relationships between the assumptions can be found in [ABM15] and [BDZ03]. Ignoring the time for exponentiation we have

- $\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}})$, where $\mathcal{B}$ and $\mathcal{B}^{\mathcal{A}}$ are PPT algorithms running in time t.
- $\mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}(\mathcal{B}) \geq (\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}}))^2$, where $\mathcal{B}$ and $\mathcal{B}^{\mathcal{A}}$ are PPT algorithms running in time t.
- $\mathrm{Adv}^{\mathtt{DSqDH}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{DDH}}_{\mathbb{G}}(\mathcal{B}^{\mathcal{A}})$, where $\mathcal{B}$ and $\mathcal{B}^{\mathcal{A}}$ are PPT algorithms running in time t.
- $\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathbb{G}}(\mathcal{B}) \geq \mathrm{Adv}^{\mathtt{CDH}}_{\mathbb{G}}((\mathcal{B}^{\mathcal{A}}))^3$, where $\mathcal{B}$ and $\mathcal{B}^{\mathcal{A}}$ are PPT algorithms running in time t.

For CTGDH see also [BCP03].

Definition 3.6. SE-NIZK *is a function defined by the tuple* $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver}, \mathsf{Sim}_1, \mathsf{Sim}, \mathsf{Extract})$ *such that* $(\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver})$ *is a non-interactive proof system and:*

- $\mathsf{Sim}_1(1^\kappa)$ *generates* CRS crs *and two trapdoors* td_s *and* td_e
- $\mathsf{Sim}(\mathsf{crs}, \mathsf{td}_s, X, l)$ *takes* crs, td_s, X, *some label* l, *and outputs a simulated proof of knowledge for* X, π
- $\mathsf{Extract}(\mathsf{crs}, \mathsf{td}_e, X, \pi, l)$ *extracts a witness* x *for* X, *from valid a proof of knowledge* π, *considering some label* l, *trapdoor* td_e *and CRS* crs, *if possible. Otherwise it aborts.*

(i) Unbounded zero knowledge property holds if the adversary cannot distinguish between real and simulated proofs. We define $\mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}()$ *as:*

$$\Pr\left[\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^\kappa) : \mathcal{A}^{\mathsf{NIZK.Prv}(\mathsf{crs},\cdot,\cdot,\cdot)}(\mathsf{crs}) = 1\right] - \Pr\left[\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa) : \mathcal{A}^{\mathsf{NIZK.Sim}'(\mathsf{crs},\mathsf{td}_s,\cdot,\cdot)}(\mathsf{crs}) = 1\right]$$

where $\mathsf{NIZK.Sim}'(\mathsf{crs}, \mathsf{td}_s, X, x, l) = \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X, l)$ *where* l *is* C *or* S. *The adversary can ask at most* n_{sim} *queries to* $\mathsf{NIZK.Sim}$ *or to* $\mathsf{NIZK.Prv}$.

(ii) Simulation-sound extractability holds if we can extract a witness x *for* X, *from valid a proof of knowledge* π, *even when* $\mathcal{A}$ *sees simulated proofs. We define* $\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}()$ *as:*

$$\Pr\Big[\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e \xleftarrow{\$} \mathsf{Sim}_1(1^\kappa); (X, \pi) \leftarrow \mathcal{A}^{\mathsf{NIZK.Sim}(\mathsf{crs},\mathsf{td}_s,\cdot,\cdot)}(\mathsf{crs}, \mathsf{td}_e) = 1 : \mathsf{NIZK.Ver}(\mathsf{crs}, X, \pi, l) = 1, ((X, l), \pi) \notin S; \mathcal{R}(x, \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X, \pi, l)) = 0\Big]$$

where S *is a set of query-response pairs* $((X, l), \pi)$ *for* $\mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, \cdot, \cdot)$.

Definition 3.7. *Given some distribution* $(g_1, ..., g_n) \xleftarrow{\$} \mathcal{D}$, *where* $\mathcal{D}$ *is a hard linear distribution in* G^n, *for some* n, *it is computationally hard to find* $(\mu_1, ..., \mu_n) \neq 0$ *such that* $g_1^{\mu_1} \cdots g_n^{\mu_n} = 1$. *More precisely, let* $\mathrm{Adv}^{\mathrm{hard-lin}}_{\mathcal{D}}(\mathcal{A})$ *be*

$$\Pr\left[(g_1, ...g_n) \xleftarrow{\$} \mathcal{D}; (\mu_1, ..., \mu_n) \xleftarrow{\$} \mathcal{A}(g_1, ...g_n) : g_1^{\mu_1} \cdots g_n^{\mu_n} = 1\right]$$

For details, we refer to [ABM15].

3.3 Algebraic adversaries

The algebraic adversary model was introduced in [PV05] and first used constructively to prove security in [ABM15], actually also in relation to the J-PAKE protocol. The model was further formalised as the *algebraic group model* in [FKL18]. In essence, given a prime order group $\mathbb{G}$, we say that an adversary algorithm is *algebraic* if whenever it outputs a group element, $g \in \mathbb{G}$, it also outputs a discrete log representation, $r_1, \ldots, r_k$, in terms of all the group elements, $g_1, \ldots, g_k$ that it received so far, i.e. $g = g_1^{r_1} \cdots g_k^{r_k}$. This gives a model which is weaker than the generic group model and allows us to do security reductions. Further, in the algebraic group model [FKL18] it was shown how important security assumptions such as Computational Diffie-Hellman assumption are related to the Discrete Log assumption, see also [BFL20] for further classifications. Importantly, the algebraic adversary model was used in [AB19] to prove perfect forward security of SPAKE2. In this paper, we will need it to prove security of the sJ-PAKE protocol since without the Zero-Knowledge Proofs from the second round terms we cannot extract the password guess directly.

3.4 Algebraic simulation-sound extractable NIZK Proof

In the original J-PAKE protocol the non-interactive zero-knowledge proofs of knowledge are instantiated as Schnorr proofs [Sch90]. In [ABM15] it was proven that when we assume algebraic adversaries, these proofs are *algebraic simulation sound extracatable NIZK* (alg-SE-NIZK). The definition of alg-SE-NIZK relies on two conditions, namely (i) Weak algebraic simulation-sound extractability and (ii) Base indistinguishability, for more details see [ABM15]. This definition is slightly weaker that SE-NIZK, however, as shown in [ABM15] the security proof for J-PAKE which relied on SE-NIZK proofs, can be updated to only assume alg-SE-NIZK proofs since all reductions were algebraic and all the bases used in the proofs were hard-linear, see Def. 3.7. Fortunately, in our proof of sJ-PAKE below this also holds true and we can simply assume SE-NIZK proofs. When the NIZK proofs in sJ-PAKE are instantiated as Schnorr proofs, the security proof can be updated as in [ABM15], especially, we already assume algebraic adversaries.

4 UC Security of J-PAKE

In this section we show that the original proof of J-PAKE given in [ABM15] can be adapted with minor modifications to show that the protocol achieves security in the Universal Composability setting [Can01]. Formally, we show that the simulator given in Figure 4 can emulate the actions of a dummy adversary interacting with J-PAKE, while interacting with the standard ideal functionality for PAKE given in Figure 3. We give the simulator and proof for the three-pass variant of the protocol for simplicity, where parties have fixed roles and there is a pre-defined message flow, but the result should extend naturally to the more flexible variant where messages can be transmitted in any order. We also note that it has been recently been pointed out [AHH21] that the classical version of the PAKE UC functionality may not be strong enough for modular design of higher level protocols. The reason is that the functionality allows the simulator to fix the output of an honest party when this party is interacting with a corrupt peer, even if the password has not been leaked. The solution proposed in [AHH21] is to treat such sessions as any other session, and require the simulator to extract the password from messages sent by corrupt parties in order to program the output (i.e., in case of unsuccessful extraction the output key produced by the honest party should be indistinguishable from random). It was suggested in [AHH21] that the security proofs of most PAKE protocols will probably extend to this case. We show that this is indeed the case for J-PAKE.

Theorem 4.1. *J-PAKE UC-emulates $\mathcal{F}_{\mathsf{pake}}$. More precisely,, there exists a simulator that interacts with $\mathcal{F}_{\mathsf{pake}}$ and presents an ideal world view that no ppt environment can distinguish from the real-world view produced by a dummy adversary interacting with J-PAKE.*

Proof: The proof shows that the simulator in Figure 4 provides a view to the environment that is consistent with that observable via the dummy adversary in the real world; more precisely, we show no ppt environment will be able to distinguish the simulated view from the real one if the following hypotheses hold: i. the NIZK

Session initiation

On $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{pw}, \mathsf{role})$ from P, ignore this query if record $\langle \mathsf{sid}, P, \cdot, \cdot, \cdot \rangle$ already exists. Otherwise record $\langle \mathsf{sid}, P, P', \mathsf{pw}, \mathsf{role} \rangle$ marked fresh and send $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{role})$ to $\mathcal{A}$.

Active attack

- On $(\mathsf{TestPw}, \mathsf{sid}, P, \mathsf{pw}^*)$ from $\mathcal{A}$, if $\exists$ a fresh record $\langle \mathsf{sid}, P, P', \mathsf{pw}, \cdot \rangle$ then:
  - If $\mathsf{pw}^* = \mathsf{pw}$ then mark it $\mathsf{compromised}$ and return "correct guess";
  - If $\mathsf{pw}^* \neq \mathsf{pw}$ then mark it $\mathsf{interrupted}$ and return "wrong guess".

Key generation

On $(\mathsf{NewKey}, \mathsf{sid}, P, K^*)$ from $\mathcal{A}$, if $\exists$ a record $\langle \mathsf{sid}, P, P', \mathsf{pw}, \mathsf{role} \rangle$ not marked $\mathsf{completed}$ then do:

- If the record is $\mathsf{compromised}$, then set $K := K^*$.
- If the record is fresh and $\exists$ a $\mathsf{completed}$ record $\langle \mathsf{sid}, P', P, \mathsf{pw}, \mathsf{role}', K' \rangle$ with $\mathsf{role}' \neq \mathsf{role}$ that was fresh when P' output (sid, K'), then set $K := K'$.
- In all other cases pick $K \leftarrow_{\$} \{0,1\}^{\lambda}$.

Finally, append K to record $\langle \mathsf{sid}, P, P', \mathsf{pw}, \mathsf{role} \rangle$, mark it $\mathsf{completed}$, and output (sid, K) to P.

Fig. 3: The original PAKE functionality $\mathcal{F}_{\mathsf{pake}}$ of Canetti et al. [CHK+05].

protocol is a simulation-sound zero-knowledge proof-of-knowledge; ii. the key derivation function is a PRF; and iii. the decisional squared Diffie-Hellman assumption holds in the underlying group (and hence also the standard DDH assumption).

Simulation strategy. Our simulator generates messages X_1 to X_4 as per the protocol, but it generates totally random values of α and β because it does not know the correct passwords. It uses its ability to produce proofs without explicitly providing a witness in all the messages it generates, which explains why the environment cannot trivially detect that the simulator is cheating in α and β. Moreover, the fact that the simulator does not need to provide witnesses to generate the proofs enables us to program hard problem instances into all the simulated messages when we prove that no adversarial strategy can efficiently catch the simulator in its cheating.

The simulator does not interfere with passively attacked sessions, i.e., sessions which are created by the adversary by simply relaying messages, following the protocol, from another simulated session. Moreover, the simulator does not need to perform any form of extraction when the adversary performs a weak form of active attack, which we call *swapping*, in which it only switches the order of the group elements (X_1, X_2) or (X_3, X_4); in this case the simulator only interrupts the sessions, so that the ideal functionality chooses completely random keys.

Finally, to deal with more general active attacks, the simulator needs to extract the discrete logarithms of all adversarially created messages, which it does by using the NIZK protocol trapdoor. We show that, except with negligible probability, this means that the simulator extracts a password guess from any adversarial α or β used in an active attack: intuitively, if all the discrete logarithms of the messages sent and received by a session are known to the simulator—which happens if the NIZK extraction does not fail—then this determines a unique password that the simulator can recover. Once it holds a password, the simulator tests if it is correct. For the case when the password guess is correct, we show that the simulator can compute the correct key—i.e., the key as the adversary expects to see it based of the simulated random α or β in that session—with overwhelming probability, which it uses to program the ideal functionality output. If the password is incorrect, the ideal functionality will choose a totally random key.

Structure of the indistinguishability proof. To prove that this is a good simulator we need to perform a number of steps, which closely follow the intuition of the original proof given in [ABM15]. Before giving an outline, we should formally introduce the notions of matching and swapped sessions.

On initialization:
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \leftarrow\$ \mathsf{NIZK.Backdoor}(\,)$
Send crs to $\mathcal{A}$
$T_p \leftarrow \{\,\}$; $T_e \leftarrow \{\,\}$

On $(\mathsf{NewSession}, \mathsf{sid}, P, P', \mathsf{role} = C)$ from $\mathcal{F}_{\mathsf{pake}}$:
$x_1, x_2 \leftarrow\$ \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$
$\pi_1 \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (P, \mathsf{sid}), (g, X_1))$
$\pi_2 \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (P, \mathsf{sid}), (g, X_2))$
$T_p[X_1] \leftarrow (\mathsf{sid}, P, g, x_1, \pi_1)$
$T_p[X_2] \leftarrow (\mathsf{sid}, P, g, x_2, \pi_2)$
$\pi_P^{\mathsf{sid}} \leftarrow (P, P', X_1, X_2, \bot, \bot, \bot, \bot, C)$
Send $(P, P', \mathsf{sid}, X_1, \pi_1, X_2, \pi_2))$ to $\mathcal{A}$

On $(P', P, \mathsf{sid}, X_3, \pi_3, X_4, \pi_4, \beta, \pi_\beta)$ from $\mathcal{A}$:
If $\pi_P^{\mathsf{sid}} \neq (P, P', X_1, X_2, \bot, \bot, \bot, \bot, C)$ ignore input
If $X_4 = 1$ then $\pi_P^{\mathsf{sid}} \leftarrow \bot$; return
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P', \mathsf{sid}), (g, X_3), \pi_3) = \bot$
  then $\pi_P^{\mathsf{sid}} \leftarrow \bot$; return
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P', \mathsf{sid}), (g, X_4), \pi_4) = \bot$
  then $\pi_P^{\mathsf{sid}} \leftarrow \bot$; return
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P', \mathsf{sid}), (X_1X_2X_3, \beta), \pi_\beta) = \bot$
  then $\pi_P^{\mathsf{sid}} \leftarrow \bot$; return
$K \leftarrow\$ \mathcal{K}$
$x_\alpha \leftarrow\$ \mathbb{Z}_q$; $\alpha \leftarrow (X_1X_3X_4)^{x_\alpha}$
$\pi_\alpha \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (P, \mathsf{sid}), (X_1X_3X_4, \alpha))$
If $\pi_{P'}^{\mathsf{sid}} = (P', P, X_1, X_2, X_3, X_4, \bot, \beta, S)$
  (Passive attack; do not interrupt)
Else If $\pi_{P'}^{\mathsf{sid}} = (P', P, X_1, X_2, X_3, X_4, \bot, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P', P, X_2, X_1, X_3, X_4, \bot, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P', P, X_1, X_2, X_4, X_3, \bot, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P', P, X_2, X_1, X_4, X_3, \bot, \beta, S)$
  (Swap attack; interrupt without pw)
  Query $\mathsf{TestPw}(P, \mathsf{sid}, \mathsf{pw} = \bot)$
Else
  (Active attack; we need to extract a password)
  $x_3 \leftarrow \mathsf{Extract}(P', \mathsf{sid}, g, X_3, \pi_3)$
  $x_4 \leftarrow \mathsf{Extract}(P', \mathsf{sid}, g, X_4, \pi_4)$
  $x_\beta \leftarrow \mathsf{Extract}(P', \mathsf{sid}, X_1X_2X_3, \beta, \pi_\beta)$
  If $X_3 \neq g^{x_3} \vee X_4 \neq g^{x_4}$ then ABORT
  If $\beta \neq (X_1X_2X_3)^{x_\beta}$ then ABORT
  Query $\mathsf{TestPw}(P, \mathsf{sid}, \mathsf{pw} = x_\beta/x_4)$
  If correct guess $K \leftarrow \mathsf{PRF}((\alpha X_2^{-x_4\mathsf{pw}})^{x_4}, \mathsf{sid})$
$T_p[\alpha] \leftarrow (P, C, X_1X_3X_4, x_\alpha, \pi_\alpha)$
$\pi_{P,C}^{\mathsf{sid}} \leftarrow (P, P', X_1, X_2, X_3, X_4, \alpha, \beta)$
Send $((P', \mathsf{sid}) \leftarrow (\alpha, \pi_\alpha))$ to $\mathcal{A}$
Query $\mathsf{NewKey}(P', \mathsf{sid}, K)$

$\mathsf{Extract}(P, \mathsf{sid}, \hat{g}, X, \pi)$
If $T_p[X] = (P, \mathsf{sid}, \hat{g}, x, \pi)$ return x
If $T_e[X] = (P, \mathsf{sid}, \hat{g}, x, \pi)$ return x
$x \leftarrow \mathsf{NIZK.Extract}(\mathsf{td}_e, (P, \mathsf{sid}), (\hat{g}, X), \pi)$
$T_e[X] \leftarrow (P, \mathsf{sid}, \hat{g}, x, \pi)$; return x

On $(\mathsf{NewSession}, \mathsf{sid}, P', P, \mathsf{role} = S)$ from $\mathcal{F}_{\mathsf{pake}}$:
$x_3, x_4 \leftarrow\$ \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$
$\pi_3 \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (\mathsf{sid}, P), (g, X_3))$
$\pi_4 \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (\mathsf{sid}, P), (g, X_4))$
$T_p[X_3] \leftarrow (P', g, x_3, \pi_3)$
$T_p[X_4] \leftarrow (P', g, x_4, \pi_4)$
$\pi_{P'}^{\mathsf{sid}} \leftarrow (P', P, \bot, \bot, X_3, X_4, \bot, \bot, S)$

On $(P, P', \mathsf{sid}, X_1, \pi_1, X_2, \pi_2)$ from $\mathcal{A}$:
If $\pi_{P'}^{\mathsf{sid}} \neq (P', P, \bot, \bot, X_3, X_4, \bot, \bot, S)$ ignore input
If $X_2 = 1$ then $\pi_{P'}^{\mathsf{sid}} \leftarrow \bot$; return
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P, \mathsf{sid}), (g, X_1), \pi_1) = \bot$
  then $\pi_{P'}^{\mathsf{sid}} \leftarrow \bot$; return
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P, \mathsf{sid}), (g, X_2), \pi_2) = \bot$
  then $\pi_{P'}^{\mathsf{sid}} \leftarrow \bot$; return
$\mathsf{Extract}(P', \mathsf{sid}, g, X_1, \pi_1)$
$\mathsf{Extract}(P', \mathsf{sid}, g, X_2, \pi_2)$
$x_\beta \leftarrow\$ \mathbb{Z}_q$; $\beta \leftarrow (X_1X_2X_3)^{x_\beta}$
$\pi_\beta \leftarrow\$ \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, (P', \mathsf{sid}), (X_1X_2X_3, \beta))$
$T_p[\beta] \leftarrow (P', \mathsf{sid}, X_1X_2X_3, x_\beta, \pi_\beta)$
$\pi_{P'}^{\mathsf{sid}} \leftarrow (P', P, X_1, X_2, X_3, X_4, \bot, \beta, S)$
Send $(P', P, \mathsf{sid}, X_3, \pi_3, X_4, \pi_4, \beta, \pi_\beta)$ to $\mathcal{A}$

On $(P, P', \mathsf{sid}, \alpha, \pi_\alpha)$ from $\mathcal{A}$:
If $\pi_{P'}^{\mathsf{sid}} \neq (P', P, X_1, X_2, X_3, X_4, \bot, \beta, S)$ ignore input
If $\mathsf{NIZK.Ver}(\mathsf{crs}, (P, \mathsf{sid}), (X_1X_3X_4, \alpha), \pi_\alpha) = \bot$
  then $\pi_{P'}^{\mathsf{sid}} \leftarrow \bot$; return
$K \leftarrow\$ \mathcal{K}$
If $\pi_P^{\mathsf{sid}} = (P, P', X_1, X_2, X_3, X_4, \alpha, \beta, C)$
  (Passive attack; do not interrupt)
Else If $\pi_P^{\mathsf{sid}} = (P, P', X_1, X_2, X_3, X_4, \alpha, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P, P', X_2, X_1, X_3, X_4, \alpha, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P, P', X_1, X_2, X_4, X_3, \alpha, \beta, S)$ or
  $\pi_{P'}^{\mathsf{sid}} = (P, P', X_2, X_1, X_4, X_3, \alpha, \beta, S)$
  (Swap attack; interrupt without pw)
  Query $\mathsf{TestPw}(P', \mathsf{sid}, \mathsf{pw} = \bot)$
Else
  (Active attack; we need to extract a password)
  Find $(X_1, P, \mathsf{sid}, \hat{g}, x_1, \pi_1)$ in $T_e \cup T_p$
  Find $(X_2, P, \mathsf{sid}, \hat{g}, x_2, \pi_2)$ in $T_e \cup T_p$
  $x_\alpha \leftarrow \mathsf{Extract}(P, \mathsf{sid}, X_1X_3X_4, \alpha, \pi_\alpha)$
  If $X_1 \neq g^{x_1} \vee X_2 \neq g^{x_2}$ then ABORT
  If $\alpha \neq (X_1X_3X_4)^{x_\alpha}$ then ABORT
  Find $(X_4, P, \mathsf{sid}, \hat{g}, x_4, \pi_4)$ in T_p
  Query $\mathsf{TestPw}(P', \mathsf{sid}, \mathsf{pw} = x_\alpha/x_2)$
  If correct guess $K \leftarrow \mathsf{PRF}((\beta X_4^{-x_2\mathsf{pw}})^{x_2}, \mathsf{sid})$
Query $\mathsf{NewKey}(P', \mathsf{sid}, K)$

Fig. 4: The simulator $\mathcal{S}$.

Definition 4.2 (Matching Sessions). *Take a client (resp. server) session with view* $(X_1, X_2, X_3, X_4, \alpha, \beta)$ *(resp.* $(X'_1, X'_2, X'_3, X'_4, \alpha', \beta')$*). We say they are* matching *if their views are identical. We say they have* matching bases *or are* b-matching *if their views match after the first two rounds, i.e., if* $(X_1, X_2, X_3, X_4) = (X'_1, X'_2, X'_3, X'_4)$.

Definition 4.3 (Swapped Sessions). *Take a client (resp. server) session with view* $(X_1, X_2, X_3, X_4, \alpha, \beta)$ *(resp.* $(X'_1, X'_2, X'_3, X'_4, \alpha', \beta')$*). We say they are* swapped *if* $(\alpha, \beta) = (\alpha', \beta')$*,* $(X_1, X_2, X_3, X_4) \neq (X'_1, X'_2, X'_3, X'_4)$*,* $\{X_1, X_2\} = \{X'_1, X'_2\}$ *and* $\{X_3, X_4\} = \{X'_3, X'_4\}$*. We say they have* swapped bases *or are* b-swapped *if their views are swapped after the first two rounds, i.e., if* $(X_1, X_2, X_3, X_4) \neq (X'_1, X'_2, X'_3, X'_4)$*,* $\{X_1, X_2\} = \{X'_1, X'_2\}$ *and* $\{X_3, X_4\} = \{X'_3, X'_4\}$.

Hop 1 Modify the real-world so that i. uncorrupted parties use simulated NIZK proofs, and ii. all adversarially generated proofs are subject to extraction. If extraction fails, the experiment aborts. We can reduce any distinguishing advantage wrt the original game to breaking the security of the NIZK protocol. From this point on, observe that both the modified real-world and the ideal-world (via the simulator) explicitly compute the discrete logarithms of all messages exchanged in all protocol sessions, or else they abort.

Hop 2 We introduce various book-keeping modifications:

- Modify both the real and the ideal games to abort if a value X_i, for $1 \leq i \leq 4$, is generated by an uncorrupted party and this collides with some other such value that previously occurred in the game.
- Modify both the real and the ideal worlds to guarantee that the bases used in the computation of α and β are binding in the following sense, Take a client session with a partial view (X_1, X_2, X_3, X_4) and a server session with a partial view (X'_1, X'_2, X'_3, X'_4), and abort if one of the two following events occur:
  - the two sessions agree on the partial view $(X_1X_3X_4, X_2) = (X'_1X'_3X'_4, X'_2)$ and they are neither b-matched nor b-swapped.
  - the two sessions agree on the partial view $(X_1X_2X_3, X_4) = (X'_1X'_2X'_3, X'_4)$ and they are neither b-matched nor b-swapped.

  Intuitively, this means that the environment cannot force the simulator to extract a password guess from an α or β that was produced by another simulated session (recall that the simulator does not try to extract passwords when sessions are matched or swapped).
- Abort if ever α or β is accepted that would cause two sessions to become swapped, but the derived key is different from the key of equivalent un-swapped traces. Note that the derived key is unaffected in the server-side if $(X_1, X_2, X_3, X_4) = (X'_2, X'_1, X'_3, X'_4)$. Similarly the derived key is unaffected in the client-side if $(X_1, X_2, X_4, X_3) = (X'_1, X'_2, X'_3, X'_4)$. More precisely, abort if:
  - α is accepted that comes from b-swapped session where $(X_1, X_2) = (X'_2, X'_1)$, or
  - β is accepted that comes from b-swapped session where $(X_3, X_4) = (X'_4, X'_3)$.

  Intuitively this means that, when dealing with swapped sessions from now on, the derived key always matches the unswapped equivalent.

This hop introduces a bad-event-based hop wrt to both worlds, which is upper-bounded by excluding collisions in simulated values using a birthday bound and a reduction to the discrete logarithm problem.[6]

Hop 3 Modify the real-world once more to use a random group element as the PRF key in any actively attacked sessions where the extracted password was incorrect. Furthermore, perform the same modification in matched or swapped sessions where the password is different on client and server. This step can be reduced to decisional squared Diffie-Hellman using a hybrid argument across the client and server sessions.

[6] Note that this step is necessary because in future game hops we will be embedding hard problem instances in the messages sent by uncorrupted parties in order to justify that the output keys look random; in these reductions password extraction might fail if the bad event above was not excluded. Interestingly, this problem is not evident from the simulator code directly: in fact, the simulator does know the discrete logarithms used for all simulated messages as we pointed out above.

Hop 4 Modify the real-world once more to use a random group element as the PRF key in any session that was accepted after a passive attack, or where the adversary just performed swapping, with matching passwords. This step can be reduced to DDH using a hybrid argument across all possible pairs of sessions (i.e., we get a squared term in the reduction to the underlying assumption). For simplicity, each reduction step actually uses an instance of a hard problem called decision triple group Diffie-Hellman, which is hard iff DDH is hard.

Hop 5 Randomize all α and β generated by uncorrupted parties in the real world and to compute the output keys for the corresponding sessions, when correct password guesses occur, as the adversary would. This step can be reduced to DDH, again using a hybrid argument across all sessions.

Hop 6 Replace all PRF outputs that are computed from independent random keys resulting from the previous hops with random strings. This follows from PRF security.

After all these modifications we have a perfect match between real and ideal worlds. In Appendix A we give a more detailed description of each step in the proof, and we also discuss how the proof covers the case of static corruptions.

5 Game-Based Security Model

We prove the sJ-PAKE protocol secure in the Real-or-Random (RoR) model [AP05] where security is defined via games and the goal of $\mathcal{A}$ is to guess the bit b (chosen at the beginning of each game) i.e. to distinguish *real* session keys from *random* strings. Unlike its weaker variant, Find-then-guess (FtG) model [BPR00], the adversary $\mathcal{A}$ is allowed to ask multiple Test queries during the security experiment.

Participants and Passwords. Each participant, denoted Π_U^i, is either a client $C \in \mathcal{C}$ or a server $S \in \mathcal{S}$. Each client C holds a password pw_C, while the server S holds a vector of passwords $\mathsf{pw}_S =< \mathsf{pw}_C >_{C \in \mathcal{C}}$, such that $\mathsf{pw}_S[C] = \mathsf{pw}_C$ for all $C \in \mathcal{C}$. For simplicity, we assume that passwords are independent and uniformly distributed, drawn from the password dictionary D.

User instances. In the real world, each user is allowed to run simultaneously more than one protocol execution, which is modelled by allowing each user an unlimited number of *instances*. Specifically, let Π_C^i and Π_S^j denote the i-th and j-th instance of client C and server S respectively.

Protocol execution and initialization phase. The protocol P is an algorithm which defines how users respond to messages from their environment. We allow each instance to execute P an unlimited number of times with as many other instances as it wishes. Formally, a bit is flipped, at the beginning of the protocol. We introduce a PPT adversary $\mathcal{A}$, that has full control of the network and communicates with each instance according to the rules of P via the following queries:

Send(U, i, m): A message m is sent to instance Π_U^i, which proceeds according to P and its response – if any – is given to $\mathcal{A}$. This query models an active adversary.

Execute(C, i, S, j): This query triggers an honest run of protocol P between Π_C^i and Π_S^j. The transcript of the execution is given to $\mathcal{A}$. This query models a passive adversary.

Reveal(U, i): When $\mathcal{A}$ triggers this query it receives a session key held by Π_U^i.

Test(U, i): At the beginning of the experiment, the challenger flips a coin and sets a bit b outside of the view of $\mathcal{A}$. Then, whenever $\mathcal{A}$ asks this query to some instance Π_U^i, it obtains a following response:

1. If $b = 1$, $\mathcal{A}$ gets the real session key sk_U^i.
2. Otherwise, $\mathcal{A}$ gets a random string $r \xleftarrow{\$} \{0,1\}^\kappa$. For consistency, if two partnered instances, Π_C^i and Π_S^j, receive a Test query, then $\mathcal{A}$ gets the same random string.

Corrupt(C, S): $\mathcal{A}$ receives a password pw_{cs} shared between a pair of instances, Π_C^i and Π_S^j.

Partnered instances. Two instances, Π_C^i and Π_S^j are *partnered* if both accept, holding the same partner identity (pid), transcript (sid) and the same session key (sk). More precisely, a client with $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, \mathsf{sk}_C^i)$ and a server with $(\mathsf{pid}_S^j, \mathsf{sid}_S^j, \mathsf{sk}_S^j)$ are partnered if:

1. $\mathsf{sid}_C^i = \mathsf{sid}_S^j$, $\mathsf{sk}_C^i = \mathsf{sk}_S^j$, $\mathsf{pid}_C^i{=}S$, $\mathsf{pid}_S^j{=}C$.
2. No other instance accepts with the same sid, except with the negligible probability.

We say an instance Π_U^i *accepts* if it holds a session key sk, transcript sid and partner id pid. A client instance can accept at most once. Two instances *terminate* if they accept and do not wish to send any further messages.

Freshness. We define *freshness* to avoid cases where $\mathcal{A}$ might trivially know the bit b, chosen at the beginning of the protocol. An instance Π_U^i in protocol P is fresh if and only if the following conditions hold:

- the instance was not queried to Test or Reveal before;
- the instance accepted;
- either the instance accepted during a query to $\mathsf{Execute}$ or there exists more than one partner instance or no partner instance exists and $\mathsf{Corrupt}$ was not called before it accepted or a unique fresh partner instance exists

We define a function $\mathsf{Fresh}(P, i)$ which checks if an instance is fresh by checking all the conditions above.

Advantage of the adversary. We formally define the advantage of the adversary in successfully guessing the bit b, against the protocol P. Let $\mathrm{Succ}_{\mathrm{P}}^{\mathrm{RoR}}(A)$ be the event that $\mathcal{A}$ asks only Test queries to instances Π_U^i that have terminated; at some point $\mathcal{A}$ outputs its guess b' and *wins* if $b' = b$, where b is selected at the beginning of the protocol. The advantage of $\mathcal{A}$ in breaking the security and guessing b is

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{RoR}}() = 2\Pr\left[\mathrm{Succ}_{\mathcal{A}}^{\mathrm{RoR}}()\right] - 1$$

The protocol P is considered to be secure if the adversary $\mathcal{A}$ cannot perform any other attack than just online password guessing during an active attack. More formally, we require that for all PPT $\mathcal{A}$:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{RoR}}() \leq \frac{n_{se}}{|D|} + negl(\kappa)$$

where n_{se} is number of Send queries, D is the password dictionary and $negl(\cdot)$ is a negligible function.

6 Game based security proof of sJ-PAKE

Theorem 6.1. *Let sJ-PAKE be the protocol described in Fig. 1. Take an RoR attacker $\mathcal{A}$ against sJ-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te} queries to* Send, $\mathsf{Execute}$, Reveal, $\mathsf{Corrupt}$, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that*

$$\begin{aligned}\mathrm{Adv}_{\mathcal{A}}^{\mathtt{sJPAKE}}() \leq\ & \frac{2n_{se}}{|\,D\,|} + \frac{(2n_{se}+4n_{ex})^2}{q} \\ & + \mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{uzk}}() + 2n_{se}\mathrm{Adv}_{\mathsf{NIZK}}^{\mathrm{ext}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}_{\mathcal{B}_4}^{\mathtt{CTGDH}}() + \mathrm{Adv}_{\mathcal{B}_{4.5.1}}^{\mathtt{DDH}}() + \mathrm{Adv}_{\mathcal{B}_{4.5.2}}^{\mathtt{DDH}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}_{\mathcal{B}_6}^{\mathtt{CSqDH}}() + 2n_{se}\mathrm{Adv}_{\mathcal{B}_{7-8.1}}^{\mathtt{DSqDH}}() + n_{se}n_{ro}\mathrm{Adv}_{\mathcal{B}_9}^{\mathtt{CSqDH}}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK, formally defined in 3.6.*

The code of all games and adversaries is given as Supplemental material E.

Proof. The proof uses a sequence of games $G_0, G_1, \ldots, G_9$. We follow [BPR00] by including initialize and finalize oracles; the adversary is only allowed to call *Initialize* as its first query, and *Finalize* as its final query; games compute their final results as a result of the call to *Finalize*.

The main challenge in the proof with respect to that in [ABM15] is that we cannot extract the discrete logarithms of adversarial α and β, since ZK-PoK proofs are removed for these elements in the simplified protocol. Instead we must use the RO queries to extract relevant information about the adversaries actions and, in particular, to detect when the adversary is successful in a password guess attack.

The proof strategy is typical of PAKE protocols. We first modify the security game gradually until the view of the adversary is independent of the passwords used in fresh sessions, which are the ones in which it can obtain an advantage. Then bound the number of password guesses that the attacker can make to win the game: in the random oracle model, this comes down to bounding the number of RO queries that can be consistent with the trace of a fresh session. In the case of sJ-PAKE we show that, unless the adversary solves a hard problem, only one RO query will satisfy this consistency constraint. This means that the total number of useful password guesses is n_{se} and hence that the adversary's guessing advantage is at most $\frac{n_{se}}{D}$, for some dictionary D. Here, we leave a remark that the proof can be extended even with passwords that have some min-entropy m [ABM15] in which case, the adversary's guessing advantage is at most $\frac{n_{se}}{2^m}$. We give an overview of the sequence of games in Fig. 5.

Useful notation. For each client-server pair (C, S), the game chooses a password uniformly sampled from a password space $\mathcal{P}$, denoted pw_{cs}. Out of simplicity, we assume that game samples the passwords uniformly and independently. We use notation $T[x]$ to denote access to a dictionary/table T at index x.

- Table T consists of random oracle queries $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$, where a query was assigned to a random value from the game. If there was a query that T does not contain, then the game assigns a new random value and records it, otherwise it returns the value that was previously assigned to that query.

- Table T_s consists of random oracle queries sid, without including the Key and the pw. It is just a new simulation of T where we replace one random value with another. We call T_s for sessions where the adversary is active.

- Table T_e consists of random oracle queries $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$. We use T_e for sessions where the adversary is passive i.e. for $\mathsf{Execute}$ queries.

- List Corr is consisted of corrupted instances and whenever a corruption occurs, we add (C, S) the the list. There are no fresh instances on the list Corr.

- List Tst is consisted of all instances for which $\mathcal{A}$ queried Test query.

Matching sessions. We define matching sessions whenever two honest instances, π_C^i and π_S^j, accept with the same session transcript sid. More precisely, all sessions resulting from $\mathsf{Execute}$ query or matching Send query where there was no Send query from $\mathcal{A}$ for that session, are matching sessions. Note, we consider all sessions resulting from $\mathsf{Execute}$ query fresh.

Instance state. Each ith instance is denoted as π_U^i has a tuple $(e, \mathsf{sid}, K, \mathsf{ac})$ that describes:

- e is a pair of the dlogs of X_l and X_{l+1}, (x_l, x_{l+1}) for $l = 1, 3$.
- sid is a session transcript of the form $(C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$.
- K is the accepted session key.
- ac is a boolean value that indicates whether the instance accepted $(\mathsf{ac} = \mathsf{T})$ or not $(\mathsf{ac} = \mathsf{F})$.

Further in the proof, we will use $\pi_U^i.e$, $\pi_U^i.\mathsf{sid}$, $\pi_U^i.\mathsf{K}$, $\pi_U^i.\mathsf{ac}$ as the individual component of the state.

G_0 **Original Protocol**: The original protocol P.
G_1 **Simulate ZK-PoK proof**s: For Send and Execute query use NIZK simulator to simulate proof of knowledge. bad_1 is set whenever adversary detects the difference between the simulated and real proofs.
$G_{1.5}$ **Extract discrete logs from adversarial proofs**. For Send queries use NIZK extractor to extract witnesses x_1', x_2', x_3', x_4'. $\mathsf{bad}_{1.5}$ is set whenever adversary submits the proof for which extraction fails.
G_2 **Force unique values**: Repetition of any X values selected from the game are not allowed. bad_2 is set whenever values repeat and is bounded by the statistical term.
G_3 **Freshness condition**: Having unique sid makes the freshness condition explicit in the game. Therefore, instead running function Fresh and checking all conditions from freshness definition, we check for freshness in every Send query and we add fr in each state of the instance.
G_4 **Randomize session keys for Execute queries**: Session keys are randomized by calling T_e, for passive adversaries, and bad_4 is set whenever there was a query in which correct Key has been computed using pw_{cs}. The probability of bad_4 happening is reduced to **CTGDH** problem.
$G_{4.5.1}$ **Randomize α for Execute queries**: α is randomized for all passive adversaries, and the advantage of the adversary is reduced to **DDH** problem.
$G_{4.5.2}$ **Randomize β for Execute queries**: β is randomized for all passive adversaries, and the advantage of the adversary is reduced to **DDH** problem.
G_5 **Randomize session keys for Send queries**: Session keys are randomized by calling T_s, for active adversaries and bad_5 is set whenever there was a query in which correct Key has been computed using pw_{cs}. bad_5 is not bounded.
G_6 **Detect duplicates**: bad_6 is a sub-event of bad_5 and is set whenever there are two queries with the same sid for $\mathsf{pw}_1 \neq \mathsf{pw}_2$. bad_5 is not bounded and bad_6 is reduced to **CSqDH** problem.
G_7 **Add algebraic representation**: In Send queries with α and β, we add algebraic representation of α and β. bad_5 is not bounded.
G_8 **Randomize α and β** for Send queries: Algebraic adversaries are introduced and hybrid argument is used and the game is split in two hops $G_{7-8.m.1}$ and $G_{7-8.m.2}$, for $1 \leq m \leq n_{se}$.

- $G_{7-8.m.1}$: We tackle the bad scenarios of α and β, for Corrupt queries. The advantage of the adversary is reduced to **DSqDH**.

- $G_{7-8.m.1}$: α and β are randomized.

G_9 **Perfect forward secrecy.** There are new entries in bad_5, where $\mathcal{A}$ asks for Corrupt query after the instances accept but before it asks a random oracle query. bad_9^1 may occur if there are new entries where $\mathcal{A}$ tests $\mathsf{pw} \neq \mathsf{pw}_{cs}$ in β' (resp. α') and asks a query with the correct key. If there was a query with the correct key and $\mathcal{A}$ tests $\mathsf{pw} = \mathsf{pw}_{cs}$ then bad_9^2 is set. bad_5 is bounded by the term $\frac{n_{se}}{|D|}$, bad_9^1 is reduced to **CSqDH** problem and bad_9^2 is bounded by $\frac{n_{se}}{|D|}$.

Fig. 5: Description of game-hops for sJ-PAKE

Game 0: Original Protocol. The first game is the original security game instantiated with sJ-PAKE, so we have

$$\mathrm{Adv}^{\mathtt{sJPAKE}}_{\mathcal{A}}() = \mid \Pr[G_0 \Rightarrow \mathsf{T}] - \frac{1}{2} \mid \tag{1}$$

Game 1: Simulate ZK-PoK proofs. For SendInit-C1,SendInit-S1 and Execute queries we use SE-NIZK to simulate the *proofs of knowledge* π_1, π_2, π_3, π_4 for X_1, X_2, X_3, X_4 .

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() \tag{2}$$

Game 0 and Game 1 are the same, unless the adversary distinguishes real from simulated proofs and it does it with the advantage $\mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}$. We show a reduction for this game in Lemma B.1. We particularly have a use of simulated proofs of knowledge in the reductions, where we will not know the witness for a given challenge. In this game, we additionally create a list List, where we add only values generated from honest instances, X_1 and X_2 from the client side, and X_3 and X_4 from the server side. We need List to make a clear distinction between the values coming from honest instances and values coming from $\mathcal{A}$. This is necessary for Game 1.5, where we will run $\mathsf{NIZK.Extract}$ only for adversarial values. In Game 2, we will insist on uniqueness only on the values X_1, X_2, X_3 and X_4 that are in List.

Game 1.5: Extract discrete logs from adversarial proofs. For Send-C2 and Send-S2 queries, whenever $\mathcal{A}$ outputs a proof π for a generated X, we run $\mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X, \pi, l)$ which extracts a witness x for X, from a valid proof of knowledge π, considering label l (C or S), trapdoor td_e and CRS crs. If the extraction fails, we set $\mathsf{bad}_{1.5}$. Game 1 and Game 1.5 are the same unless $\mathsf{bad}_{1.5}$ occurs.

$$\mid \Pr[G_1 \Rightarrow \mathsf{T}] - \Pr[G_{1.5} \Rightarrow \mathsf{T}] \mid \leq \Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}] \tag{3}$$

The reduction in B.2, to justify this hop, guesses a pair (X, π) where the first $\mathsf{bad}_{1.5}$ happens and, when this event occurs it submits (X, π). The probability that it chooses the correct pair where the extraction failed with the advantage $\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}$ can be bound:

$$\Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}] \leq 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}()$$

Game 2: Force unique values
Repetition of any values X_1, X_2, X_3 or X_4, previously seen is an execution is not allowed. We set bad_2 whenever values, X_1, X_2, X_3 or X_4, repeat. Game 2 is the same as Game 1, unless bad_2 occurs. More precisely, we have two values for SendInit-C1 and SendInit-S1, four values for $\mathsf{Execute}$ queries and the size of the group q:

$$\Pr[G_{1.5} \Rightarrow \mathsf{T}] - \Pr[G_2 \Rightarrow \mathsf{T}] \leq \frac{(2n_{se} + 4n_{ex})^2}{q} \tag{4}$$

This means that $2n_{se} + 4n_{ex}$ is a maximal number of values that can repeat and we can bound the bad event by the birthday paradox bound in case X_1, X_2, X_3 or X_4 collide. In case of a bad event, we mark an instance for which values repeat, as "Invalid".

Game 3: Adding freshness.
Now that we have unique sessions and we add explicit freshness conditions for all sessions. These are initially false, and then set to their correct value at the time the session accepts in Send-C3 and for Send-S3. A fresh session may later become unfresh if its key is given to the adversary as a result of a Reveal query. We also remove all bad events. This change does not affect the view of the adversary, so we have:

$$\Pr[G_3 \Rightarrow \mathsf{T}] = \Pr[G_2 \Rightarrow \mathsf{T}] \tag{5}$$

Game 4: Randomize session keys for matching sessions.
For matching sessions we no longer use the random oracle to derive the key, and use a totally random key instead. We keep these removed entries in a new list T_e. If at any point a random oracle query is placed that could cause an inconsistency in the adversary's view, we set a bad flag bad_4. Note, for Send queries, we check if the sessions are matching in Send-S3 and Send-C3.

$$|\Pr[G_3 \Rightarrow \mathsf{T}] - \Pr[G_4 \Rightarrow \mathsf{T}]| \leq \Pr[G_4 \Rightarrow \mathsf{bad}_4] \tag{6}$$

The reduction to justify this hop, guesses the lth Execute query where the first bad_4 happens and, when this event occurs, it solves an instance of the Computational Triple Group problem, CTGDH, so we have:

$$\Pr[G_4 \Rightarrow \mathsf{bad}_4] \leq n_{ro} n_{ex} \mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Game 4.5: Randomize alpha/beta for Execute queries.
We randomize α and β for sessions resulting from Execute queries using two intermediate games where we first randomize α in Game 4.5.1, then β in Game 4.5.2.

Game 4.5.1 Randomize α
We randomize α in sessions resulting from Execute queries. Game 4.5.1 is indistinguishable from Game 4 unless $\mathcal{A}$ solves DDH problem. The reduction is shown in Lemma B.4.

$$|\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}]| \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

Game 4.5.2 Randomize β
Now, we randomize β in sessions resulting from Execute queries. Game 4.5.2 is indistinguishable from Game 4.5.1 unless $\mathcal{A}$ solves DDH problem. The reduction is shown in Lemma B.5.

$$|\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}]| \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Game 5: Randomize session keys for Send queries.
We randomize session keys for all Send queries, i.e. for active attacks. Concretely, we no longer use the random oracle to compute the output key of such sessions, when they are fresh, and use a totally random key instead. We introduce a table T_s to keep track of the inputs excluded from the random oracle input. In the case of a random oracle query that could cause an inconsistency in the adversary's view, we set a bad flag bad_5. We naturally have:

$$|\Pr[G_{4.5.2} \Rightarrow \mathsf{T}] - \Pr[G_5 \Rightarrow \mathsf{T}]| \leq \Pr[G_5 \Rightarrow \mathsf{bad}_5]$$

Game 6: Detect duplicates.
Rather than checking for bad_5 in the game we construct a list of random oracle queries that would cause bad_5 and, at the end of the game we divide the checking for bad_5 into two sub-events. We set a new bad flag if ever there are two entries in the list of problematic random oracle queries that are consistent with the same sid, i.e., if there exist queries $(\mathsf{sid}, \mathsf{Key}_1, \mathsf{pw}_1)$ and $(\mathsf{sid}, \mathsf{Key}_2, \mathsf{pw}_2)$, where Key_1 and Key_2 are computed correctly, for $\mathsf{pw}_1 \neq \mathsf{pw}_2$. Such pair of queries we call *duplicates* and whenever we detect them, we set bad_6. In this event does not occur and the list of problematic queries is not empty, we set bad_5. In this game by bounding bad_6, we restrict the set of executions where bad_5 may occur, so we have:

$$\Pr[G_5 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_6 \Rightarrow \mathsf{bad}_5] + \Pr[G_6 \Rightarrow \mathsf{bad}_6]$$

The reduction for this game, guesses the rth session where the first bad_6 happens, together with the indices of the two problematic random oracle queries, and reduces the bad event to the Computational Square Diffie-Hellman problem, CSqDH. This implies:

$$\Pr[G_6 \Rightarrow bad_6] \leq n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

The reduction is shown in Lemma B.6.

Game 7: Add algebraic representation.
In this hop we keep the check for bad_5 as before, and we introduce algebraic adversaries by adding algebraic representation whenever there is Send query with α or β. Therefore, we have:

$$\Pr[G_6 \Rightarrow \mathsf{bad}_5] = \Pr[G_7 \Rightarrow \mathsf{bad}_5]$$

Game 8: Randomizing α and β for Send queries.

$$\Pr[G_7 \Rightarrow \mathsf{bad}_5] - \Pr[G_8 \Rightarrow \mathsf{bad}_5] \leq 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

We use a hybrid argument to randomize α and β for all Send queries, one session at the time. This means, if we consider the mth session, then all sessions prior to mth will have random α and β, and for all sessions after mth, we will compute α and β normally. We cannot randomize messages with α and β in one go, because of the following scenario. Let us say $\mathcal{A}$ communicates with a client instance. In the second round, the Client chooses α randomly and sends it to $\mathcal{A}$. Then $\mathcal{A}$ decides to corrupt and send β' as $\beta' = X_4'^{x_2\mathsf{pw}_{cs}}g^s$, where s is something that only $\mathcal{A}$ knows. Here, we face a problem because $\mathcal{A}$ can fix any s it wants and our reduction will not be able to detect what $\mathcal{A}$ is doing and respond with appropriate key. For this reason we assume we are dealing with algebraic adversaries, meaning, that whenever $\mathcal{A}$ outputs β', it also outputs alg' that gives the representation of β' in terms of other group elements it observed during the game. These group elements are g, and honestly generated X_1, X_2 and α. This analysis means that any group element that $\mathcal{A}$ produces can be rewritten as a representation of the form $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$. We start the change for mth session, for $1 \leq m \leq n_{se}$ and we split the Game 8 into two games: Game $7 - 8.m.1$ where $\mathcal{A}$ sends β' with alg', we compute α honestly and use alg' to decompose β' and Game $7-8.m.2$, where we randomize α.

Game 7-8.m.1 We make the change in all instances before mth session, and for all instances after mth, we simulate as in Game 7. Let us say, algebraic adversary corrupts the session with an honest client instance right after it receives an honestly computed α, i.e. $\alpha = (X_1X_3'X_4')^{x_2\mathsf{pw}_{cs}}$, but before sending β'. Then $\mathcal{A}$ sends β' with $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$, and we use Rewrite that rewrites β' as $\beta' = g^aX_1^bX_2^c\alpha^d$. Then, we reduce the construction of $\mathsf{Key} = (\frac{\beta'}{X_4'^{x_2\mathsf{pw}_{cs}}})^{x_2}$ by checking two cases:

(a) In case of Corrupt, if $X_2^c\alpha^d \neq X_2^{x_4'\mathsf{pw}_{cs}}$, then $\mathcal{A}$ gets a random key. Furthermore, $\mathcal{A}$ needs to solve a Decisional Square Diffie-Hellman problem, DSqDH to compute the key.
(b) In case of Corrupt, if $X_2^c\alpha^d = X_2^{x_4'\mathsf{pw}_{cs}}$, then the we compute the real key, i.e. $\mathsf{Key} = (\frac{\beta'}{X_4^{x_2\mathsf{pw}_{cs}}})^{x_2}$. Furthermore, in case ($b$), after rearranging the exponents we get $\mathsf{Key} = g^{(x_2a+x_1x_2b)} = g^{x_2a}g^{x_2x_1b}$. Now, the final thing we need to do in (b), is to embed α in the Key to prepare for the next hop, and we get $\mathsf{Key} = g^{x_2a}(\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{g^{x_2(x_3'+x_4')}})^b$. We tackle the keys for fresh sessions as before. We stress that the only change in Game $7 - 8.m.1$ is in case of corruption in Send-C3 or Send-S3. If that does not happen we simulate everything as in Game 7.

$$|\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}]| \leq 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

We reduce the advantage of the adversary in Game 7-8.m.1 to DSqDH, shown in Lemma B.7.

Game 7-8.m.2. For all instances before mth, we make the change and for all instances after mth, we simulate as in Game 7-8.m.1. In this hop we randomize α.

$$|\Pr[G_{7-8.m.1} \Rightarrow T] - \Pr[G_{7-8.m.2} \Rightarrow T]| \leq \Pr[G_8 \Rightarrow \mathsf{bad}_5]$$

In this game, the test bit b is perfectly hidden as in fresh sessions so the adversary has zero advantage in noticing that α is randomized. Game 7-8.m.1. and Game 7-8.m.2. are the same unless bad_5 occurs. We stress that we do the same hybrid analysis for randomizing β.

Game 9: Perfect forward secrecy.
In this hop, it remains to bound bad_5 and address the cases where $\mathcal{A}$ asks for $\mathsf{Corrupt}$ query after the instances accept but before it asks a random oracle query. Everything is randomized in Send and $\mathsf{Execute}$, therefore we can delay password sampling until the end or in case of corruption. Now, among new entries that might cause bad_5, there could be entries where $\mathcal{A}$ sends a message with β' (resp. α'), where it tested a different password, $\mathsf{pw}' \neq \mathsf{pw}_{cs}$ and then later asked a query with pw_{cs}. In that case we set a bad flag bad_9^1. Now, it might also happen that $\mathcal{A}$ sent β' (resp. α') where $\mathsf{pw}' = \mathsf{pw}_{cs}$. In that case we set a bad flag bad_9^2. More precisely, we split bad_5 after corruption into bad_9^1 and bad_9^2. Let us say, $\mathcal{A}$ sent β' with alg' and then accepted. Then, $\mathcal{A}$ corrupts and asks for a random oracle query $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$. Thanks to the algebraic adversary, we can use alg of β' it produced in terms of g, X_1, X_2 and α which it observed in the game. These elements are computed honestly, $X_1 = g^{x_1}$, $X_2 = g^{x_2}$ and $\alpha = g^w$ by the game. In another words, β' can be rewritten as a representation of the form $\mathsf{alg}' = [(g^a), (X_1^b), (X_2^c), (\alpha^d)]$. Therefore, we use function $\mathsf{Rewrite}$ to decompose β' to $\beta' = g^a X_1^b X_2^c \alpha^d$. Then, in case of corruption, we reduce the construction of $\mathsf{Key} = (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$ by checking two cases:

(a) In case of $\mathsf{Corrupt}$, if $g^{x_2 c} \neq X_4^{x_2 \mathsf{pw}_{cs}}$ then $\mathsf{pw}' \neq \mathsf{pw}_{cs}$ and we set bad_9^1. In this case, $\mathcal{A}$ needs to solve a Computational square Diffie-Hellman problem, CSqDH to compute the correct key.

(b) In case of $\mathsf{Corrupt}$, if $g^{x_2 c} = X_4^{x_2 \mathsf{pw}_{cs}}$, we set bad_9^2. It means that $\mathsf{pw}' = \mathsf{pw}_{cs}$ and $\mathcal{A}$ submitted the winning query. To compute the key, the expressions cancel and we get $\mathsf{Key} = g^{(x_2 a + x_1 x_2 b + x_2 w d)} = g^{x_2(a + x_1 b + w d)}$.
We check bad_5 as before. Note that bad_9^1 and bad_9^2 are disjoint. So, we have the following bound

$$\Pr[G_8 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_9 \Rightarrow \mathsf{bad}_5] + \Pr[G_9 \Rightarrow \mathsf{bad}_9^1] + \Pr[G_9 \Rightarrow \mathsf{bad}_9^2]$$

The entries where bad_5 might happen are in the list which size is now at most n_{se}, so we have:

$$\Pr[G_9 \Rightarrow \mathsf{bad}_5] \leq \frac{n_{se}}{\mid D \mid}$$

We reduce the probability of bad_9^1 to CSqDH and we show the reduction in Lemma B.8, where we plug in the challenge for lth session, and guess the entry for which bad_9^1 might have occurred. For this reason we have

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq n_{se} n_{ro} \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()$$

Finally, the number of remaining entries that can cause bad_9^2 is at most $\frac{n_{se}}{|D|}$, so we have

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq \frac{n_{se}}{\mid D \mid}$$

This completes the proof and yields the bound.

$$\begin{aligned}
\mathrm{Adv}^{\mathtt{sJPAKE}}_{\mathcal{A}}() \leq \; & \frac{2n_{se}}{\mid D \mid} + \frac{(2n_{se} + 4n_{ex})^2}{q} \\
& + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\
& + n_{ro} n_{ex} \mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\
& + n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se} n_{ro} \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()
\end{aligned}$$

□

7 Efficiency analysis of J-PAKE, sJ-PAKE and all its variants

In this section, we only focus on J-PAKE and its variants and provide a concrete analysis of complexity that also comes out as a result of this paper. Conversely, a comparison of J-PAKE with other known PAKE

protocols is shown in [ABM15]. We mainly rely on empirical results and performance analysis of J-PAKE, RO-J-PAKE, CRS-J-PAKE shown in [LST16] and convey the same analysis on sJ-PAKE, sRO-J-PAKE and sCRS-J-PAKE. We explicitly highlight the differences in computation costs, given in Table 1, and give the overall conclusion of which protocol(s) are the most efficient one(s). In terms of communication, J-PAKE, weighs 6 groups elements plus 12 ZK-PoK scalars in $\mathbb{Z}_q$ and 28 exponentiation in terms of computation which makes J-PAKE the least efficient among them all. Then, sJ-PAKE follows with 2 ZK-PoKs dropped in the second round, resulting in 22 exponentiation. RO-J-PAKE and CRS-J-PAKE drop 2 ZK-PoKs and 2 group elements in the first round, making a total of 20 exponentiation. Here, we stress that even though the difference in computation with sJ-PAKE is only two exponentiation, sJ-PAKE is a more desirable "lightweight" version, as it requires minimal changes to the current J-PAKE implementation. In contrast, the implementation of RO-J-PAKE and CRS-J-PAKE requires more changes due to additional assumption CRS for CRS-J-PAKE and additional cost of a hash H_0 for the RO-J-PAKE. Our final analysis includes sRO-J-PAKE and sCRS-J-PAKE. The total computation cost drops to only 14 exponentiations when dropping two ZK-PoKs in the first round (along with two group elements) and dropping two ZK-PoKs in the second round. Compared to J-PAKE, sRO-J-PAKE and sCRS-J-PAKE have a significant drop in exponentiation which makes the proof of sJ-PAKE beneficial as it suggests that sRO-J-PAKE and sCRS-J-PAKE can be proven secure in the same manner. We layout the security proof of sRO-J-PAKE and sCRS-J-PAKE in Sec. D. Furthermore, with the sJ-PAKE protocol and its security proof, there is a stronger motivation for implementing sRO-J-PAKE and sCRS-J-PAKE in practice.

Table 1: Comparison of Complexity of all J-PAKE(s) and and the corresponding assumptions used to prove their security against active attackers.

Protocol	**Complexity**		**Hardness Ass.**[1]	**Forward secrecy**
	Communication[2]	**Computation**		
J-PAKE	$6 \times \mathbb{G} + 12 \times \mathbb{Z}_q$	28 | p |-bit exp [3]	DSqDH	PFS[4]
RO-J-PAKE	$4 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	20 | p |-bit exp+$2H_0$[5]	DSqDH	PFS
CRS-J-PAKE	$4 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	20 | p |-bit exp	DSqDH	PFS
sJ-PAKE	$6 \times \mathbb{G} + 8 \times \mathbb{Z}_q$	22 | p |-bit exp	CSqDH	PFS
sRO-J-PAKE	$4 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	14 | p |-bit exp+$2H_0$	CSqDH	PFS
sCRS-J-PAKE	$4 \times \mathbb{G} + 4 \times \mathbb{Z}_q$	14 | p |-bit exp	CSqDH	PFS

[1] DSqDH stands for Decisional Square Diffie-Hellman, while CSqDH stands for Computational Square Diffie-Hellman
[2] $\mathbb{G}$ denotes a group element, $\mathbb{Z}_q$ a scalar
[3] exp. denotes exponentiation in $\mathbb{G}$. ZK-PoK costs three exponentiation each (one to create and two to verify)
[4] Perfect Forward Secrecy
[5] $H_0 : \{0,1\}^* \to \mathbb{G}$

7.1 Conclusion

We have presented a UC proof of the J-PAKE protocol, a significantly stronger result than previously available: allowing the exploitation of composition theorems of the UC framework. We also provide a proof of a lightweight version of J-PAKE, sJ-PAKE, that avoids the NIZK proofs of knowledge in the second round, thus significantly improving the efficiency both in terms of computation and communication. We explicitly show the challenging parts of the proof (Game 8 and Game 9) where we use algebraic adversaries to prove perfect forward secrecy. For now, it is not clear how to incorporate algebraic adversaries in the UC

setting, which justifies our choice for proving sJ-PAKE in Indistinguishability-based RoR model. At first sight, sJ-PAKE seems to fill the requirements to be proven secure in *relaxed* UC framework [ABB+20], but we leave it as a part of future work. Furthermore, we provide arguments that the efficiency gains of earlier lightweight variants of J-PAKE, RO-J-PAKE and CSR-J-PAKE, that reduce the number of NIZK proofs in the first round can be combined with those of sJ-PAKE, thus providing ultra-light variants, and we provide game-based proofs of security for these variants too.

References

AB19. M. Abdalla and M. Barbosa. Perfect forward security of SPAKE2. Cryptology ePrint Archive, Report 2019/1194, 2019. https://eprint.iacr.org/2019/1194.

ABB+20. M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu. Universally composable relaxed password authenticated key exchange. Cryptology ePrint Archive, Report 2020/320, 2020. https://eprint.iacr.org/2020/320.

ABM15. M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *2015 IEEE Symposium on Security and Privacy*, pages 571–587. IEEE Computer Society Press, May 2015.

AFP05. M. Abdalla, P. Fouque, and D. Pointcheval. Password-Based Authenticated Key Exchange in the Three-Party Setting. In *Public-Key Cryptography – PKC 2005*, *LNCS* 3386, pages 65–84. Springer, 2005.

AHH21. M. Abdalla, B. Haase, and J. Hesse. Security analysis of cpace. Cryptology ePrint Archive, Report 2021/114, 2021. https://eprint.iacr.org/2021/114.

AP05. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, *LNCS* 3376, pages 191–208. Springer, Heidelberg, February 2005.

BCP03. E. Bresson, O. Chevassut, and D. Pointcheval. The group Diffie-Hellman problems. In *SAC 2002*, *LNCS* 2595, pages 325–338. Springer, Heidelberg, August 2003.

BDZ03. F. Bao, R. Deng, and H. Zhu. Variations of diffie-hellman problem. In *ICICS 2003*, *LNCS* 2836, pages 301–312. Springer, 2003.

BFL20. B. Bauer, G. Fuchsbauer, and J. Loss. A classification of computational assumptions in the algebraic group model. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 121–151. Springer, Heidelberg, August 2020.

BM92. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *1992 IEEE Symposium on Research in Security and Privacy, SP 1992*, pages 72–84, 1992.

BMP00. V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *EUROCRYPT 2000*, *LNCS* 1807, pages 156–171. Springer, Heidelberg, May 2000.

BPR00. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000*, *LNCS* 1807, pages 139–155. Springer, Heidelberg, May 2000.

BR94. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO'93*, *LNCS* 773, pages 232–249. Springer, Heidelberg, August 1994.

BRRS18. J. Becerra, P. B. Rønne, P. Y. A. Ryan, and P. Sala. Honeypakes. In *Security Protocols XXVI - 26th International Workshop, Cambridge, UK, March 19-21, 2018, Revised Selected Papers*, *Lecture Notes in Computer Science* 11286, pages 63–77. Springer, 2018.

Can01. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

CHK+05. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, *LNCS* 3494, pages 404–421. Springer, Heidelberg, May 2005.

CK01. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT 2001*, *LNCS* 2045, pages 453–474. Springer, Heidelberg, May 2001.

DH76. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

FKL18. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 33–62. Springer, Heidelberg, August 2018.

GMR06. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In *Advances in Cryptology - CRYPTO 2006*, pages 142–159, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

Har15. D. Harkins. Dragonfly Key Exchange. https://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/, 2015.

HL19. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.

HR10. F. Hao and P. Ryan. J-PAKE: Authenticated Key Exchange without PKI. *Transactions on Computational Science*, 11:192–206, 2010.

HS14. F. Hao and S. Shahandashti. The speke protocol revisited. 12 2014.

HZ06. F. Hao and P. Zielinski. A 2-round anonymous veto protocol. In *Security Protocols, 14th International Workshop, Cambridge, UK, March 27-29, 2006, Revised Selected Papers*, *Lecture Notes in Computer Science* 5087, pages 202–211. Springer, 2006.

Jab96. D. P. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

Jab97. D. P. Jablon. Extended password key exchange protocols immune to dictionary attack, 1997.

JKX18. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks. In *Advances in Cryptology – EUROCRYPT 2018*, LNCS. Springer, 2018.

JR13. A. Juels and R. L. Rivest. Honeywords: making password-cracking detectable. In *ACM CCS 2013*, pages 145–160. ACM Press, November 2013.

KOY01. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001*, *LNCS* 2045, pages 475–494. Springer, Heidelberg, May 2001.

LST16. J. Lancrenon, M. Skrobot, and Q. Tang. Two more efficient variants of the J-PAKE protocol. In *ACNS 16*, *LNCS* 9696, pages 58–76. Springer, Heidelberg, June 2016.

Mac01. P. MacKenzie. On the Security of the SPEKE Password-Authenticated Key Exchange Protocol. Cryptology ePrint Archive, Report 2001/057, 2001. http://eprint.iacr.org/2001/057.

Mac02. P. D. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. Technical Report 2002-46, DIMACS, 2002.

Mer82. M. Merritt. Key reconstruction. In *CRYPTO'82*, pages 321–322. Plenum Press, New York, USA, 1982.

MTI86. T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.

Ope16. OpenSSL. https://www.openssl.org/, April 2, 2016.

Pal16. Pale Moon. http://www.palemoon.org, April 2, 2016.

PV05. P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *ASIACRYPT 2005*, *LNCS* 3788, pages 1–20. Springer, Heidelberg, December 2005.

PW17. D. Pointcheval and G. Wang. Vtbpeke: Verifier-based two-basis password exponential key exchange. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 301–312, New York, NY, USA, 2017. ACM.

Sch90. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, *LNCS* 435, pages 239–252. Springer, Heidelberg, August 1990.

Sho20. V. Shoup. Security analysis of spake2+. Cryptology ePrint Archive, Report 2020/313, 2020. https://eprint.iacr.org/2020/313.

Thr16. Thread Protocol. http://threadgroup.org/, April 2, 2016.

TWMP07. D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the secure remote password (srp) protocol for tls authentication. RFC 5054, RFC Editor, November 2007.

Wu98. T. D. Wu. The secure remote password protocol. In *NDSS'98*. The Internet Society, March 1998.

A Ommitted details for UC proof

We give here a more detailed description of each step in the proof, except for the last one, which is straightforward. In all steps below, note that all reductions that we build to justify the game hops have full control of the experiments and therefore know all the passwords that the environment inputs to the ideal functionality as inputs to uncorrupted parties.

HOP 1: NIZK SECURITY. The real-world is modified so that the CRS is generated as per the simulator code, which means that now the game has access to simulation and extraction trapdoors. Uncorrupted parties now use the NIZK simulator for all the generated proofs, so that they do not need to explicitly use the corresponding discrete logarithm as witness. Furthermore, in addition to verifying all proofs, the uncorrupted parties use the extraction trapdoor to recover the discrete logarithm of any group element that was computed by the adversary wrt to the basis prescribed by the protocol. (Note that this means first checking that the game does not already know the discrete log by keeping track of the proofs that were provided to the adversary, which is exactly what the simulator is doing.) If extraction fails or gives an incorrect result, the game aborts. We omit the details of the reduction to NIZK security, which follows the same steps as in [ABM15]. Note that both in the ideal world and in this modified real-game, the experiments explicitly compute (or they generate) the discrete logarithms of *all* protocol messages (if they do not abort). Moreover, these discrete logarithms uniquely define passwords $\mathsf{pw}_\beta = x_\beta/x_4$ and $\mathsf{pw}_\alpha = x_\alpha/x_2$, which the simulator extracts to check if the environment can actually compute the correct key with high probability. Intuitively, we will show that this is only the case if the environment created α (resp. β) itself, according to the rules of the protocol, and using the same password that it used to initialize the session to which α (resp. β) is delivered. In all other cases the protocol key is indistinguishable from random to the adversary.

HOP 2: REMOVE AMBIGUITY. We argue that the modifications introduced in this game and described above create a bad event with negligible probability. The modification that excludes collisions on the honestly generated X_i values introduces a statistical distance to the original games, which can be easily bounded using a birthday bound. The remaining modifications are treated using reductions to the DL problem; in all of these cases this implies guessing the session at which the event will occur to embed the DL problem instance.

Let us analyze the second modification. Consider any client-side and any server-side session as above agreeing on $(X_1X_2X_3, X_4) = (X'_1X'_2X'_3, X'_4)$. Clearly, if $X_3 \neq X'_3$ then the sessions are neither matched nor swapped, so we do not need to consider this case and we proceed assuming $X_3 = X'_3$. Suppose there is a collision on $X_1X_2 = X'_1X'_2$ when the two sessions are neither b-matched nor b-swapped, which means that we know $(X'_1, X'_2) \neq (X_1, X_2)$ and $(X'_1, X'_2) \neq (X_2, X_1)$. This (sub)-bad event can be reduced to the DL problem by setting $(X_1 = X, X_2 = g^{x_2})$: any offending tuple allows recovering the discrete logarithm of X as $x'_1 + x'_2 - x_2$.

Now we can extend the analysis to the full bad event. Suppose $X_1X_2X_3 = X'_1X'_2X'_3$, but we know $X_1X_2 \neq X'_1X'_2$, so it must be the case that $X_1 \neq X'_1$. Again we set $X_1 = X$ and we recover the discrete logarithm as $x = x'_1 + x'_2 + x'_3 - x_2 - x_3$. Note that this argument follows exactly as in [ABM15] and that the exact same reasoning applies to the case $(X_1X_3X_4, X_2) = (X'_1X'_3X'_4, X'_2)$.

Interestingly, the final bad event we consider in this hop, which deals with accepting an α or β for b-swapped sessions that leads to different derived key is not considered in [ABM15]. This is because all such sessions can be treated there as passive attacks, since it is guaranteed that client and server use the same password. In our case we have the possibility that passwords differ on both sides, and hence must treat some of these sessions as in the case of active attacks, which means we will need to compute the derived key. It is therefore important that, when b-swapping occurs, either the derived key is not changed, in which case we can treat the sessions as passively attacked sessions, or that sessions cannot become fully swapped (i.e., the original α or β cannot be accepted), in which case we can treat them as actively attacked sessions. The proof of this case is simple. Suppose α is accepted that comes from a session s.t. $(X_1, X_2) = (X'_2, X'_1)$. Then, the proof justifying α was verified using a basis $X_2X_3X_4$, which differs from $X_1X_3X_4$ (or else the session would have aborted due to a collision). This implies a discrete log x_α was extracted such that $\alpha = (X_1X_3X_4)^{x_2\mathsf{pw}} = (X_2X_3X_4)^{x_\alpha}$. This discrete logarithm permits computing the discrete logarithm of x_1 as $x_\alpha(x_2 + x_3 + x_4)/x_2\mathsf{pw} - (x_3 + x_4)$ and hence the bad event can be excluded with a reduction to the discrete logarithm problem similarly to the reductions above. The case of β is identical.

HOP 3: ACTIVELY ATTACKED SESSIONS. We modify the real world so that all sessions that are neither matched nor swapped explicitly extract a password as per the simulator code and check if the password is correct. If not, these sessions use a uniform random element as key to the PRF. Furthermore, if they are

matched or swapped, but use different passwords, then we also randomize the key to the PRF (but perform no extraction). We will use a hybrid argument that first modifies all such sessions on the client side, and then all of those on the server side. We show that any environment that can detect such a change can be used to break Decision Squared Diffie-Hellman. This problem, which as discussed in [ABM15] is a stronger assumption than DDH, requires an adversary to distinguish tuples of the form (g^x, W) where W is either $W = g^{x^2}$ or an independently sampled random element. We present the client-side argument first and then only state the differences for the server side.

The reduction programs $X_2 = g^x$ in the i-th client session. This means implicitly that $x_2 = x$, and this is the only discrete logarithm unknown to the reduction for all X_i messages occurring in the game. This means, in particular, that α can be computed as $(g^x)^{\mathsf{pw}(x_1+x_3+x_4)}$. Moreover, the output key of this session in the case of a correct password guess in an active attack (which the reduction can check as per the simulator code) will have the form $g^{(x_1+x_3)x_2x_4\mathsf{pw}}$. The reduction can also compute this value, as it knows all discrete logarithms except for x_2. The same applies to matching or swapped sessions with the same password.

On the other hand, if the password guess is incorrect, or if we have matched or swapped sessions with different passwords, the key generated by this session will be of the form $(\beta X_4^{-x_2\mathsf{pw}})^{x_2} = g^{x_2(x_\beta(x_1+x_2+x_3)-x_2x_4\mathsf{pw})}$, where we know $x_\beta \neq \mathsf{pw}x_4$. Expressing the key as a function of W we get $W^{x_\beta - x_4\mathsf{pw}}g^{x_\beta x_2(x_1+x_3)}$ and it is clear the key is completely random if W is random.

For all the other client-side sessions no problems can arise in the reduction, as it knows all the necessary discrete logarithms. Similarly, the sessions on the server side have not yet been modified and the correct keys can be computed using the code for uncorrupted parties.

We consider now the hybrid argument for server-side sessions. The strategy is exactly the same: the SqDH problem instance is programmed into X_4, and all the details of the reduction are the same with appropriate renaming, except when it comes to the fact that we need to simulate client-side sessions which have already been modified by the previous hybrid. The only relevant case is the case of actively attacked client sessions where the reduction must perform a correct password guess, but it does not know x_β because β was programmed by the reduction. However, in this case we know that the session is not matched nor swapped (because extraction is taking place). We also know that the basis for β is checked as $X_1X_2X_3$, which must be equal to $X_1'X_2'X_3'$ (as otherwise the x_β would have been extracted). By our second hop, this implies the discrete log of X_4 must be known, and hence we can check the password guess as $\beta = (X_1X_2X_3)^{x_4\mathsf{pw}}$ and compute a correct key, if needed.

Hop 4: Remaining passively attacked and swapped sessions. We modify the real world so that all passively attacked or swapped sessions where the same password was used on both sides use a uniform random element as key to the PRF (the same key is derived on both sides). We will use a hybrid argument over all pairs (i, j) where i refers to the i-th client-side session and j indexes server side sessions. If anyone of these two sessions is passively attacked or swapped, we replace the PRF key with a random group element. We show that any environment that can detect such a change can be used to break Decision triple group Diffie-Hellman. This problem, which as discussed in [ABM15] is hard if DDH is hard, requires an adversary to distinguish tuples of the form $(g^x, g^y, g^z, g^{xy}, g^{xz}, g^{yz}, W)$ where W is either $W = g^{xyz}$ or an independent random element.

The reduction programs $X_2 = g^x$ on the client-side session and $(X_3, X_4) = (g^y, g^z)$ on the server side. This means implicitly that $x_2 = x$, $x_3 = y$ and $x_4 = z$, and therefore that α can be computed as $((g^x)_1^x g^{xy} g^{xz})^{\mathsf{pw}}$, whenever the sessions are matching with the same passwords. Similarly, β on the server side can be computed as $((g^z)_1^x g^{xz} g^{yz})^{\mathsf{pw}}$, whenever the sessions are matching with the same password. Moreover, the derived keys on either side can be computed as $((g^x z)^{x_1} g^{xyz})^{\mathsf{pw}}$, and they become totally random when g^{xyz} is replaced with a random element. Similar reasoning applies for when the sessions are swapped (since we know that derived keys must be consistent with matching sessions).

Recall that all actively attacked sessions with wrong password guesses have been modified to use random group elements as PRF keys. The same happens for matching or swapped sessions using different passwords. However, some actively attacked sessions still give out correct keys if the passwords are correct. And it could happen that the sessions do not match each other and they match some other sessions in the game. In all of these cases we need to verify that the reduction can *back-track* and correctly simulate these sessions. First

of all note that in all cases we can compute α and β correctly as described above. Moreover, if the sessions are matched or swapped with any other sessions in the game, then the reduction knows enough discrete logarithms (in those other sessions) to compute the output keys correctly.

It remains to consider the cases where these sessions turn out to be actively attacked. We need to consider various cases (we give only the client side, as the server side works similarly). The problem, of course, is that we do not know x_2 and we might not be able to extract x_3, x_4 and x_β, but we do know the i-th client session is *not* matching or swapping with the j-th server session, which means that at least one of X'_3, X'_4, or β were *not* programmed by the reduction and therefore the discrete logarithm is known.

- Suppose x_β is known. Then the correct password check can be done in the exponents as $(X'_4)^{\mathsf{pw}} = g^{x_\beta}$. Note also that, if the check goes through, we recover x_4, which means that the correct key can be computed as $((g^x)^{x_1} g^{xy})^{x_4 \mathsf{pw}}$.
- Suppose x_β is not known, which means β was programmed by the reduction. We know the session is not paired nor swapped, but we know that the basis for β generation must have been the same, as otherwise x_β would have been extracted. By our second hop this implies the discrete log of X_4 must be known, and hence we can check the password guess as $\beta = (X_1 X_2 X_3)^{x_4 \mathsf{pw}}$ and compute a correct key, if needed, just as in the previous case.

Let us see what happens in all the other sessions, after indices (i, j) to make sure the reduction can replicate the game. It must correctly check for password guesses in actively attacked sessions and compute the correct keys if needed, as well as compute the correct keys for yet untouched passively attacked and swapped sessions. This is done as follows:

- For yet untouched passively attacked or passive sessions, the reduction knows the discrete logarithms of the messages X_i generated in that session, and it also knows the password, so it can compute the correct key. Note that in this case there is no need to perform password checks.
- For all remaining actively attacked sessions, observe that the reduction will know more information than in the case of session i, and so the same argument applies.

Hop 5: Randomizing α and β. The details for this hop follow exactly as in [ABM15]. We give a brief overview. The idea is to transform α and β into random values in all sessions and to compute the output keys for the corresponding sessions, when correct password guesses occur, as the adversary would (this is what the simulator does). For this it is crucial to observe that we have randomized the input to the PRF in all sessions, except those in which the attacker places a correct password guess. This means that, only in these cases will the simulation need to compute the correct output key (which the simulator then passes on to the ideal functionality in order to match the real world). The proof again uses a hybrid argument, first over all client-side sessions, and then over all server-side sessions. We give only the first case here. The idea is to get a DDH challenge (X, Y, Z) and program the i-th client session to set $X_1 = X$ and $X_2 = Y$ and to compute $\alpha = (Z Y_{x_3 + x_4})^{\mathsf{pw}}$, which gives the correct value of α if $Z = \mathsf{CDH}(X, Y)$ and a random group element if Z is uniform in the group. If a correct password guess occurs, the output key can be computed as $(Z X_2^{x_3}) x_4 \mathsf{pw}$. Note that if Z is the CDH of X and Y, the computed key is consistent with the computation based on β, as required. It is clear that the reduction can simulate all other client-side sessions and all server-side sessions that are matching or swapped, since it only needs to compute random keys. The only problem could happen when checking a correct password guess on the server side while not knowing the discrete logarithm for α, but this can never occur due to our second hop.

Extension to Static Corruptions. The extension to static corruptions is straightforward (recall that we are using the stronger version of the ideal PAKE functionality from [AHH21]). The simulator is extended as follows: whenever a message is delivered from a corrupt party, the simulator just treats it as if it is part of an active attack. In particular, it runs the extractor on it to recover the discrete logarithm (unless the message is replayed, in which case it already knows the discrete logarithm). If extraction fails, then the simulator aborts. When α or β is delivered from a corrupt party, the simulator extracts a password guess such as in the case of active attacks, and it asks the functionality whether the guess is correct. If the guess is

correct, then it programs the output of the functionality for the honest party using the correctly computed key. Otherwise, the functionality will use a completely random key (as the password test failed). The proof of simulator correctness extends in the following way: whenever dealing with a session that is interacting with a corrupted party, treat it exactly as the sessions that are under active attack. With all the game hops modified in this way, we can gradually transform the real-world with static corruptions into the ideal world with our extended simulator.

B Reductions

We now prove auxiliary lemmas supporting the proof of Theorem D.2.

Lemma B.1. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ such that*

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathrm{uzk}}_{\mathcal{B}_{\mathsf{NIZK}}}()$$

Proof. We consider $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ attacker in Fig. 13 and we prove that the attacker distinguishes real from random proofs with the advantage $\mathrm{Adv}^{\mathrm{uzk}}_{\mathcal{B}_{\mathsf{NIZK}}}$. We simulate proofs in the following queries. For SendInit-C1(C, i, S), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_1 = g^{x_1}$ and $X_2 = g^{x_2}$ and calls simulation oracle $\mathsf{Sim_O}$ that takes X_1 and C and outputs π_1, then $\mathsf{Sim_O}$ takes X_2 and C and outputs π_2. For SendInit-S1(S, i, C), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_3 = g^{x_3}$ and $X_4 = g^{x_4}$ and calls $\mathsf{Sim_O}$ that takes X_3 and S and outputs π_3, then $\mathsf{Sim_O}$ takes X_4 and S and outputs π_4. For Execute(C, S, i, j), $\mathcal{B}^{\mathsf{uzk}}_{\mathsf{NIZK}}$ generates $X_1 = g^{x_1}$, $X_2 = g^{x_2}$, X_3 and π_4 for X_4 and calls $\mathsf{Sim_O}$ to simulate π_1, π_2, π_3, π_4 for corresponding X_1, X_2, X_3, and X_4. The attacker interpolates between Game 0 and Game 1, meaning if it knows it is playing Game 0 it outputs 1 and if it knows it is playing Game 1, it outputs 0. □

Lemma B.2. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ such that*

$$\Pr[G_{1.5} \Rightarrow \mathsf{bad}_{1.5}] \leq 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}()$$

Proof. We consider $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ attacker in Fig. 15. We set a bad flag $\mathsf{bad}_{1.5}$ whenever the extractor fails and denote the advantage of the attacker as $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}$. We check if the extractor returns the correct witness in the following queries. In Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$, we check if received value X'_3 comes from the adversary (not in List) in which case we call $\mathsf{NIZK.Extract}(\mathsf{crt}, \mathsf{td}_e, X'_3, \pi'_3, S)$ which outputs a witness x'_3. Now, we check if $X'_3 = g^{x'_3}$, in which case the witness is valid. Otherwise, either the extraction fails and $\mathsf{bad}_{1.5}$ occurred or the witness is not valid and in either case we add the pair (X'_3, π'_3) to $\mathsf{List_{Ext}}$. We do the same for X'_4 and in Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$ for X'_1 and X'_2. At the end of the game, $\mathcal{B}^{\mathsf{ext}}_{\mathsf{NIZK}}$ guesses one pair (X', π') from $\mathsf{List_{Ext}}$ for which $\mathsf{bad}_{1.5}$ might have occurred. Since, the extractor fails with the advantage $\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}$ and we guess one pair from $\mathsf{List_{Ext}}$, we can bound the probability of $\mathsf{bad}_{1.5}$ happening with $2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathcal{B}_{\mathsf{NIZK}}}$. □

Lemma B.3. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_4$ such that*

$$\Pr[G_4 \Rightarrow \mathsf{bad}_4] \leq n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Proof. We consider a CTGDH attacker $\mathcal{B}_4$ in Fig 19. It gets a challenge of the form $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$ and finds $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$. The attacker uses the challenge for the lth $\mathsf{Execute}$ query to compute $X_1 = g^{x_1}$, $X_2 = X$, $X_3 = Y$, $X_4 = Z$, $\alpha = (D_{XY}D_{XZ}X^{x_1})^{\mathsf{pw}_{cs}}$ and $\beta = (D_{XZ}D_{YZ}Z^{x_1})^{\mathsf{pw}_{cs}}$. If there was a query with a collision in T, we add it to the list T_4. Then, when the game finishes, the reduction guesses one query from T_4 for which bad_4 might have occurred. If the guess was successful, it means that bad_4 occurred and $\mathcal{A}$ solved $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$ so $\mathcal{B}_4$ can recover g^{xyz} by computing:

$$g^{xyz} = \frac{\mathsf{Key}^{\frac{1}{\mathsf{pw}_{cs}}}}{D_{XZ}^{x_1}}$$

For Send queries, the attacker runs everything as in Game 4. □

Lemma B.4. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{4.5.1}$ such that*

$$\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

Proof. We consider a DDH attacker $\mathcal{B}_{4.5.1}$ in Fig 21. It gets a generalized challenge of the form $(X_1, \ldots, X_{n_{se}}, Y_1, \ldots, Y_{n_{se}}, Z_1, \ldots, Z_{n_{se}})$ and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge for each Execute query to compute $X_1 = X_k$, $X_2 = Y_k$, $X_3 = g^{x_3}$, $X_4 = g^{x_4}$, $\alpha = (Z_k X_2^{x_3+x_4})^{\mathsf{pw}_{cs}}$ and $\beta = X_2^{x_4 \mathsf{pw}_{cs}} g^{(x_1+x_2)x_4 \mathsf{pw}_{cs}}$. The attacker interpolates between Game 4 and Game 4.5.1, meaning if (X_k, Y_k, Z_k) is a real DDH tuple, it is playing Game 4 and otherwise it is playing Game 4.5.1. □

Lemma B.5. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{4.5.2}$ such that*

$$\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Proof. We consider a DDH attacker $\mathcal{B}_{4.5.2}$ in Fig 23. It gets a generalized challenge of the form $(X_1, \ldots, X_{n_{se}}, Y_1, \ldots, Y_{n_{se}}, Z_1, \ldots, Z_{n_{se}})$ and submits $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. The attacker uses the challenge for each Execute query to compute $X_1 = g^{x_1}$, $X_2 = g^{x_2}$, $X_3 = X_k$, $X_4 = Z_k$, $\alpha \xleftarrow{\$} \mathcal{G}$ and $\beta = (Z_k X_4^{x_1+x_2})^{\mathsf{pw}_{cs}}$. The attacker interpolates between Game 4.5.1 and Game 4.5.2, meaning if (X_k, Y_k, Z_k) is a real DDH tuple, it is playing Game 4.5.1 and otherwise it is playing Game 4.5.2. □

Lemma B.6. *For every attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_6$ such that*

$$\Pr[G_6 \Rightarrow \mathsf{bad}_6] \leq n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

Proof. We consider a CSqDH attacker $\mathcal{B}_6$ in Fig 26. The reduction $\mathcal{B}_6$ gets a challenge X and finds the solution to CSqDH. The attacker guesses the lth fresh session, where it uses the challenge for SendInit-C1(C, i, S) query to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$ it computes $\alpha = X_2^{(x_1+x'_3+x'_4)\mathsf{pw}_{cs}}$. We add all queries where bad_6 might have occurred to T_{bad}. In the end of the game, the reduction guesses one query from T_{bad}. If the guess was successful it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_6$ can recover g^{x^2} in the following way:

$$\mathsf{Key}_1 = (\frac{\beta'}{X^{x'_4 \mathsf{pw}_{cs}}})^x = (\frac{X^{x_1+x'_3+x'_4}}{X^{x'_4 \mathsf{pw}_{cs}}})^x = X^{(x_1+x'_3)x'_4 \mathsf{pw}_1} g^{x^2 x'_4(\mathsf{pw}_1 - \mathsf{pw}_{cs})} \tag{1}$$

$$\mathsf{Key}_2 = (\frac{\beta'}{X^{x'_4 \mathsf{pw}_{cs}}})^x = (\frac{X^{x_1+x'_3+x'_4}}{X^{x'_4 \mathsf{pw}_{cs}}})^{x_2} = X^{(x_1+x'_3)x'_4 \mathsf{pw}_2} g^{x^2 x'_4(\mathsf{pw}_2 - \mathsf{pw}_{cs})} \tag{2}$$

where x is unknown. From (1) and (2) follows:

$$g^{x^2} = (\frac{\mathsf{Key}_1}{\mathsf{Key}_2 X^{(x_1+x'_3)x'_4(\mathsf{pw}_1 - \mathsf{pw}_2)}})^{\frac{1}{x'_4(\mathsf{pw}_1 - \mathsf{pw}_2)}}$$

We do the same procedure on the server side and we plug-in the challenge for SendInit-S1(S, i, C). Note, in case of Corrupt query in lth session, the session is no longer fresh and bad cannot occur and it means that the attacker made the wrong guess so we abort. □

Lemma B.7. *For every algebraic attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_{7-8.1}$ such that*

$$\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}] \leq 2 n_{se} \mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

Proof. We consider a DSqDH attacker $\mathcal{B}_{7-8.1}$ in Fig 29. It gets a challenge (X, Y) and submits $b = 1$ if (X, Y) is a real DSqDH tuple or $b = 0$ otherwise. The attacker uses a hybrid argument and for fresh mth session, it plugs-in the challenge for SendInit-C1(C, i, S) to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$ it computes $\alpha = X_2^{(x_1+x'_3+x'_4)\mathsf{pw}_{cs}}$. In Send-C3$(C, i, S, \beta', \mathsf{alg}')$ where $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$, in case of $\mathsf{Corrupt}$, we rewrite β' as $\beta' = g^a X_1^b X_2^c \alpha^d$. Then we check:

(a) If $X_2^c \alpha^d \neq X_2^{x'_4 \mathsf{pw}_{cs}}$ the key is computed as
$\mathsf{Key} = X_2^a (\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{X_2^{(x'_3+x'_4)}})^b Y^{(c+d(x_1+x'_3+x'_4)\mathsf{pw}_{cs} - x'_4 \mathsf{pw}_{cs})}$. This means if $Y = g^{x^2}$ then the key is real and random otherwise.

(b) If $X_2^c \alpha^d = X_2^{x'_4 \mathsf{pw}_{cs}}$, we embed α in the key and we compute it as $\mathsf{Key} = X_2^{(a+x_1 b)} = X_2^a (\frac{\alpha^{\frac{1}{\mathsf{pw}}}}{X_2^{(x'_3+x'_4)}})^b$.

$\mathcal{A}$ interpolates between the two games, meaning, if $Y = g^{x^2}$ then it is playing Game 7, otherwise it is playing Game 7-8.m.1.
Note, we do the same analysis on the server's side, for SendInit-S1(S, i, C), Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$ and Send-S3$(S, i, C, \alpha', \mathsf{alg}')$, to bound problematic cases of α' (coming from $\mathcal{A}$) . □

Lemma B.8. *For every algebraic attacker $\mathcal{A}$, there exists an attacker $\mathcal{B}_9$ such that*

$$\Pr[G_9 \Rightarrow \mathsf{bad}_9^2] \leq n_{se} n_{ro} \mathrm{Adv}^{\mathsf{CSqDH}}_{\mathcal{B}_9}()$$

Proof. We consider a CSqDH attacker $\mathcal{B}_9$ in Fig 32. The reduction $\mathcal{B}_9$ gets a challenge X and finds the solution to CSqDH. The attacker guesses the lth fresh session, where instances accept, and there was a $\mathsf{Corrupt}$ query and a query $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$ where there was β' with $\mathsf{pw} \neq \mathsf{pw}_{cs}$. $\mathcal{B}_9$ plugs-in the challenge for SendInit-C1(C, i, S) query to compute $X_1 = g^{x_1}$, $X_2 = X$, and for Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$ it computes $\alpha = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. We add all queries where bad_9^1 might have occurred to T_9. In the end of the game, the reduction guesses one query from T_9. If the guess was successful it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_9$ can recover g^{x^2}:

$$g^{x^2} = (\frac{\mathsf{Key}}{X^{(a+x_1 b+wd)}})^{\frac{1}{c - x_4 \mathsf{pw}_{cs}}}$$

We do the same procedure on the server side and we plug-in the challenge for SendInit-S1(S, i, C). □

C J-PAKE

The J-PAKE description. The protocol is symmetric and consists of two rounds. In the first round a Client chooses two random values x_1 and x_2 from $\mathbb{Z}_p$ and computes $X_1 = g^{x_1}$ and $X_2 = g^{x_2}$. Then it generates proof of knowledge for π_1 for X_1 and π_2 for X_2. A Server does the same, only with x_3 and x_4, computing $X_3 = g^{x_3}$ and $X_4 = g^{x_4}$ and generating π_3 and π_4 for X_3 and X_4 respectively. Both sides exchange a tuple of *Identity, Values, Corresponding proofs*, without any order who goes first. Then, both sides verify the proofs π_1, π_2, π_3, π_4 and abort if verification fails.

In the second round, the Client computes $\alpha = g^{(x_1+x_3+x_4)x_2 \mathsf{pw}}$, along with corresponding proof π_α for exponent $x_2 \mathsf{pw}$, and sends α and π_α to the Server. The Server does the same: computes $\beta = g^{(x_1+x_2+x_3)x_4 \mathsf{pw}}$, generates corresponding proof π_β for exponent $x_4 \mathsf{pw}$ and sends β and π_β to the Client. Both sides verify the proofs π_α, π_β and abort if verification fails.

In the last step, the Client computes $\mathsf{Key} = (\beta X_4^{-x_2 \mathsf{pw}})^{x_2}$ and the Server computes $\mathsf{Key} = (\alpha X_2^{-x_4 \mathsf{pw}})^{x_4}$, which results in both sides holding the same key $\mathsf{Key} = g^{(x_1+x_3)x_2 x_4 \mathsf{pw}}$. The session key follows as $K = H_1(\mathsf{Key})$, where H_1 is a hash function mapping into $\{0,1\}^\kappa$, with κ being the security parameter.

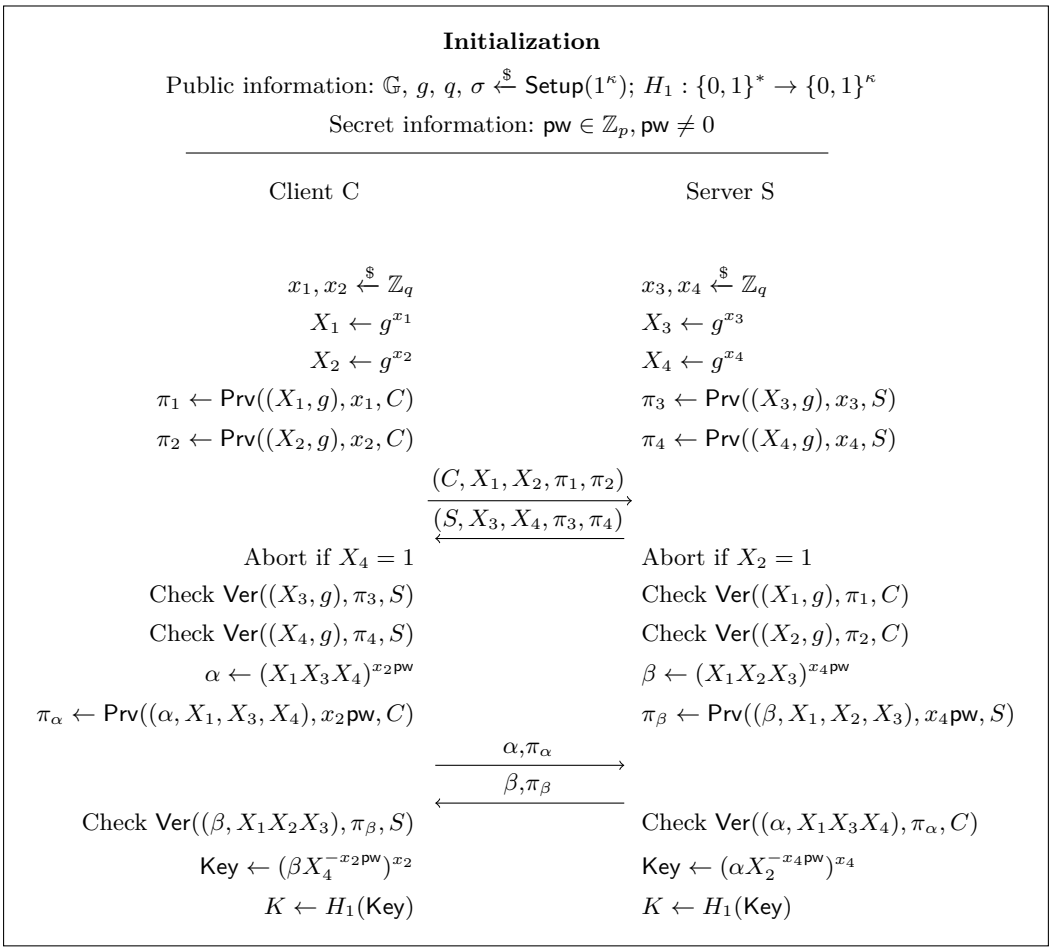

Fig. 6: The J-PAKE protocol

D Variations of s-JPAKE

In this section, we describe the sRO-J-PAKE (Fig. 7) and sCRS-J-PAKE (Fig. 9) protocols in full detail. Moreover, we provide the security proof of both protocols in the same model as sJ-PAKE (Sec. 5). We analyze both proofs in depth by going through games of sJ-PAKE and only point out the differences with sJ-PAKE. For simplicity, we leave the names and the number of games, bad events and reductions the same as the one in sJ-PAKE (Sec. 6 and Sec. B). In addition, all changes in games and reductions of sRO-J-PAKE and sCRS-J-PAKE imply the changes in their code-based proof. Unfortunately, due to space constraints, we do not provide code-based proof for sRO-J-PAKE and sCRS-J-PAKE. Most of the changes in the proof for sRO-J-PAKE and sCRS-J-PAKE occur in the reductions where we plug-in challenges differently compared to sJ-PAKE due to missing values X_1 and X_3. We modify the equations we use to compute the solution to a hard problem accordingly. Lastly, we use the same assumptions as in sJ-PAKE, defined in Sec. 3.2.

For a given security parameter k, let $\mathbb{G}$ be a finite multiplicative group[7] of prime order q, such that $|q| := k$. Being the strongest assumption necessary, we will assume the Computational Square Diffie Hellman (CSqDH, see Sec. 3.2) holds over $\mathbb{G}$.

D.1 sRO-J-PAKE

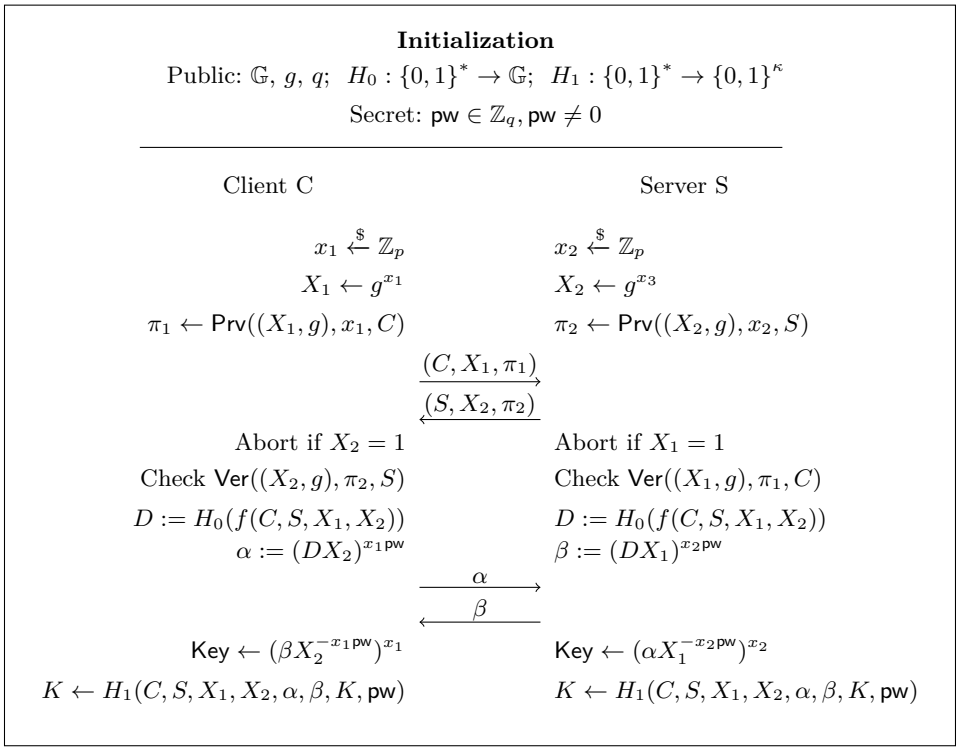

Fig. 7: The sRO-J-PAKE protocol.

[7] The group of interest is either a SFF or EC group. Throughout this paper, protocols will be presented multiplicatively.

Description. Let H_0 be a full-domain hash mapping $\{0,1\}^*$ to $\mathbb{G}$. H_1 is a hash function from $\{0,1\}^*$ to $\{0,1\}^\kappa$. A function f is used to ensure that both parties sort values identically. Basically, D plays the role of (X_1X_3). In the first round a client C generates X_1 and the proof of knowledge, π_1 for discrete log x_1 and transmits the identity, X_1 and π_1 to a Server S, while S does the same computation only with X_2 and π_2 and sends the identity, X_2 and π_2 to C. When both parties receive the message from the first round, they compute common value $D = H_0(f(C, S, X_1, X_2))$. Then C generates $\alpha = (DX_2)^{x_1\mathsf{pw}}$ and sends it to S, and at the same time S generates $\beta = (DX_1)^{x_2\mathsf{pw}}$ and sends it to C. After receiving α and β, both parties compute the key by the protocol $\mathsf{Key} = (\beta X_2^{-x_1\mathsf{pw}})^{x_1}$ (resp. $\mathsf{Key} = (\alpha X_1^{-x_2\mathsf{pw}})^{x_2}$). Both parties should hold the same session key $K = H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$.

D.2 Security game-based proof of sRO-J-PAKE

Before the analysis of the game-based proof, we stress that adding full domain hash H_0 to the protocol means that our simulator will have to administer two random oracles: one oracle that responds to $H_1(C, S, X_1, X_2, \alpha, \beta, \mathsf{Key}, \mathsf{pw})$ queries[8] and another oracle that responds to $D = H_0(f(C, S, X_1, X_2))$ queries. Both oracles have to make sure that responses are consistent. We define RO to respond in case $\mathcal{A}$ asks $H_0(f(C, S, X_1, X_2))$ in Fig. 8.

For each fresh RO query $H_0(f(C, S, X_1, X_2))$ the simulator computes $D := g^d$, where $d \xleftarrow{\$} \mathbb{Z}_p$ and returns D to $\mathcal{A}$. The simulator administrates all the *query - response* by adding them to the list. If $\mathcal{A}$ already asked for H_0, the simulator simply retrieves the response D from the list and gives it to $\mathcal{A}$. The number of random oracle queries to H_0 we denote as n_{h0}.

Fig. 8: Simulation of H_0

As already mentioned, function D in the sRO-J-PAKE protocol substitutes X_1X_3 term. Thus, the only way the security proof will work is that the simulator simulates H_0 as $D = g^d$, where it knows d. Since d is chosen randomly to define D in Fig. 8 and x_1 (resp. x_2) can be extracted from the ZK-PoK, the reduction will have all the information it needs to simulate the responses to any query that $\mathcal{A}$ asks. That being said, we give a formal proof of sRO-J-PAKE (Fig. 7), and we only state the potential differences with respect to sJ-PAKE protocol. We also refer the reader to [LST16] where the proof of RO-J-PAKE is displayed in detail and we claim that sRO-J-PAKE has the same differences with sJ-PAKE as RO-J-PAKE has with J-PAKE. Furthermore, we stress that we modify all games by changing X_1, X_2, X_3 and X_4 to X_1, X_2 and π_1, π_2, π_3 and π_4 to π_1, π_2 and we add the second oracle that simulates random queries for H_0.

Theorem D.1. *Let sRO-J-PAKE be the protocol described in Fig. 7. Take an RoR attacker $\mathcal{A}$ against sRO-J-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te}, n_{ro}, n_{ho} queries to* Send, Execute, Reveal, Corrupt, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional*

[8] This oracle is already defined and administrated in sJ-PAKE proof in Sec. 6.

Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that

$$\begin{aligned}\mathrm{Adv}^{\mathtt{sRO-J-PAKE}}_{\mathcal{A}}() \leq\ & \frac{2n_{se}}{|\ D\ |} + \frac{(n_{se}+2n_{ex})^2}{q} \\ & + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK, formally defined in 3.6.*

Proof. **Game 0: Original protocol.**

$$\mathrm{Adv}^{\mathtt{sRO-J-PAKE}}_{\mathcal{A}}() = |\ \Pr[G_0 \Rightarrow \mathsf{T}] - \frac{1}{2}\ |$$

Game 1: Simulate ZK-PoK proofs. This game is the same as in Game 1 of sJ-PAKE, except we simulate only proofs of knowledge of X_1 and X_2. Everything else stays the same. We have

$$\Pr[G_0 \Rightarrow \mathsf{T}] - \Pr[G_1 \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}()$$

Game 1.5: Extract discrete logs from adversarial proofs. This game is the same as in Game 1.5 of sJ-PAKE, except we extract only from adversarial proofs of knowledge π_1 and π_2. Everything else stays the same. We have

$$|\ \Pr[G_1 \Rightarrow \mathsf{T}] - \Pr[G_{1.5} \Rightarrow \mathsf{T}]| \leq 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}()$$

Game 2: Force unique values
This game is the same as in Game 2 of sJ-PAKE except we force uniqueness of X_1 and X_2. We bound this change

$$\Pr[G_{1.5} \Rightarrow \mathsf{T}] - \Pr[G_2 \Rightarrow \mathsf{T}] \leq \frac{(n_{se}+2n_{ex})^2}{q}$$

Game 3: Adding freshness. Description of this game is the same as in Game 3 of sJ-PAKE. There is no change in the bound.

$$\Pr[G_3 \Rightarrow \mathsf{T}] = \Pr[G_2 \Rightarrow \mathsf{T}]$$

Game 4: Randomize session keys for matching sessions.
Description of this game is the same as in Game 4 of sJ-PAKE, except for the natural change of X_1, X_2, X_3 and X_4 to X_1, X_2. This change implies a modification in the reduction to CTGDH problem. We state the differences with the reduction described in Lemma B.3:
Given the challenge $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$, we build an attacker $\mathcal{B}_4$ that finds $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$. We plug-in the challenge in $X_1 = X$ and $X_2 = Y$ while the values Z is embedded as the output of $H_0(C, S, X_1, X_2)$ and we set $\alpha = (D_{XY}D_{XZ})^{\mathsf{pw}_{cs}}$ and $\beta = (D_{XZ}D_{YZ})^{\mathsf{pw}_{cs}}$. Then, when the game finishes, the attacker guesses one query for which bad_4 might have occurred. If the guess was successful, it means that bad_4 occurred and $\mathcal{A}$ solved $\mathsf{CTGDH}(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$ so $\mathcal{B}_4$ can recover g^{xyz} by computing:

$$g^{xyz} = \mathsf{Key}^{\frac{1}{\mathsf{pw}_{cs}}}$$

Therefore we have the same bound:

$$|\ \Pr[G_3 \Rightarrow \mathsf{T}] - \Pr[G_4 \Rightarrow \mathsf{T}]| \leq n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}()$$

Game 4.5: Randomize alpha/beta for Execute queries.
Description of this game is the same as in Game 4.5 of sJ-PAKE. The only change we do is in the reduction to DDH problem when randomizing both α and β in Games 4.5.1. and 4.5.2. We state the differences with the reductions described in Lemma B.4 and Lemma B.5:
Given the generalized challenge $(X_1, ..., X_{n_{se}}, Y_1, ..., Y_{n_{se}}, Z_1, ..., Z_{n_{se}})$, we build attackers $\mathcal{B}_{4.5.1}$ and $\mathcal{B}_{4.5.2}$ that submit $b = 1$ if (X_k, Y_k, Z_k) is a real DDH tuple and $b = 0$ if it is a random DDH tuple, for some $1 \leq k \leq n_{ex}$. The problem is tightly equivalent to DDH by self-reducibility. We plug-in the challenge in $X_1 = X_k$ and $X_2 = Y_k$ and we set $\alpha = (X_k^d Z_k)^{\mathsf{pw}_{cs}}$ (resp. $\beta = (Y_k^d Z_k)^{\mathsf{pw}_{cs}}$) where D is the output of $H_0(C, S, X_1, X_2)$ computed as $D = g^d$. The bounds for Game 4.5.1 and Game 4.5.1 do not change with respect to the same games in s-JPAKE:

$$\Pr[G_4 \Rightarrow \mathsf{T}] - \Pr[G_{4.5.1} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}()$$

$$\Pr[G_{4.5.1} \Rightarrow \mathsf{T}] - \Pr[G_{4.5.2} \Rightarrow \mathsf{T}] \leq \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}()$$

Game 5: Randomize session keys for Send queries. Description of this game is the same as in Game 5 of sJ-PAKE so the bound does not change:

$$|\Pr[G_{4.5.2} \Rightarrow \mathsf{T}] - \Pr[G_5 \Rightarrow \mathsf{T}]| \leq \Pr[G_5 \Rightarrow \mathsf{bad}_5]$$

Game 6: Detect duplicates.
Description of this game is the same as in Game 6 of sJ-PAKE, except for modification in the reduction to CSqDH problem. We state the differences with the reduction described in Lemma B.6:
Given the challenge X, we build an attacker $\mathcal{B}_6$ that finds solution to CSqDH problem. Let us say that $\mathcal{B}_6$ plugs-in the challenge in the lth session on the clients side, in $X_1 = X$ and it sets $\alpha = X^{(X_2'+d)\mathsf{pw}_{cs}}$, where x_2' is the witness extracted from adversarial proof π_2' and d is the discrete log of D which represents the output of $H_0(C, S, X_1, X_2)$. Then, when the game finishes, $\mathcal{B}_6$ guesses one query for which bad_6 might have occurred. If the guess was successful it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_6$ can recover g^{x^2} in the following way:

$$\mathsf{Key}_1 = (\frac{\beta'}{X^{x_2'\mathsf{pw}_{cs}}})^x = (\frac{(XD)^{\mathsf{pw}_1}}{X^{x_2'\mathsf{pw}_{cs}}})^x = X^{dx_2'\mathsf{pw}_1} g^{x^2 x_2'(\mathsf{pw}_1 - \mathsf{pw}_{cs})} \tag{1}$$

$$\mathsf{Key}_2 = (\frac{\beta'}{X^{x_2'\mathsf{pw}_{cs}}})^x = (\frac{(XD)^{\mathsf{pw}_2}}{X^{x_2'\mathsf{pw}_{cs}}})^x = X^{dx_2'\mathsf{pw}_2} g^{x^2 x_2'(\mathsf{pw}_2 - \mathsf{pw}_{cs})} \tag{2}$$

where x is unknown. From (1) and (2) follows:

$$g^{x^2} = (\frac{\mathsf{Key}_1}{\mathsf{Key}_2 X^{dx_2'(\mathsf{pw}_1 - \mathsf{pw}_2)}})^{\frac{1}{x_2'(\mathsf{pw}_1 - \mathsf{pw}_2)}}$$

We do the same procedure on the server side and we plug-in the challenge in $X_2 = X$ and set $\beta = X^{(X_2'+d)\mathsf{pw}_{cs}}$. The bound and other remarks do not change:

$$\Pr[G_5 \Rightarrow \mathsf{bad}_5] \leq \Pr[G_6 \Rightarrow \mathsf{bad}_5] + n_{se} n_{ro}^2 \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}()$$

Game 7: Add algebraic representation. Description of this game is the same as in Game 7 of sJ-PAKE. Thus, the bound does not change:

$$\Pr[G_6 \Rightarrow \mathsf{bad}_5] = \Pr[G_7 \Rightarrow \mathsf{bad}_5]$$

Game 8: Randomizing α and β for Send queries.
Description of this game is the same as in Game 8 of sJ-PAKE. The only change we do is in the reduction to DSqDH problem when bounding bad cases of α and β in Game 7-8.1. We state the differences with the reduction described in Lemma B.7:
Given the challenge (X, Y)), we build an attacker $\mathcal{B}_{7-8.m.1}$ that submits $b = 1$ if (X, Y) is a real DSqDH tuple and $b = 0$ if it is a random DSqDH tuple. $\mathcal{B}_{7-8.m.1}$ uses a hybrid argument and for fresh mth session, it plugs-in the challenge for the client side $X_1 = X$ and sets $\alpha = X^{(x_2'+d)\mathsf{pw}_{cs}}$ where d is selected by the attacker and x_2' is extracted from adversarial proof of knowledge. We consider to have algebraic adversary $\mathcal{A}$ and whenever it sends β', it also sends (a, b, c), so in case of Corrupt, we rewrite $\beta = g^a X_1^b \alpha^c$. Then we check:

(a) If $X_1^b \alpha^c \neq X_1^{x_2'\mathsf{pw}_{cs}}$ the key is computed as
$\mathsf{Key} = X_1^a Y^{(b+d\mathsf{pw}_{cs})} = \alpha^{\frac{a}{(x_2'+d)\mathsf{pw}_{cs}}} Y^{(b+d\mathsf{pw}_{cs})}$. This means if $Y = g^{x^2}$ then the key is real and random otherwise.

(b) If If $X_1^b \alpha^c = X_1^{x_2'\mathsf{pw}_{cs}}$, we embed α in the key and we compute it as $\mathsf{Key} = \alpha^{\frac{a}{(x_2'+d)\mathsf{pw}_{cs}}}$.
$\mathcal{A}$ interpolates between the two games, meaning, if $Y = g^{x^2}$ then it is playing Game 7, otherwise it is playing Game 7-8.m.1.
The analysis is the same on the server's side. Furthermore, the bounds and other remarks for Game 7-8.1 and Game 7-8.2 do not change with respect to the same games in s-JPAKE. Thus, we have a bound for Game 7-8.1

$$\Pr[G_7 \Rightarrow \mathsf{T}] - \Pr[G_{7-8.m.1} \Rightarrow \mathsf{T}] \leq 2\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}()$$

And the bound for for Game 7-8.2 where the only way that $\mathcal{A}$ notice the difference between the two games if bad_5 happens. Therefore, everything stays the same:

$$|\Pr[G_{7-8.m.1} \Rightarrow T] - \Pr[G_{7-8.m.2} \Rightarrow T]| \leq \Pr[G_8 \Rightarrow \mathsf{bad}_5]$$

Game 9: Perfect Forward Secrecy.
Description of this game is the same as in Game 9 of sJ-PAKE, except for modification in the reduction to CSqDH problem. We state the differences with the reduction described in Lemma B.8:
Given the challenge X, we build an attacker $\mathcal{B}_9$ that finds solution to CSqDH problem. Let us say that $\mathcal{B}_9$ plugs-in the challenge in the lth session on the clients side, in $X_1 = X$ and it sets $\alpha = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. When $\mathcal{A}$ sends β' with (a, b, c), we rewrite it as $\beta' = g^a X_1^b \alpha^c$. Then, when the game finishes, $\mathcal{B}_9$ guesses one query for which bad_9^1 might have occurred. If the guess was successful it means $\mathcal{A}$ solved CSqDH and $\mathcal{B}_9$ can recover g^{x^2} in the following way:

$$g^{x^2} = \left(\frac{\mathsf{Key}}{X^{(a+wc)}}\right)^{\frac{1}{b-x_2'\mathsf{pw}_{cs}}}$$

We do the same procedure on the server side and we plug-in the challenge for $X_2 = X$ and set $\beta = g^w$, for $w \xleftarrow{\$} \mathbb{Z}_q$. The bound and other remarks do not change:

$$\Pr[G_8 \Rightarrow \mathsf{bad}_5] \leq \frac{2n_{se}}{|D|} + n_{se} n_{ro} \mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()$$

□

D.3 sCRS-J-PAKE

Description. At the beginning of the protocol, sCRS-J-PAKE sets up a common reference string U (as in CRS-J-PAKE). Basically, U plays the role of $(X_1 X_3)$. The first round is the same as in sRO-J-PAKE (RO-J-PAKE), with the values X_1, X_3 and its proofs π_1 and π_3 eliminated. In the second round, after receiving

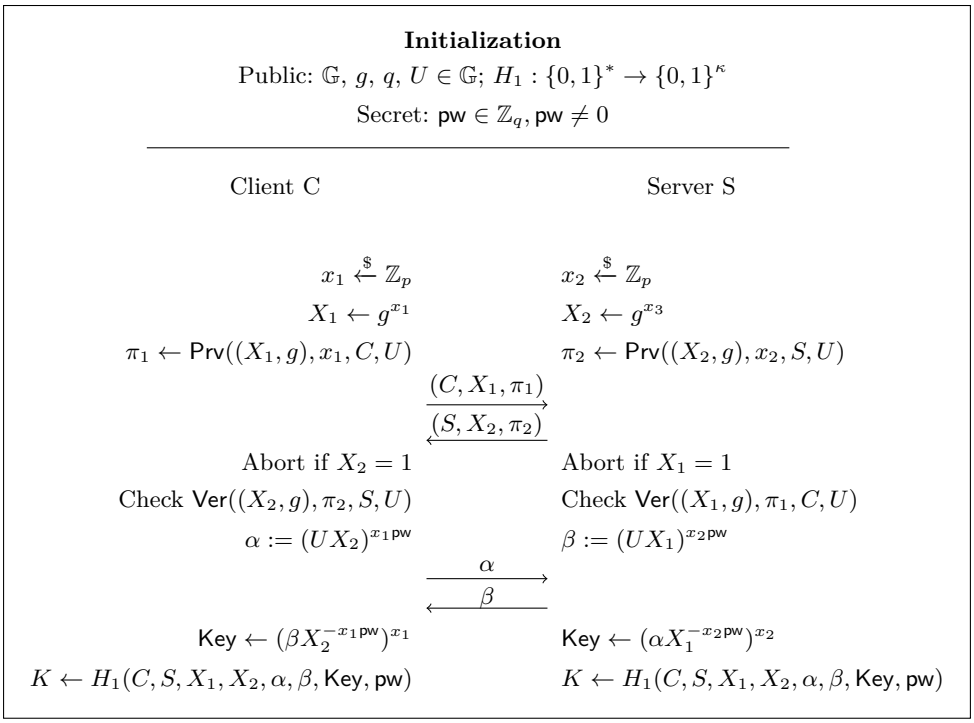

Fig. 9: The sCRS-J-PAKE protocol.

the first message, a Client computes $\alpha = (UX_2)^{x_1 pw}$ and a Server $\beta = (UX_1)^{x_2 pw}$ and they exchange α and β. Both parties compute the key as $\mathsf{Key} = (\beta X_2^{-x_1 \mathsf{pw}})^{x_1}$ (resp. $\mathsf{Key} = (\alpha X_1^{-x_2 \mathsf{pw}})^{x_2}$) and should hold the same session key, $K = H_1(C, S, X_1, X_2, \alpha, \beta, K, \mathsf{pw})$.

D.4 Security game-based proof of sCRS-J-PAKE

Theorem D.2. *Let sCRS-J-PAKE be the protocol described in Fig. 9. Take an RoR attacker $\mathcal{A}$ against sCRS-J-PAKE, making at most n_{se}, n_{ex}, n_{re}, n_{co}, n_{te} queries to* Send, Execute, Reveal, Corrupt, Test *and RO, respectively. For every such attacker $\mathcal{A}$, there exist attackers: $\mathcal{B}_4$ against Computational Triple Group Diffie-Hellman problem, $\mathcal{B}_{4.5.1}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_{4.5.2}$ against Decisional Diffie-Hellman problem, $\mathcal{B}_6$ against Computational Squared Diffie-Hellman problem, $\mathcal{B}_{7-8.1}$ against Decisional Squared Diffie-Hellman problem and $\mathcal{B}_9$ against Computational Squared Diffie-Hellman problem such that*

$$\begin{aligned}\mathrm{Adv}^{\mathtt{sCRS-J-PAKE}}_{\mathcal{A}}() \leq\ & \frac{2n_{se}}{|\,D\,|} + \frac{(2n_{se}+4n_{ex})^2}{q} \\ & + \mathrm{Adv}^{\mathrm{uzk}}_{\mathsf{NIZK}}() + 2n_{se}\mathrm{Adv}^{\mathrm{ext}}_{\mathsf{NIZK}}() \\ & + n_{ro}n_{ex}\mathrm{Adv}^{\mathtt{CTGDH}}_{\mathcal{B}_4}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.1}}() + \mathrm{Adv}^{\mathtt{DDH}}_{\mathcal{B}_{4.5.2}}() \\ & + n_{se}n_{ro}^2\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_6}() + 2n_{se}\mathrm{Adv}^{\mathtt{DSqDH}}_{\mathcal{B}_{7-8.1}}() + n_{se}n_{ro}\mathrm{Adv}^{\mathtt{CSqDH}}_{\mathcal{B}_9}()\end{aligned}$$

where $\mathrm{Adv}^{\mathrm{uzk}}$ *and* $\mathrm{Adv}^{\mathrm{ext}}$ *are advantages for the security of the SE-NIZK, formally defined in 3.6.*

Proof. The proof of sCRS-J-PAKE is the same as sRO-J-PAKE in Sec. D.2 with the differences:

– We do not need additional random oracle to simulate H_0. Instead, we have CRS U, known to both parties.

– In the reduction described in Lemma B.3, we embed a challenge Z in place of U in α (resp. β)
– We add Game 3.5 (explained below) where we explicitly save discrete log of U during the execution, needed for simulation in later hops.

Game 3.5: Keep the discrete log of the public parameter.
During the initialization phase described in Fig. 10 , the discrete log u is saved as a record (U, u) on the list for future hops. More precisely, the simulator can retrieve the record (U, u) from the list at any time of the execution of the protocol.

Initialization:
Choose $u \xleftarrow{\$} \mathbb{Z}_p$.
Compute $U = g^u$ and save the record (u, U) to the list.
For $C \in \mathcal{C}$, $S \in \mathcal{S}$: $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\kappa)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return U, CRS, C, S

Fig. 10: Initialization phase for sCRS-J-PAKE

□

E Games and adversaries for the proof of Theorem D.2.

Game 0: Original Protocol

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{crs} \xleftarrow{\$} \mathsf{NIZK.Setup}(1^\kappa)$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_1, g), x_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_2, g), x_2, C)$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_3, g), x_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_4, g), x_4, S)$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3 . X'_4)^{x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X'_1 X'_2 X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{/x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{/x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_1, g), x_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_2, g), x_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_3, g), x_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{Prv}(\mathsf{crs}, (X_4, g), x_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi^i_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi^i_U) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 11: Game 0

Game 1: Simulate ZK-PoK proofs.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X_1' X_2' X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crt}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1+x_3+x_4)x_2 pw_{cs}}$;
$\beta = g^{(x_1+x_2+x_3)x_4 pw_{cs}}$;
$\mathsf{Key} = g^{(x_1+x_3)x_2 x_4 pw_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 12: Game 1

Reduction for Game 1

Initialize (crs)
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\boxed{\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C);}$
$\boxed{\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\boxed{\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S);}$
$\boxed{\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4, S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow (X_1 X_3' X_4')^{x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2, C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow (X_1' X_2' X_3)^{x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\boxed{\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C);}$ $\boxed{\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C);}$
$\boxed{\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S);}$ $\boxed{\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S);}$
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1+x_3+x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1+x_2+x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1+x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = 1$ return 1;
Else return 0;
Abort.

Fig. 13: Reduction for Game 1

Game 1.5: Extract discrete logs from adversarial proofs.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List}_{\mathsf{Ext}} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
$bad_{1.5} = \mathsf{T}$; For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $bad_{1.5} = \mathsf{T}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $bad_{1.5} = \mathsf{T}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(\mathsf{S}, \mathsf{i}, \mathsf{C}, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $bad_{1.5} = \mathsf{T}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $bad_{1.5} = \mathsf{T}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$Key \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi^i_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi^i_U) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b' = b$;

Fig. 14: Game 1.5

Reduction for Game 1.5

Initialize $(\mathsf{crs}, \mathsf{td}_e)$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$\pi_1 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_1, C)$;
$\pi_2 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
$\pi_3 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_3, S)$;
$\pi_4 \leftarrow \mathsf{Sim}_{\mathsf{O}}(X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_3', \pi_3'\}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ do $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_4', \pi_4'\}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ Abort;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_1', \pi_1'\}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\mathsf{List}_{\mathsf{Ext}} \leftarrow \mathsf{List}_{\mathsf{Ext}} \cup \{X_2', \pi_2'\}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\alpha = g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, T)$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, T)$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(X', \pi') \xleftarrow{\$} \mathsf{List}_{\mathsf{Ext}}$ do $\mathsf{NIZK.Finalize}(X', \pi')$;
Return $b = b'$;

Fig. 15: Reduction for Game 1.5

Game 2: Force unique values

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
bad_2=F;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \perp)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \perp)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3')x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \perp)$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \perp)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
If $X_1 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then bad_2=T; $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then bad_2=T; $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\mathsf{Fresh}(\pi_U^i) = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 16: Game 2

Game 3: Adding freshness.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\boxed{\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr};}$
$\alpha \leftarrow g^{(x_1 + x_3' + x_4') x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\boxed{\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr};}$
$\beta \leftarrow g^{(x_1' + x_2' + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\boxed{\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T});}$
$\boxed{\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr};}$
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\boxed{\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T});}$
$\boxed{\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr};}$
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 = g^{x_1}$; $X_2 = g^{x_2}$; $X_3 = g^{x_3}$; $X_4 = g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \xleftarrow{\$} \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha = g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta = g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{Key} = g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T})$;
$\pi_S^i = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\boxed{\forall (j, V) \text{ s.t. } (\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid}) \text{ do } \pi_V^j.\mathsf{fr} = \mathsf{F};}$
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 17: Game 3

Game 4: Randomize session Keys for passive adversaries.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$T_e \leftarrow \{\}$; $bad_4 = \mathsf{F}$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ Abort;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \leftarrow g^{(x_1 + x_3 + x_4) x_2 \mathsf{pw}_{cs}}$;
$\beta \leftarrow g^{(x_1 + x_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$\mathsf{Key}^* \leftarrow g^{(x_1 + x_3) x_2 x_4 \mathsf{pw}_{cs}}$;
If $(\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}) \in T$ then $bad_4 = \mathsf{T}$;
$T_e \leftarrow T_e \cup \{\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T_e$ then $bad_4 = \mathsf{T}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 18: Game 4

Reduction for Game 4

Initialize $(X, Y, Z, D_{XY}, D_{XZ}, D_{YZ})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$}$ NIZK.Backdoor( );
$T_e \leftarrow \{\}$; $T_4 \leftarrow \{\}$; $bad_4 = \mathsf{F}$
$l \xleftarrow{\$} \{1, \ldots n_{ex}\}$; $p \leftarrow 0$; $r \leftarrow 0$;
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$r \leftarrow r + 1$;
If $r \neq l$
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $r = l$
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$r \leftarrow r + 1$;
If $r \neq l$
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $r = l$
$X_3 \leftarrow Y$; $x_3 \leftarrow \perp$; $X_4 \leftarrow Z$; $x_4 \leftarrow \perp$;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, (X_3', g), π_3', S) $= \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, (X_4', g), π_4', S) $= \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_3' \notin$ List then:
  $x_3' \leftarrow$ NIZK.Extract(crs, td_e, X_3', π_3', S);
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_4' \notin$ List then:
  $x_4' \leftarrow$ NIZK.Extract(crs, td_e, X_4', π_4', S);
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, (X_1', g), π_1', C) $= \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, (X_2', g), π_2', C) $= \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_1' \notin$ List then:
  $x_1' \leftarrow$ NIZK.Extract(crs, td_e, X_1', π_1', C);
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_2' \notin$ List then:
  $x_2' \leftarrow$ NIZK.Extract(crs, td_e, X_2', π_2', C);
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee$ $(C, S) \notin$ Corr;
Else
  Key $\leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
Else
  Key $\leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$p \leftarrow p + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $p = l$
  $X_2 \leftarrow X$; $X_3 \leftarrow Y$; $X_4 \leftarrow Z$;
  $x_2 \leftarrow \perp$; $x_3 \leftarrow \perp$; $x_4 \leftarrow \perp$;
Else
  $x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
  $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
If $x_2 = \perp \vee x_3 = \perp \vee x_4 = \perp$
  $\alpha \leftarrow (D_{XY} D_{XZ} X^{x_1})^{\mathsf{pw}_{cs}}$
  $\beta \leftarrow (D_{XZ} D_{YZ} Z^{x_1})^{\mathsf{pw}_{cs}}$
  sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
  If $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then $T_4 \leftarrow T_4 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}\}$;
Else
  $\alpha \leftarrow g^{(x_1 + x_3 + x_4)x_2 \mathsf{pw}_{cs}}$;
  $\beta \leftarrow g^{(x_1 + x_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
  sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
  $\mathsf{Key}^* \leftarrow g^{(x_1 + x_3)x_2 x_4 \mathsf{pw}_{cs}}$;
  If $(\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}) \in T$ then $bad_4 = \mathsf{T}$;
  Else $T_e \leftarrow T_e \cup \{\mathsf{sid}, \mathsf{Key}^*, \mathsf{pw}_{cs}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $\exists$ $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T_e$ then $T_4 \leftarrow T_4 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, pw] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow$ Reveal(U, i); $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_4$ do $\mathcal{B}_4$.Finalize(g^{xyz});
Abort.

Fig. 19: Reduction for Game 4

Game 4.5.1 Randomize α

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\boxed{\alpha \xleftarrow{\$} \mathcal{G};}$
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

$\mathbf{H}(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 20: Game 4.5.1

Reduction for Game 4.5.1.

Initialize $(X_1, \ldots X_{ex}, Y_1, \ldots, Y_{ex}, Z_1, \ldots, Z_{ex})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$k \leftarrow 0$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^j_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$k = k + 1$;
$X_1 \leftarrow X_k$; $x_1 \leftarrow \perp$;
$X_2 \leftarrow Y_k$; $x_2 \leftarrow \perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \leftarrow (Z_k X_2^{x_3 + x_4})^{\mathsf{pw}_{cs}}$;
$\beta \leftarrow X_2^{x_4 \mathsf{pw}_{cs}} g^{(x_1 + x_2) x_4 \mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = 1$ return 1;
Else return 0;
Abort.

Fig. 21: Reduction for Game 4.5.1

Game 4.5.2 Randomize β

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2($C, i, S, X'_3, X'_4, \pi'_3, \pi'_4$**)**
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\boxed{\beta \xleftarrow{\$} \mathcal{G};}$
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b'**)**
Return $b = b'$;

Fig. 22: Game 4.5.2

Reduction for Game 4.5.2

Initialize $(X_1, \ldots X_{ex}, Y_1, \ldots, Y_{ex}, Z_1, \ldots, Z_{ex})$
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$k \leftarrow 0$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^j_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$k = k + 1$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
$X_3 \leftarrow X_k$; | $x_3 \leftarrow \perp$;
$X_4 \leftarrow Y_k$; | $x_4 \leftarrow \perp$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$;
$\beta \leftarrow (Z_k X_4^{x_1 + x_2})^{\mathsf{pw}_{cs}}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$T_e \leftarrow T_e \cup \{\mathsf{sid}, k, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
If $b' = b$ return 1;
Else return 0;
Abort.

Fig. 23: Reduction for Game 4.5.2

Game 5: Randomize session Keys for Send queries.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$;
$T_s \leftarrow \{\}$; bad_5=F;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}(\,)$;
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_3 \notin$ List then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_4 \notin$ List then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4)x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If $X'_1 \notin$ List then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow$ Invalid;
If $X'_2 \notin$ List then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Key $\leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else
  $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
    Key* $\leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
    If Key* = Key then bad_5=T; Abort.
  $K \xleftarrow{\$} \mathcal{K}$;
  $T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Key $\leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x'_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else
  $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
    Key* $\leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
    If Key* = Key then bad_5=T; Abort.
  $K \xleftarrow{\$} \mathcal{K}$;
  $T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$
    Key* $\leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  If $T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$
    Key* $\leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  If Key* = Key then bad_5=T; Abort.
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
Return $b = b'$;

Fig. 24: Game 5

Game 6: Detect duplicates

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
bad_5=F; $bad_6 = \mathsf{F}$; $T_6 \leftarrow \{\}$;

For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;

$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
  $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
  $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K)$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$; Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $\exists\ \mathsf{pw} \neq \mathsf{pw}'$
  $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \xleftarrow{\$} T_6 \wedge (\mathsf{sid}, \mathsf{Key}', \mathsf{pw}') \xleftarrow{\$} T_6$ do $bad_6 = \mathsf{T}$;
If bad_6=F $\wedge$ $T_6 \neq \emptyset$ then bad_5=T;
Return $b = b'$;

Fig. 25: Game 6

Reduction for Game 6.

Initialize (X)

$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$;
$T_6 \leftarrow \{\}$; $T_{\mathsf{bad}} \leftarrow \{\}$;
$l \xleftarrow{\$} \{1, \ldots n_{se}\}$; $r \leftarrow 0$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)

If $\pi^i_C \neq \perp$ return $\perp$;
$r = r + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $r = l$
$X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
Else
$x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 = g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)

If $\pi^i_S \neq \perp$ return $\perp$;
$r = r + 1$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $r = l$
$X_4 \leftarrow X$; $x_4 \leftarrow \perp$;
Else
$x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$

If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_3 \notin \mathsf{List}$ then:
$x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X'_4 \notin \mathsf{List}$ then:
$x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $x_2 = \perp$
$\alpha \leftarrow X_2^{(x_1 + x'_3 + x'_4)\mathsf{pw}_{cs}}$;
Else
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4) x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$

If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_1 \notin \mathsf{List}$ then:
$x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X'_2 \notin \mathsf{List}$ then:
$x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
If $x_4 = \perp$
$\beta \leftarrow X_4^{(x'_1 + x'_2 + x_3)\mathsf{pw}_{cs}}$;
Else
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3) x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3(C, i, S, β')

If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
If $x_2 \neq \perp$
$\mathsf{Key} \leftarrow (\frac{\beta'}{X_4^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
If $x_2 = \perp$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then Abort. ("Wrong guess")
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
If $x_2 \neq \perp$
$\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $x_2 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K)$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3(S, i, C, α')

If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg \mathsf{fr}$ then
If $x_4 \neq \perp$
$\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
If $x_4 = \perp$
If $\exists\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \in T$ then Abort. ("Wrong guess")
Else if $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall\ (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
If $x_4 \neq \perp$
$\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$
If $x_4 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K)$;
$\pi^i_S = ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)

If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi^i_C \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi^j_S \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$

$\forall \mathsf{sid} \in T_s$ then
If $x_2 = \perp \vee x_4 = \perp$
$T_{\mathsf{bad}} \leftarrow T_{\mathsf{bad}} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)

$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)

If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)

If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')

For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $\exists\ \mathsf{pw} \neq \mathsf{pw}'$
$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \xleftarrow{\$} T_{\mathsf{bad}} \wedge (\mathsf{sid}, \mathsf{Key}', \mathsf{pw}') \xleftarrow{\$} T_{\mathsf{bad}}$ do
$\mathcal{B}_6.\mathsf{Finalize}(g^{x^2})$; Abort.
If $T_{\mathsf{bad}} \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
Return $b = b'$;

Fig. 26: Reduction for Game 6

Game 7: Adding algebraic representation.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; $T_6 \leftarrow \{\}$;
bad_5=F;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$}$ NIZK.Backdoor( );
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow$ Invalid;
If Ver(crs, $(X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If Ver(crs, $(X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_3 \notin$ List then:
$x'_3 \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X'_3, \pi'_3, S$);
If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_4 \notin$ List then:
$x'_4 \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X'_4, \pi'_4, S$);
If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\alpha \leftarrow g^{(x_1 + x'_3 + x'_4)x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow$ Invalid;
If Ver(crs, $(X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If Ver(crs, $(X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If $X'_1 \notin$ List then:
$x'_1 \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X'_1, \pi'_1, C$);
If $X'_1 \neq g^{x'_1}$ then $\pi^i_C \leftarrow$ Invalid;
If $X'_2 \notin$ List then:
$x'_2 \leftarrow$ NIZK.Extract(crs, $\mathsf{td}_e, X'_2, \pi'_2, C$);
If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$\beta \leftarrow g^{(x'_1 + x'_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3$(C, i, S, \beta', \boxed{\mathsf{alg}'})$
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j$, $(\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
Key $\leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}_{cs}}})^{x_2}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else $\forall \mathsf{sid} \in T_s$ then $(S, (x_3, x_4), K, \boxed{\mathsf{alg}}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
$\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}}})^{x_2}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (C, (x_1, x_2), K, \boxed{\mathsf{alg}'})$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3$(S, i, C, \alpha', \boxed{\mathsf{alg}'})$
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^i_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j$, $(\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
Key $\leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}_{cs}}})^{x_4}$;
$K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \boxed{\mathsf{alg}}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
$\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}})^{x_4}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \boxed{\mathsf{alg}'})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
If $\forall T_s[\mathsf{sid}] = (C, (x_1, x_2), K)$;
$\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4^{\prime x_2 \mathsf{pw}}})^{x_2}$;
If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K)$;
$\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2^{\prime x_4 \mathsf{pw}}})^{x_4}$;
If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, Key, \mathsf{pw}] = \perp$ then
$T[\mathsf{sid}, Key, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, Key, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow$ Reveal(U, i); $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then bad_5=T;
Return $b = b'$;

Fig. 27: Game 7

Game 7-8.1: Randomizing α and β.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; $\mathsf{Tst} \leftarrow \{\}$; $\mathsf{Corr} \leftarrow \{\}$; $\mathsf{List} \leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$; $t \leftarrow 0$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
$\mathsf{CRS} \leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$t \leftarrow t + 1$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S,i,C)
If $\pi_S^i \neq \perp$ return $\perp$;
$t \leftarrow t + 1$;
$X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_3' \notin \mathsf{List}$ then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4' \notin \mathsf{List}$ then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_1' \notin \mathsf{List}$ then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2' \notin \mathsf{List}$ then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
$\mathsf{fr} \leftarrow (C, S) \notin \mathsf{Corr}$;
$\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{(x_2 a + x_1 x_2 b)} = g^{x_2 a} \left(\frac{\frac{1}{\alpha^{\mathsf{pw}_{cs}}}}{g^{x_2(x_3' + x_4')}}\right)^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}}\right)^{x_2}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if $\mathsf{sid} \in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow \left(\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}}\right)^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
$\mathsf{fr} \leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
$\mathsf{fr} \leftarrow \mathsf{fr} \vee (C, S) \notin \mathsf{Corr}$;
If $\neg\mathsf{fr}$ then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{(x_4 a + x_3 x_4 b)} = g^{x_4 a} \left(\frac{\frac{1}{\beta^{\mathsf{pw}_{cs}}}}{g^{x_4(x_1' + x_2')}}\right)^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}}\right)^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else If $\mathsf{sid} \in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$
  $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_2 \in \mathsf{List}$ then $\pi_C^i \leftarrow \mathsf{Invalid}$;
If $X_3 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
If $X_4 \in \mathsf{List}$ then $\pi_S^j \leftarrow \mathsf{Invalid}$;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
$\mathsf{List} \leftarrow \mathsf{List} \cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}}\right)^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}}\right)^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
$\mathsf{Corr} \leftarrow \mathsf{Corr} \cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in \mathsf{Tst}$ return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
$\mathsf{Tst} \leftarrow \mathsf{Tst} \cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash \mathsf{Corr}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
Return $b = b'$;

Fig. 28: Game 7-8.1

Reduction for Game 7-8.m.1

Initialize (X, Y)
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$;
For $C \in \mathcal{C}$, $S \in \mathcal{S}$ do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi^i_C \neq \perp$ return $\perp$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $t \leq m$
  $X_2 \leftarrow X$;
If $t > m$
  $x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi^i_S \neq \perp$ return $\perp$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $t \leq m$
  $X_4 \leftarrow X$;
If $t > m$
  $x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X'_3, X'_4, \pi'_3, \pi'_4)$
If $\pi^i_C \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X'_4 = 1$ then $\pi^j_S \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_3, g), \pi'_3, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_4, g), \pi'_4, S) = \mathsf{F}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_3 \notin$ List then:
  $x'_3 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_3, \pi'_3, S)$;
  If $X'_3 \neq g^{x'_3}$ then $\pi^j_S \leftarrow$ Invalid;
If $X'_4 \notin$ List then:
  $x'_4 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_4, \pi'_4, S)$;
  If $X'_4 \neq g^{x'_4}$ then $\pi^j_S \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
If $t \leq m$
  $\alpha \leftarrow X_2^{(x_1 + x'_3 + x'_4)\mathsf{pw}_{cs}}$;
If $t > m$
  $\alpha \leftarrow g^{(x_1 + x'_3 + x'_4)x_2 \mathsf{pw}_{cs}}$;
$\pi^i_C \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X'_3, X'_4, \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X'_1, X'_2, \pi'_1, \pi'_2)$
If $\pi^i_S \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X'_2 = 1$ then $\pi^i_C \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_1, g), \pi'_1, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X'_2, g), \pi'_2, C) = \mathsf{F}$ then $\pi^i_C \leftarrow$ Invalid;
If $X'_1 \notin$ List then:
  $x'_1 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_1, \pi'_1, C)$;
  If $X'_1 \neq g^{x'_1}$ then then $\pi^i_C \leftarrow$ Invalid;
If $X'_2 \notin$ List then:
  $x'_2 \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X'_2, \pi'_2, C)$;
  If $X'_2 \neq g^{x'_2}$ then $\pi^i_C \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
If $t \leq m$
  $\beta \leftarrow X_4^{(x'_1 + x'_2 + x_3)\mathsf{pw}_{cs}}$;
If $t > m$
  $\beta \leftarrow g^{(x'_1 + x'_2 + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi^i_S \leftarrow ((x_3, x_4), (C, S, X'_1, X'_2, X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi^i_C.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi^i_C \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi^j_S.\mathsf{sid}) \wedge (\pi^j_S.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
    If $g^c (X_1 X'_3 X'_4)^{d \cdot \mathsf{pw}_{cs}} \neq X'^{\mathsf{pw}_{cs}}_4$ then
      $\mathsf{Key} = X_2^{a} \left(\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{X_2^{(x'_3 + x'_4)}}\right)^b Y^{(c + d(x_1 + x'_3 + x'_4)\mathsf{pw}_{cs} - x'_4 \mathsf{pw}_{cs})}$;
    Else
      $\mathsf{Key} = X_2^{a} \left(\frac{\alpha^{\frac{1}{\mathsf{pw}_{cs}}}}{X_2^{(x'_3 + x'_4)}}\right)^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\alpha'}{X'^{x_4 \mathsf{pw}_{cs}}}\right)^{x_2}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x'_3, x'_4), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow \left(\frac{\beta'}{X'^{x_2 \mathsf{pw}}_4}\right)^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi^i_C = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi^j_S.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi^j_S \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi^j_C.\mathsf{sid}) \wedge (\pi^j_C.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
    If $g^c (X'_1 X'_2 X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X'^{\mathsf{pw}_{cs}}_2$ then
      $\mathsf{Key} = X_4^{a} \left(\frac{\beta^{\frac{1}{\mathsf{pw}_{cs}}}}{X_4^{(x'_1 + x'_2)}}\right)^b Y^{(c + d(x'_1 + x'_2 + x_3)\mathsf{pw}_{cs} - x'_2 \mathsf{pw}_{cs})}$;
    Else
      $\mathsf{Key} = X_4^{a} \left(\frac{\beta^{\frac{1}{\mathsf{pw}_{cs}}}}{X_4^{(x'_1 + x'_2)}}\right)^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  Else if $t > m$
    $\mathsf{Key} \leftarrow \left(\frac{\alpha'}{X'^{x_4 \mathsf{pw}_{cs}}_2}\right)^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else if $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X'^{x_4 \mathsf{pw}}_2}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi^i_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi^i_C \neq \perp \vee \pi^j_S \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi^i_C \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi^j_S \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi^j_S \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi^i_C \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi^j_S \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H(sid,Key,pw)
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\beta'}{X'^{x_2 \mathsf{pw}}_4}\right)^{x_2}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow \left(\frac{\alpha'}{X'^{x_4 \mathsf{pw}}_2}\right)^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$;
Return pw_{cs};

Reveal(U, i)
If $\pi^i_U.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi^j_V.\mathsf{sid} = \pi^i_U.\mathsf{sid})$ do $\pi^j_V.\mathsf{fr} = \mathsf{F}$;
Return $\pi^i_U.\mathsf{K}$;

Test(U, i)
If $\pi^i_U.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$ Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
If $b' = b$ return 1;
Else return 0; Abort.
Return $b = b'$

Fig. 29: Reduction for Game 7-8.m.1

Game 7-8.m.2.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; $T_6 \leftarrow \{\}$;
$bad_5 = \mathsf{F}$;
$(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_3' \notin$ List then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_4' \notin$ List then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
If $t \leq m$
  $\alpha \xleftarrow{\$} \mathcal{G}$;
If $t > m$
  $\alpha \leftarrow g^{(x_1 + x_3' + x_4')x_2 \mathsf{pw}_{cs}}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F}, t)$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_1' \notin$ List then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_2' \notin$ List then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
If $t \leq m$
  $\beta \xleftarrow{\$} \mathcal{G}$;
If $t > m$
  $\beta \leftarrow g^{(x_1' + x_2' + x_3)x_4 \mathsf{pw}_{cs}}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr}, t)$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
    If $g^c (X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{x_2 a} (\frac{1}{g^{x_2 (x_3' + x_4')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  If $t > m$
    $\mathsf{Key} \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}_{cs}}})^{x_2}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr}, t)$ return $\perp$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  If $t \leq m$
    Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
    If $g^c (X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
      $K \xleftarrow{\$} \mathcal{K}$;
    Else
      $\mathsf{Key} = g^{x_4 a} (\frac{1}{\beta^{\mathsf{pw}_{cs}} / g^{x_4 (x_1' + x_2')}})^b$;
      $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
  If $t > m$
    $\mathsf{Key} \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}_{cs}}})^{x_4}$;
    $K \leftarrow H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr}, t)$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
$\mathsf{sid} = (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge \mathsf{pw} = \mathsf{pw}_{cs}$ then $bad_5 = \mathsf{T}$;
Return $b = b'$;

Fig. 30: Game 7-8.2

Game 9: Perfect forward secrecy.

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; $T_6 \leftarrow \{\}$; $T_9 \leftarrow \{\}$;
bad_5=F; bad_9^1=F; bad_9^2=F;
(crs, td_s, td_e) $\xleftarrow{\$}$ NIZK.Backdoor( );
CRS $\leftarrow (\mathbb{G}, g, q,$ crs);
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp,$ F, F);
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp,$ F, F);
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp,$ F, F) return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, $(X_3', g), \pi_3', S)$ = F then $\pi_S^j \leftarrow$ Invalid;
If Ver(crs, $(X_4', g), \pi_4', S)$ = F then $\pi_S^j \leftarrow$ Invalid;
If $X_3' \notin$ List then:
  $x_3' \leftarrow$ NIZK.Extract(crs, td_e, X_3', π_3', S);
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_4' \notin$ List then:
  $x_4' \leftarrow$ NIZK.Extract(crs, td_e, X_4', π_4', S);
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\alpha \leftarrow g^w$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp,$ F, fr);
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp,$ F, F) return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, $(X_1', g), \pi_1', C)$ = F then $\pi_C^i \leftarrow$ Invalid;
If Ver(crs, $(X_2', g), \pi_2', C)$ = F then $\pi_C^i \leftarrow$ Invalid;
If $X_1' \notin$ List then:
  $x_1' \leftarrow$ NIZK.Extract(crs, td_e, X_1', π_1', C);
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_2' \notin$ List then:
  $x_2' \leftarrow$ NIZK.Extract(crs, td_e, X_2', π_2', C);
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\beta \leftarrow g^w$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp,$ F, fr);
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp,$ F, fr) return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
  If $g^c(X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_2 a}(\frac{1}{g^{x_2(x_3'+x_4')}})^{\frac{1}{\mathsf{pw}_{cs}}b}$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K,$ T, fr);
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp,$ F, fr) return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
  If $g^c(X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_4 a}(\frac{1}{g^{x_4(x_1'+x_2')}})^{\frac{1}{\mathsf{pw}_{cs}}b}$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else if $\exists (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
  If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K,$ T, fr);
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_1 \leftarrow$ NIZK.Sim(crs, td_s, X_1, C);
$\pi_2 \leftarrow$ NIZK.Sim(crs, td_s, X_2, C);
$\pi_3 \leftarrow$ NIZK.Sim(crs, td_s, X_3, S);
$\pi_4 \leftarrow$ NIZK.Sim(crs, td_s, X_4, S);
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K,$ T, T);
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K,$ T, T);
Return sid;

H(sid, Key, pw)
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2'}$;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4'}$;
  If $\mathsf{Key}^* = \mathsf{Key}$
    If $(C, S) \notin$ Corr do $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
    If $(C, S) \in$ Corr $\wedge$ pw $= \mathsf{pw}_{cs} \wedge bad_9^2 =$ F
      then $T_9 \leftarrow T_9 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg})$;
    Rewrite $\mathsf{alg} = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c(X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} = X_2'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg})$;
    Rewrite $\mathsf{alg} = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c(X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} = X_4'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq$ T $\vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} =$ F;
Return $\pi_U^i.$K;

Test(U, i)
If $\pi_U^i.\mathsf{fr} =$ F return $\perp$;
$K_0 \leftarrow$ Reveal(U, i); $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
If $T_6 \neq \emptyset \wedge$ pw $= \mathsf{pw}_{cs}$ then bad_5=T;
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_9$ do bad_9^1=T;
If bad_9^2=T then bad_9^1=F;
Return $b = b'$;

Fig. 31: Game 9

Reduction for Game 9

Initialize
$b \xleftarrow{\$} \{0,1\}$;
$T \leftarrow \{\}$; Tst $\leftarrow \{\}$; Corr $\leftarrow \{\}$; List $\leftarrow \{\}$; Corr $\leftarrow \{\}$;
$T_6 \leftarrow \{\}$; $T_{\mathsf{bad}_9} \leftarrow \{\}$; $T_{\mathsf{bad}_5} \leftarrow \{\}$;
bad_9^2=F; $(\mathsf{crs}, \mathsf{td}_s, \mathsf{td}_e) \xleftarrow{\$} \mathsf{NIZK.Backdoor}()$;
CRS $\leftarrow (\mathbb{G}, g, q, \mathsf{crs})$;
Return CRS;

SendInit-C1(C, i, S)
If $\pi_C^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_1 \xleftarrow{\$} \mathbb{Z}_q$; $X_1 \leftarrow g^{x_1}$;
If $r = l$
  $X_2 \leftarrow X$; $x_2 \leftarrow \perp$;
Else
  $x_2 \xleftarrow{\$} \mathbb{Z}_q$; $X_2 = g^{x_2}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
List $\leftarrow$ List $\cup \{X_1, X_2\}$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(C, X_1, X_2, \pi_1, \pi_2)$;

SendInit-S1(S, i, C)
If $\pi_S^i \neq \perp$ return $\perp$;
$r = r + 1$;
$x_3 \xleftarrow{\$} \mathbb{Z}_q$; $X_3 \leftarrow g^{x_3}$;
If $r = l$
  $X_4 \leftarrow X$; $x_4 \leftarrow \perp$;
Else
  $x_4 \xleftarrow{\$} \mathbb{Z}_q$; $X_4 \leftarrow g^{x_4}$;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_3, X_4\}$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$;
Return $(S, X_3, X_4, \pi_3, \pi_4)$;

Send-C2$(C, i, S, X_3', X_4', \pi_3', \pi_4')$
If $\pi_C^i \neq ((x_1, x_2), (C, S, X_1, X_2, \perp, \perp, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_4' = 1$ then $\pi_S^j \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_3', g), \pi_3', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_4', g), \pi_4', S) = \mathsf{F}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_3' \notin$ List then:
  $x_3' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_3', \pi_3', S)$;
  If $X_3' \neq g^{x_3'}$ then $\pi_S^j \leftarrow$ Invalid;
If $X_4' \notin$ List then:
  $x_4' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_4', \pi_4', S)$;
  If $X_4' \neq g^{x_4'}$ then $\pi_S^j \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\alpha \leftarrow g^w$;
$\pi_C^i \leftarrow ((x_1, x_2), (C, S, X_1, X_2, X_3', X_4', \alpha, \perp), \perp, \mathsf{F}, \mathsf{fr})$;
Return α;

Send-S2$(S, i, C, X_1', X_2', \pi_1', \pi_2')$
If $\pi_S^i \neq ((x_3, x_4), (C, S, \perp, \perp, X_3, X_4, \perp, \perp), \perp, \mathsf{F}, \mathsf{F})$ return $\perp$;
If $X_2' = 1$ then $\pi_C^i \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_1', g), \pi_1', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $\mathsf{Ver}(\mathsf{crs}, (X_2', g), \pi_2', C) = \mathsf{F}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_1' \notin$ List then:
  $x_1' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_1', \pi_1', C)$;
  If $X_1' \neq g^{x_1'}$ then $\pi_C^i \leftarrow$ Invalid;
If $X_2' \notin$ List then:
  $x_2' \leftarrow \mathsf{NIZK.Extract}(\mathsf{crs}, \mathsf{td}_e, X_2', \pi_2', C)$;
  If $X_2' \neq g^{x_2'}$ then $\pi_C^i \leftarrow$ Invalid;
fr $\leftarrow (C, S) \notin$ Corr;
$w \xleftarrow{\$} \mathbb{Z}_q$; $\beta \leftarrow g^w$;
$\pi_S^i \leftarrow ((x_3, x_4), (C, S, X_1', X_2', X_3, X_4, \perp, \beta), \perp, \mathsf{F}, \mathsf{fr})$;
Return β;

Send-C3$(C, i, S, \beta', \mathsf{alg}')$
If $\pi_C^i.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \alpha, \perp)$ then $\perp$;
If $\pi_C^i \neq ((x_1, x_2), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta')$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_S^j.\mathsf{sid}) \wedge (\pi_S^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite $\mathsf{alg}' = [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$;
  If $g^c(X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} \neq X_4'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_2 a}(\frac{1}{\alpha^{\mathsf{pw}_{cs}} g^{x_2(x_3'+x_4')}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(S, (x_3', x_4'), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ do
  If $x_2 \neq \perp$
    $\mathsf{Key}^* \leftarrow (\frac{\beta'}{X_4'^{x_2 \mathsf{pw}}})^{x_2}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
  If $x_2 = \perp$
    $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_C^i = ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Send-S3$(S, i, C, \alpha', \mathsf{alg}')$
If $\pi_S^j.\mathsf{sid} \neq (C, S, X_1, X_2, X_3, X_4, \perp, \beta)$ then $\perp$;
If $\pi_S^i \neq ((x_3, x_4), \mathsf{sid}, \perp, \perp, \mathsf{F}, \mathsf{fr})$ return $\perp$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha', \beta)$;
fr $\leftarrow \exists j, (\mathsf{sid} = \pi_C^j.\mathsf{sid}) \wedge (\pi_C^j.\mathsf{fr} = \mathsf{T})$;
fr $\leftarrow$ fr $\vee (C, S) \notin$ Corr;
If $\neg$fr then
  Rewrite $\mathsf{alg}' = [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$;
  If $g^c(X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} \neq X_2'^{\mathsf{pw}_{cs}}$ then
    $K \xleftarrow{\$} \mathcal{K}$;
  Else
    $\mathsf{Key} = g^{x_4 a}(\frac{1}{\beta^{\mathsf{pw}_{cs}} g^{x_4(x_1'+x_2')}})^b$;
    $K = H(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs})$;
Else if sid $\in T_s$ then $(C, (x_1, x_2), K, \mathsf{alg}) \leftarrow T_s[\mathsf{sid}]$;
Else $\forall (\mathsf{sid}, \mathsf{Key}, \mathsf{pw}) \in T$ then
  If $x_4 \neq \perp$
    $\mathsf{Key}^* \leftarrow (\frac{\alpha'}{X_2'^{x_4 \mathsf{pw}}})^{x_4}$;
    If $\mathsf{Key}^* = \mathsf{Key}$ then $T_6 \leftarrow T_6 \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
  If $x_4 = \perp$
    $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
$K \xleftarrow{\$} \mathcal{K}$;
$T_s[\mathsf{sid}] \leftarrow (S, (x_3, x_4), K, \mathsf{alg}')$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{fr})$;
Return T;

Execute(C, S, i, j)
If $\pi_C^i \neq \perp \vee \pi_S^j \neq \perp$ return $\perp$;
$x_1, x_2, x_3, x_4 \xleftarrow{\$} \mathbb{Z}_q$;
$X_1 \leftarrow g^{x_1}$; $X_2 \leftarrow g^{x_2}$; $X_3 \leftarrow g^{x_3}$; $X_4 \leftarrow g^{x_4}$;
If $X_1 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_2 \in$ List then $\pi_C^i \leftarrow$ Invalid;
If $X_3 \in$ List then $\pi_S^j \leftarrow$ Invalid;
If $X_4 \in$ List then $\pi_S^j \leftarrow$ Invalid;
$\pi_1 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_1, C)$;
$\pi_2 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_2, C)$;
$\pi_3 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_3, S)$;
$\pi_4 \leftarrow \mathsf{NIZK.Sim}(\mathsf{crs}, \mathsf{td}_s, X_4, S)$;
List $\leftarrow$ List $\cup \{X_1, X_2, X_3, X_4\}$;
$\alpha \xleftarrow{\$} \mathcal{G}$; $\beta \xleftarrow{\$} \mathcal{G}$;
sid $= (C, S, X_1, X_2, X_3, X_4, \alpha, \beta)$;
$K \xleftarrow{\$} \mathcal{K}$;
$\pi_C^i \leftarrow ((x_1, x_2), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
$\pi_S^i \leftarrow ((x_3, x_4), \mathsf{sid}, K, \mathsf{T}, \mathsf{T})$;
Return sid;

H$(\mathsf{sid}, \mathsf{Key}, \mathsf{pw})$
$\forall \mathsf{sid} \in T_s$ then
  If $x_2 = \perp \vee x_4 = \perp$
    If $(C, S) \notin$ Corr do $T_{\mathsf{bad}_5} \leftarrow T_{\mathsf{bad}_5} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
    If $(C, S) \in$ Corr $\wedge$ pw $= \mathsf{pw}_{cs} \wedge bad_9^2 =$ F
      do $T_{\mathsf{bad}_9} \leftarrow T_{\mathsf{bad}_9} \cup \{\mathsf{sid}, \mathsf{Key}, \mathsf{pw}\}$;
If $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] = \perp$ then
  $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}] \xleftarrow{\$} \mathcal{K}$;
Return $T[\mathsf{sid}, \mathsf{Key}, \mathsf{pw}]$;

Corrupt(C, S)
Corr $\leftarrow$ Corr $\cup \{(C, S)\}$; $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
$\forall \mathsf{sid} \in T_s$ then
  If $T_s[\mathsf{sid}] = (C, (x_1, x_2), K, \mathsf{alg})$;
    Rewrite alg $= [((g, a), (X_3, b), (X_4, c)), (\beta, d)]$
    If $g^c(X_1' X_2' X_3)^{d \cdot \mathsf{pw}_{cs}} = X_2'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
  If $T_s[\mathsf{sid}] = (S, (x_3, x_4), K, \mathsf{alg}')$;
    Rewrite alg $= [((g, a), (X_1, b), (X_2, c)), (\alpha, d)]$
    If $g^c(X_1 X_3' X_4')^{d \cdot \mathsf{pw}_{cs}} = X_4'^{\mathsf{pw}_{cs}}$ do bad_9^2=T;
Return pw_{cs};

Reveal(U, i)
If $\pi_U^i.\mathsf{ac} \neq \mathsf{T} \vee (U, i) \in$ Tst return $\perp$;
$\forall (j, V)$ s.t. $(\pi_V^j.\mathsf{sid} = \pi_U^i.\mathsf{sid})$ do $\pi_V^j.\mathsf{fr} = \mathsf{F}$;
Return $\pi_U^i.\mathsf{K}$;

Test(U, i)
If $\pi_U^i.\mathsf{fr} = \mathsf{F}$ return $\perp$;
$K_0 \leftarrow \mathsf{Reveal}(U, i)$; $K_1 \xleftarrow{\$} \mathcal{K}$;
Tst $\leftarrow$ Tst $\cup (U, i)$;
Return K_b;

Finalize(b')
For $(C \times S) \in (\mathcal{C} \times \mathcal{S}) \backslash$Corr do $\mathsf{pw}_{cs} \xleftarrow{\$} \mathcal{P}$;
For $(\mathsf{sid}, \mathsf{Key}, \mathsf{pw}_{cs}) \xleftarrow{\$} T_{\mathsf{bad}_9}$ do $\mathcal{B}_9.\mathsf{Finalize}(g^{x^2})$;
Return $b = b'$;

Fig. 32: Reduction for Game 9