Dual-System Simulation-Soundness with Applications to UC-PAKE and More

Charanjit S. Jutla
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
csjutla@us.ibm.com

Arnab Roy
Fujitsu Laboratories of America
Sunnyvale, CA 94058, USA
arnab@cs.stanford.edu

Abstract

We introduce a novel concept of dual-system simulation-sound non-interactive zero-knowledge (NIZK) proofs. Dual-system NIZK proof system can be seen as a two-tier proof system. As opposed to the usual notion of zero-knowledge proofs, dual-system defines an intermediate partial-simulation world, where the proof simulator may have access to additional auxiliary information about the potential language member, for example a membership bit, and simulation of proofs is only guaranteed if the membership bit is correct. Further, dual-system NIZK proofs allow a quasi-adaptive setting where the CRS can be generated based on language parameters. This allows for the further possibility that the partial-world CRS simulator may have access to further trapdoors related to the language parameters. We show that for important hard languages like the Diffie-Hellman language, such dual-system proof systems can be given which allow unbounded partial simulation soundness, and which further allow transition between partial simulation world and single-theorem full simulation world even when proofs are sought on non-members. The construction is surprisingly simple, involving only two additional group elements in asymmetric bilinear pairing groups.

As a first application we show a first single-round universally-composable password authenticated key-exchange (UC-PAKE) protocol which is secure under dynamic corruption in the erasure model. The single message flow only requires four group elements under the SXDH assumption, which is at least two times shorter than earlier schemes.

As another application we give a short keyed-homomorphic CCA-secure encryption scheme. The ciphertext in this scheme consist of only six group elements (under the SXDH assumption) and the security reduction is linear-preserving. An earlier scheme of Libert et al based on their efficient unbounded simulation-sound QA-NIZK proofs only provided a quadratic-preserving security reduction, and further had ciphertexts almost twice as long as ours.

Keywords: NIZK, bilinear pairings, UC-PAKE, keyed-homomorphic encryption, SXDH.

Contents

1 Introduction

Since the introduction of simulation-sound non-interactive zero-knowledge (NIZK) proofs in [Sah99], simulation-soundness has become an essential cryptographic tool. While the idea of zero-knowledge simulation [GMR89] brought rigour to the concept of semantic security, simulation-soundness of some form is usually implict in most cryptographic applications. While the original construction of [Sah99] was rather inefficient, the advent of pairing based cryptography, and in particular Groth-Sahai NIZK proofs [GS08], has led to much more efficient simulation-sound NIZK constructions. Pairing-based cryptography has also led to efficient construction of powerful primitives where simulation-soundness is not very explicit.

It has been shown that different forms of simulation-soundness suffice for many applications. Indeed, the original application (CCA2-secure encryption) considered in [Sah99] only required what is known as single-theorem simulation-soundness (also known as one-time simulation-soundness). However, many other cryptographic constructions are known only using unbounded simulation-sound NIZK proofs. In this paper, we introduce the concept of **dual-system simulation-sound** NIZK proof systems, which lie somewhere in between one-time and unbounded simulation-soundness. The aim is to show that this weaker concept suffices for constructions where unbounded simulation-soundness was being used till now. We also show that in many applications this new concept of dual-system simulation soundness is implicit, in the sense that we cannot get a generic construction from a NIZK proof, but can use the underlying ideas of the dual-system simulation-sound NIZK proofs.

Indeed, our novel definition is inspired by the dual-system identity-based encryption (IBE) scheme of Waters [Wat09], where such a concept was implicit, and led to the first IBE scheme which was fully-secure under static and standard assumptions. So without further ado, we jump straight into the main idea of the new concept. In dual-system simulation-sound NIZK proof systems we will consider three worlds: the real-world, the partial-simulation world, and the one-time full-simulation world. The real world consists of a common-reference string (CRS), an efficient prover P, and an efficient verifier V. The concept of completeness and soundness of P and V with respect to a witness-relation R is well-understood. The full-simulation world is also standard, and it includes two simulators: a CRS simulator and a proof simulator. The proof simulator is a zero-knowledge simulator in that it can simulate proofs even without access to the witness. In order to achieve this, the CRS simulator generates the CRS in a potentially different way and produces a trapdoor for the proof simulator. The **partial-simulation** world we consider also has a CRS simulator, and a proof simulator, but this proof simulator is allowed partial access to the witness (or some other auxiliary information) about the member on which the proof is sought.

At this point, we also bring in the possibility of the CRS being generated as a function of the language or witness-relation under consideration. The recent quasi-adaptive NIZK (QA-NIZK) proofs of [JR13] allow this possibility for distributions of witness-relations. The CRS in the real and the full-simulation world is generated based on a language parameter generated according to some distribution. Now we consider the possibility that in the partial-simulation world, the CRS simulator actually generates the language parameter itself. In other words, the CRS simulator has access to the "witness" of the language parameter. For example, the CRS simulator may know the discrete-logs of the language parameters. This leads to the possibility that in the partial simulation world the proof simulator may have access to further trapdoors which makes simulation and/or simulation soundness easier to achieve.

In this paper, we will only define and consider dual-system simulation sound QA-NIZK proofs

(called DSS-QA-NIZK), where the only auxiliary information that the partial proof simulator gets is a single bit which is called the **membership bit**. The membership bit indicates whether the potential member on which the proof is sought is in the language or not. We show that we can achieve unbounded partial-simulation soundness for important languages like the Diffie-Hellman language (and more general such languages) by relatively simple constructions. The constructions also allow one-time full-ZK simulation, and hence form a DSS-QA-NIZK for the Diffie-Hellman language. Indeed, for the Diffie-Hellman language, under the standard SXDH assumption in bilinear pairing groups, the DSS-QA-NIZK only requires two more group elements.

We now give the main idea as to why such a construction is useful. Most applications only need a full-simulation NIZK proof for the hard language under whose hardness the application is to be proven secure. However, the particular application may have a more complex language for which the NIZK proofs are required, and the security proof may require soundness of the NIZK system while proofs of many elements (real or fake) of such a complex language are being simulated. The idea is that multiple simulations of such elements can be performed in a partial-simulation manner, and full simulation is only required of one member at a time, and that too of the underlying hard language member.

Applications.

1. Keyed-Homomorphic CCA-secure Encryption. As a first application we consider the keyed-homomorphic CCA-secure encryption scheme notion of [EHO+13]. In such an encryption scheme, a further functionality called Eval is available which using a key can homomorphically combine valid ciphertexts. The scheme should provide IND-CCA2 security when this Eval key is unavailable to the adversary, and should continue to enjoy IND-CCA1 security when the Eval key is exposed to the adversary. Emura et al. also gave constructions for such a scheme, albeit schemes which are not publicly verifiable, and further satisfying a weaker notion than CCA1-security when Eval key is revealed. Recently, Libert et al gave a publicly-verifiable construction which is more efficient and also CCA1-secure when Eval key is revealed. Their construction is based on a new and improved unbounded simulation-sound QA-NIZK for linear subspace languages. We show in this paper that ideas from DSS-QA-NIZK for the Diffie-Hellman language suffice, and leads to a much improved construction. While the construction in [LPJY14], under the SXDH assumption, requires nine group elements in one group, and two more in the other plus a one-time signature key pair, our construction *only requires six group elements* in any one of the bilinear groups. Further, while the earlier construction only had a quadratic-preserving security reduction, our reduction is linear-preserving.

The main idea of our construction is as follows. The ciphertexts consist of an El-Gamal encryption of the message M, say $\mathbf{g}^r, M \cdot \mathbf{g}^{k \cdot r}$ for a public key $\mathbf{g}^k$. The public key also consists of a member $\mathbf{g}^a$, and the ciphertext includes $\mathbf{g}^{a \cdot r}$. It is well-known [JR12] that if a one-time simulation-sound NIZK proof of $\mathbf{g}^r$ and $\mathbf{g}^{a \cdot r}$ being of the correct form is included in the ciphertext then it becomes a publicly-verifiable CCA2-secure encryption scheme. In our keyed-homomorphic construction we include a DSS-QA-NIZK for $\mathbf{g}^r$ and $\mathbf{g}^{a \cdot r}$ being of the correct form, which is just the requirement that this be a Diffie-Hellman tuple. The DSS-QA-NIZK we give for this language has the nice property that the CRS in the real-world and the partial world are statistically identical. Thus, the public key can be generated by the partial-world CRS simulator, and the partial-simulation trapdoor it generates can be the Eval key. Since the partial simulation world is unbounded simulation-sound, the Eval functionality can be provided by first verifying that the Eval is performed on valid ciphertexts, homomorphically combining the triples of the form $\mathbf{g}^r, \mathbf{g}^{a \cdot r}, M \cdot \mathbf{g}^{k \cdot r}$, and then calling

the partial-simulator with the membership bit set as one (and this is a valid membership bit as the partial-world verifier is unbounded simulation sound). We also show that our DSS-QA-NIZK construction remains CCA1-style secure when the partial-simulator trapdoor is revealed.

2. Universally-Composable Password-Authenticated Key Exchange (UC-PAKE) The UC-PAKE ideal functionality was introduced in [CHK$^+$05] where they also gave a three round construction. In [KV11] a single-round construction for UC-PAKE was given using Groth-Sahai NIZK proofs along with unbounded simulation-soundness construction of [CCS09]. In [JR12], further improvement was obtained by generating the key in the target bilinear-group. The construction still required unbounded simulation-soundness. Later [BBC$^+$13] gave a construction based on novel trapdoor smooth projective hash functions instead of unbounded simulation-sound NIZK proofs. Their construction yielded a single-round UC-secure PAKE scheme under static corruption, and each message consisted of six group elements in one group, and another five elements in the other group, under the SXDH assumption. The key is generated in the target group.

We now give a single-round construction based on dual-system simulation-soundness which is UC-secure under dynamic corruption (in the erasure model), and which has only a total of four group elements in each message (three in one group and one in the other). The key is generated in the target group. The construction is not a generic construction from the DSS-QA-NIZK for the Diffie-Hellman language, but uses its underlying idea as well as the various component algorithms of the DSS-QA-NIZK. The main idea of the construction is given in more detail in Section 5.3. All previous constructions, with the exception of a two-round construction of [ACP09], were only secure in the static corruption model.

1.1 Dual-System Simulation-Soundness

A witness relation is a binary relation on pairs of inputs, the first called the witness and the second called a language member. Note that each witness relation R defines a corresponding language L which is the set of all x for which there exists a witness w, such that $R(w, x)$ holds.

We will consider Quasi-Adaptive NIZK proofs for a probability distribution $\mathcal{D}$ on a collection of (witness-) relations $\mathcal{R} = \{R_\rho\}$ [JR13]. Recall that in a quasi-adaptive NIZK, the CRS can be set after the language parameter has been chosen according to $\mathcal{D}$. To define dual-system simulation soundness of such NIZK proofs, we will consider three worlds: the real-world, the partial-simulation world, and the one-time (or single theorem) full-simulation world. While the real-world and the full-simulation world should be familiar from earlier definitions of NIZK proof systems or quasi-adaptive NIZK proof systems, the partial-simulation world leads to interesting possibilities. To start with, in the partial simulation world, one would like the proof simulator to have access to partial or complete witness of the potential language member. In case, the proof simulator is being invoked on a non-language member, it is not immediately clear what this witness can be, unless we also define a language and a distribution for a super-language which includes the language under consideration as a subset. Finally, in the quasi-adaptive setting, the language parameters may actually be generated by the CRS simulator and hence the simulator may have access to, say, the discrete logs of the language parameters which can serve as trapdoors.

Rather than consider these general settings, we focus on a simple partial-simulation setting, where (a) the CRS simulator can generate the language parameters itself and (b) the proof simulator when invoked with a potential language member x is given an additional bit β, which we call the **membership bit** which represents the information whether the potential language member x is indeed a member or not.

The partial simulation world is required to be unbounded simulation-sound and hopefully it should be easier to prove (given that its simulators have additional information) than usual unbounded simulation-soundness. We also allow the partial simulation world to be sound with respect to a private verifier (this concept has been considered earlier in [JR12]), and this further leads to the possibility of easier and/or simpler constructions.

A surprising property achievable under such a definition is that one can go back and forth between the partial-simulation world and the one-time full-simulation world even when simulating fake tuples. However, between the real-world and the partial-simulation world one must only pass the prover/simulator real language members.

Dual-system Simulation-Sound Quasi-Adaptive NIZK (DSS-QA-NIZK). A Dual-system proof system for a collection of witness-relations $\mathcal{R}_\lambda = \{R_\rho\}$, with parameters sampled from a distribution $\mathcal{D}$ over associated parameter language $\mathcal{L}_{\rm par}$ consists of:

- Following real-world PPT components:
  A pair of **CRS generators** $(\mathsf{K}_0, \mathsf{K}_1)$, where K_0 takes a unary string and produces an ensemble parameter λ. The ensemble parameter λ is used to sample a witness-relation parameter ρ using $\mathcal{D}_\lambda$. Next, K_1 uses ρ (and λ) to produce the real-world CRS ψ. A **prover** P that takes as input a CRS, a language member and its witness and produces a proof. A **verifier** V that takes as input a CRS, a potential language member, and a proof, and outputs a single bit.

- In addition, there are the following partial-simulation world PPT components:
  A **semi-functional CRS simulator** sfK_1 that takes ensemble parameter λ as input and produces a witness relation parameter ρ, a semi-functional CRS σ, as well as two trapdoors τ and η. The first trapdoor is used by the proof simulator, and the second by the private verifier. A **semi-functional simulator** sfS that takes a CRS, a trapdoor τ, a potential language member and a membership-bit β to produce a proof. A **private verifier** pV that takes a CRS, a trapdoor η, a potential language member and a proof and outputs a single bit.

- Finally, there are the following one-time full simulation PPT components:
  A **one-time full-simulation CRS generator** otfK_1, that takes as input the ensemble parameter λ, the witness relation parameter ρ to produce a CRS and three trapdoors τ, τ_1 and η. A **one-time full simulator** otfS that takes as input a CRS, a trapdoor τ_1, and a potential language member and produces a proof. A **semi-functional verifier** sfV that takes as input a CRS, a trapdoor η, a potential language member, a proof and outputs a bit. The adversaries also have access to semi-functional simulator.

All of the above provers, proof simulators and verifiers may also take a **label** l as an additional argument.

The definition of the real-world quasi-adaptive system to be complete and (computationally) sound are standard and can be found in [JR13]. Such a proof system is called a **dual-system simulation-sound quasi-adaptive NIZK** for a collection of witness relations $\mathcal{R}_\lambda = \{R_\rho\}$, with parameters sampled from a distribution $\mathcal{D}$, if for all non-uniform PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4)$ all of the following properties are satisfied:

- *partial-ZK:*
  $\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); \rho \leftarrow \mathcal{D}_\lambda;\ \psi \leftarrow \mathsf{K}_1(\lambda, \rho);\ \ \mathcal{A}_1^{\mathsf{P}(\psi,\cdot,\cdot;\cdot),\, \mathsf{V}(\psi,\cdot,\cdot;\cdot)}(\psi) = 1] \approx$

$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); (\rho,\sigma,\tau,\eta) \leftarrow \mathsf{sfK}_1(\lambda);\ \mathcal{A}_1^{\mathsf{X}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{pV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\sigma) = 1]$,
where the calls to P (with inputs a witness and a language member) are restricted to ones satisfying R_ρ, **and** $\mathsf{X}(\sigma,\tau,x,w;l)$ is defined to be $\mathsf{sfS}(\sigma,\tau,x,R_\rho(w,x);l)$ (i.e. witness is dropped, and membership bit β is just 1),

- *unbounded partial-simulation (relative) soundness:*
$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); (\rho,\sigma,\tau,\eta) \leftarrow \mathsf{sfK}_1(\lambda);\ (x,l,\pi) \leftarrow \mathcal{A}_2^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{pV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\sigma)$:
$[\neg\exists w \text{ s.t. } R_\rho(w,x) = 1, \text{ and } \mathsf{pV}(\sigma,\eta,x,\pi;l) = 1] \approx 0$.

- *one-time full-ZK:*
$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); (\rho,\sigma,\tau,\eta) \leftarrow \mathsf{sfK}_1(\lambda);\ (x,l,\beta,s) \leftarrow \mathcal{A}_3^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{pV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\sigma);$
$\pi \leftarrow \mathsf{sfS}(\sigma,\tau,x,\beta;l) : \mathcal{A}_4^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{pV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\pi,s) = 1] \approx$
$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); \rho \leftarrow \mathcal{D}_\lambda;\ (\sigma,\tau,\tau_1,\eta) \leftarrow \mathsf{otfK}_1(\lambda,\rho);\ \ (x,l,\beta,s) \leftarrow \mathcal{A}_3^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{sfV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\sigma);$
$\pi \leftarrow \mathsf{otfS}(\sigma,\tau_1,x;l) : \mathcal{A}_4^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot;\cdot),\ \mathsf{sfV}(\sigma,\eta,\cdot,\cdot;\cdot)}(\pi,s) = 1]$,
where β is a correct L_ρ-membership bit for x, **and** all calls to sfS also have correct L_ρ-membership bits **and** $\langle x,\pi;l\rangle$ is not queried to sfV/pV. Here s is a state variable.

Remark 1. Note that there is *no restriction* in the unbounded partial-simulation soundness definition of x, l (along with a β) being called to the first oracle sfS and returning π as proof.
Remark 2. Note that in the partial-ZK definition, the calls to the prover are restricted to ones satisfying the relation. However, the calls to the provers (i.e. sfS and otfS) in the one-time full-ZK definition are only restricted to having the correct membership bit β. In particular, one can go back and forth with non-members.
Remark 3. We now show that sfS generated proofs on potential language members (whether members or not, i.e. β being 1 or 0) are accepted by real-world verifier V (with semi-functional CRS σ). Of course, the private verifier pV will *even* reject proofs generated on non-language members by sfS. This justifies the name "semi-functional simulator". To be precise, for distributions $\mathcal{E}$ of fake language members (i.e. $\beta = 0$) such that with high probability no efficient adversary can distinguish them from real language members (i.e. $\beta = 1$), the semi-functional proofs generated on such tuples are accepted by V. The proof is standard and uses both the partial-ZK and the one-time full ZK property, and will be given in the full version of the paper.

It can also be shown that the semi-functional verifier sfV is still complete, i.e. it accepts language members and proofs generated on them by $\mathsf{P}(\sigma,\cdot,\cdot;\cdot)$ (i.e. P instantiated with CRS generated by otfK_1). As opposed to P and pV, it may no longer be sound. This justifies the name "semi-functional verifier" a la Waters' dual-system IBE construction.

2 Dual-System Simulation-Sound QA-NIZK for DH Language

We will consider bilinear groups $(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T)$, with an efficiently computable pairing e from $\mathbb{G}_1\times\mathbb{G}_2$ to $\mathbb{G}_T$. Each group will be assumed to be cyclic with prime order q. For the construction to be secure, we will also assume that the external Diffie-Hellman (XDH) assumption holds for group $\mathbb{G}_1$. Essentially, it says that the DDH assumption holds in group $\mathbb{G}_1$. Later, we also give a construction based on the decisional linear (DLIN) assumption.

We consider the following class of languages parameterized by $\mathbf{g}_2, \mathbf{g}_2^a$ $(\in \mathbb{G}_2^2)$. The **Diffie-Hellman language** corresponding to one such parameter $\rho = \langle \mathbf{g}_2, \mathbf{g}_2^a\rangle$ is $L_\rho = \{\langle \mathbf{g}_2^r, \mathbf{g}_2^{a\cdot r}\rangle \,|\, r \in \mathbb{Z}_q\}$.

Note that the witness relation R_ρ corresponding to this language is defined by $R_\rho(r, \langle \mathbf{x}_1, \mathbf{x}_2 \rangle)$ being true iff $\mathbf{x}_1 = \mathbf{g}_2^r$ and $\mathbf{x}_2 = (\mathbf{g}_2^a)^r$.

The distribution $\mathcal{D}$ under which the DSS-QA-NIZK is obtained is defined by picking $\mathbf{g}_2$ as a generator for $\mathbb{G}_2$ according to a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ generation algorithm for which DDH assumption holds for $\mathbb{G}_1$, and then choosing a randomly from $\mathbb{Z}_q$.

We now define the various components of the DSS-QA-NIZK system for the above class of Diffie-Hellman languages under distribution $\mathcal{D}$. In all the algorithms below l will denote label, the additional input.

- The real-world algorithms:
  - The algorithm K_0 is just the group generation algorithm (it takes a unary string 1^m as input), and it also generates a collision-resistant hash function $\mathcal{H}$. The CRS generation algorithm K_1 takes language parameter $\rho = \langle \mathbf{g}_2, \mathbf{g}_2^a \rangle$ and other bilinear group parameters as input and generates the CRS as follows: it picks $c, b, d, e, \Delta_1', \Delta_2'$ randomly and independently from $\mathbb{Z}_q$ and sets the CRS to be $(\mathbf{CRS}_p, \mathbf{CRS}_v, \mathcal{H})$:
    $\mathbf{CRS}_p = \{\rho,\ \mathbf{d} = \mathbf{g}_2^d,\ \mathbf{e} = \mathbf{g}_2^e,\ \mathbf{w}_1 = \mathbf{g}_2^{(\Delta_1'+d)/b} \cdot (\mathbf{g}_2^a)^{c/b},\ \mathbf{w}_2 = \mathbf{g}_2^{(\Delta_2'+e)/b}\}$,
    $\mathbf{CRS}_v = \{\mathbf{g}_1,\ \mathbf{c} = \mathbf{g}_1^c,\ \mathbf{b} = \mathbf{g}_1^b,\ \mathbf{v}_1 = \mathbf{g}_1^{-\Delta_1'},\ \mathbf{v}_2 = \mathbf{g}_1^{-\Delta_2'}\}$.
  - The **prover** P takes as input $\mathbf{CRS}_p$, a language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and its witness r and produces a proof π consisting of two group $\mathbb{G}_2$ elements T and W as follows: Compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$. Next, compute $T = (\mathbf{d} \cdot \mathbf{e}^\iota)^r$, and $W = (\mathbf{w}_1 \cdot \mathbf{w}_2^\iota)^r$.
  - The **verifier** V takes as input $\mathbf{CRS}_v$, a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$, and a proof $\pi = (T, W)$, computes $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$, and outputs true iff
    $$e((\mathbf{v}_1 \mathbf{v}_2^\iota), \mathbf{x}_1^{-1}) \cdot e(\mathbf{c}, \mathbf{x}_2) \cdot e(\mathbf{g}_1, T) \ = \ e(\mathbf{b}, W).$$
- The partial-simulation world PPT components:
  - The **semi-functional CRS simulator** sfK_1 takes group parameters and $\mathcal{H}$ as input and produces a witness relation parameter $\rho = \langle \mathbf{g}_2, \mathbf{g}_2^a \rangle$ by picking a at random from $\mathbb{Z}_q$. It produces the semi-functional CRS $\sigma = (\mathbf{CRS}_p, \mathbf{CRS}_v, \mathcal{H})$ as follows:
    it further picks $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ randomly and independently from $\mathbb{Z}_q$, and computes $d = d_1 + a \cdot d_2$, $e = e_1 + a \cdot e_2$, $\Delta_1 = (\Delta_1' + d + a \cdot c)/b$, $\Delta_2 = (\Delta_2' + e)/b$, and sets the CRS to be $(\sigma_p, \sigma_v, \mathcal{H})$:
    $\sigma_p = \{\rho,\ \mathbf{d} = \mathbf{g}_2^d,\ \mathbf{e} = \mathbf{g}_2^e,\ \mathbf{w}_1 = \mathbf{g}_2^{\Delta_1},\ \mathbf{w}_2 = \mathbf{g}_2^{\Delta_2}\}$,
    $\sigma_v = \{\mathbf{g}_1,\ \mathbf{c} = \mathbf{g}_1^c,\ \mathbf{b} = \mathbf{g}_1^b,\ \mathbf{v}_1 = \mathbf{g}_1^{-\Delta_1'},\ \mathbf{v}_2 = \mathbf{g}_1^{-\Delta_2'}\}$.
    It outputs $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ as proof simulator trapdoors τ, and outputs a, c, b, d, e, Δ_1, Δ_2 as private-verifier trapdoors η.
  - The **semi-functional simulator** sfS uses trapdoors $\{c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'\}$ to produce a (partially-simulated) proof for a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a binary bit β as follows: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$; if $\beta = 1$, compute and output
    $$T = \mathbf{x}_1^{d_1 + \iota e_1} \cdot \mathbf{x}_2^{d_2 + \iota e_2}\ ,\ W = \mathbf{x}_1^{(\Delta_1' + d_1 + \iota \cdot (\Delta_2' + e_1))/b} \cdot \mathbf{x}_2^{(d_2 + \iota e_2 + c)/b},$$
    else choose y at random from $\mathbb{Z}_q$; Compute and output
    $$T = \mathbf{x}_1^{d_1 + \iota e_1} \cdot \mathbf{g}_2^y\ ,\ W = \mathbf{x}_1^{(\Delta_1' + d_1 + \iota \cdot (\Delta_2' + e_1))/b} \cdot \mathbf{x}_2^{c/b} \cdot \mathbf{g}_2^{y/b}.$$
    This proof is partially simulated as it uses the bit β.

– The **private verifier** pV uses trapdoors $\{a, c, b, d, e, \Delta_1, \Delta_2\}$ to check a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a proof T, W as follows: it outputs 1 iff $\mathbf{x}_2 = \mathbf{x}_1^a$ and $T = \mathbf{x}_1^{d+\iota e}$ and $W = \mathbf{x}_1^{\Delta_1 + \iota \Delta_2}$, where $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$.

- Finally, the one-time full simulation PPT components:

  – The **one-time full-simulation CRS generator** otfK_1 takes as input the bilinear group parameters, $\mathcal{H}$, and the language parameter $\rho = \langle \mathbf{g}_2, \mathbf{g}_2^a \rangle$ to produce the CRS as in the real-world, but with a little twist: it picks $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ randomly and independently from $\mathbb{Z}_q$ and sets the CRS to be $(\sigma_p, \sigma_v, \mathcal{H})$:
  $\sigma_p = \{\rho, \mathbf{d} = \mathbf{g}_2^{d_1} \cdot (\mathbf{g}_2^a)^{d_2}, \mathbf{e} = \mathbf{g}_2^{e_1} \cdot (\mathbf{g}_2^a)^{e_2}, \mathbf{w}_1 = \mathbf{g}_2^{(\Delta_1' + d_1)/b} \cdot (\mathbf{g}_2^a)^{(d_2+c)/b}, \mathbf{w}_2 = \mathbf{g}_2^{(\Delta_2' + e_1)/b} \cdot (\mathbf{g}_2^a)^{e_2/b}\}$,
  $\sigma_v = \{\mathbf{g}_1, \mathbf{c} = \mathbf{g}_1^c, \mathbf{b} = \mathbf{g}_1^b, \mathbf{v}_1 = \mathbf{g}_1^{-\Delta_1'}, \mathbf{v}_2 = \mathbf{g}_1^{-\Delta_2'}\}$.
  It also outputs $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ as all three trapdoors τ, τ_1 and η.

  – The **one-time full simulator** otfS takes as input the trapdoors $\{c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'\}$ and a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ to produce a proof as follows: $T = \mathbf{x}_1^{d_1 + \iota e_1} \cdot \mathbf{x}_2^{d_2 + \iota e_2}$, $W = \mathbf{x}_1^{(\Delta_1' + \iota \Delta_2' + d_1 + \iota e_1)/b} \cdot \mathbf{x}_2^{(d_2 + \iota e_2 + c)/b}$, where $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$.

  – The **semi-functional verifier** sfV uses trapdoors $\{c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'\}$ to verify a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a proof T, W as follows: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$ and output 1 iff

$$(\mathbf{x}_1^{\Delta_1' + \iota \Delta_2'} \cdot \mathbf{x}_2^c \cdot T = W^b) \textbf{ and } (T = \mathbf{x}_1^{(d_1 + \iota e_1)} \cdot \mathbf{x}_2^{(d_2 + \iota e_2)}),$$

Theorem 1 *The above algorithms constitute a computationally-sound, complete and dual-system simulation sound quasi-adaptive NIZK proof system for the Diffie-Hellman language with language parameters sampled according to $\mathcal{D}$ above, given any bilinear-group generation algorithm for which the DDH assumption holds for groups $\mathbb{G}_1$. Let $\mathrm{ADV}_{\mathit{XDH}}$ denote any adversary's advantage in breaking the DDH assumption in group $\mathbb{G}_1$. Then, the probability of Adversary's success in the soundness experiment is at most $2 \cdot \mathrm{ADV}_{\mathit{XDH}} + O(1/q)$. Further, unbounded partial-simulation soundness is perfect. The advantage of adversary $\mathcal{A}_1$ in the partial-ZK property experiment is at most $2 \cdot N \cdot \mathrm{ADV}_{\mathit{XDH}} + O(N/q)$, where N is the number of $\mathcal{A}_1$'s oracle calls to the second oracle. Finally, $\mathcal{A}_4$'s advantage in the one-time full-ZK property experiment is at most $O((N+M)/q)$, where M and N are the number of calls of $\mathcal{A}_3$ and $\mathcal{A}_4$ combined to the first and second oracle resp.*

The theorem is proved in the next section.

3 Proof of DSS-QA-NIZK for DH Language

In this section we prove the theorem stated in the previous section about the security of the DSS-QA-NIZK construction for the Diffie-Hellman Language. After the proof, in the next sub-section, we show how the proof extends to the situation where the partial-world simulation key is revealed adaptively to the Adversary. Finally, we also show how to extend the above construction so that it is secure in a CCA1-security sense even after the simulation key is revealed.

Proof: [Theorem 1]

Completeness: Completeness of the real-world prover follows by simple inspection.

Soundness: We first define an extended Diffie-Hellman tag-based language

$$\overline{L}_\rho = \{\mathbf{x}_1, \mathbf{x}_2, T, \iota \mid \exists r \ : \mathbf{x}_1 = \mathbf{g}_2^r, \mathbf{x}_2 = \mathbf{a}^r, T = (\mathbf{d}\mathbf{e}^\iota)^r\}. \tag{1}$$

We will show that the element W is the single element proof for the quasi-adaptive NIZK for $\overline{L}_\rho$ [JR14]. For this extended language d and e are additional language parameters. Hence, from the soundness of this QA-NIZK [JR14] follows the soundness of the system in consideration. But to be self-contained, we provide a proof of soundness for the language $\overline{L}_\rho$ here, with the adversary getting the full CRS generated by K_1.

We will consider several experiments between a challenger $\mathcal{C}$ and adversary $\mathcal{A}$ here, starting with $\mathbf{H}_0$ which is same as the experiment of the soundness definition except that the challenger also generates ρ according to $\mathcal{D}$. The challenger $\mathcal{C}$ runs the verifier V on $\mathcal{A}$'s input $\mathbf{x}_1, \mathbf{x}_2, T$ and proof W and outputs the result of the verifier.

In $\mathbf{H}_1$, the challenger $\mathcal{C}$ generates the CRS differently by first picking Δ_1 and Δ_2 at random, and then setting $\Delta_1' = \Delta_1 \cdot b - d - c \cdot a$, and $\Delta_2' = \Delta_2 \cdot b - e$. It is easy to see that the CRS generated in $\mathbf{H}_1$ is identically distributed to that generated in $\mathbf{H}_0$. Further, instead of running the verifier V on $\mathcal{A}$'s input, the challenger runs the following test: it first picks s at random and then outputs:

$$e((\mathbf{v}_1\mathbf{v}_2^\iota)^s, \mathbf{x}_1^{-1}) \cdot e(\mathbf{c}^s, \mathbf{x}_2) \cdot e(\mathbf{g}_1^s, T) \cdot e(\mathbf{b}^s, W^{-1}).$$

Since probability of s being zero is only $1/q$, and further that $\mathbb{G}_1$ is a group, the probability of $\mathcal{C}$ outputting 1 in $\mathbf{H}_1$ is at most the probability of $\mathcal{C}$ outputting 1 in $\mathbf{H}_0$ (plus $1/q$).

In $\mathbf{H}_2$, the $\mathcal{C}$ also picks a random and independent s', and first computes a $\xi = e(\mathbf{g}_1^{s'}, T \cdot \mathbf{x}_2^c \cdot \mathbf{x}_1^{-(d+\iota e+a\cdot c)})$, and then outputs as follows:

$$e((\mathbf{v}_1\mathbf{v}_2^\iota)^s, \mathbf{x}_1^{-1}) \cdot e(\mathbf{c}^s, \mathbf{x}_2) \cdot e(\mathbf{g}_1^s, T) \cdot e(\mathbf{b}^s, W^{-1}) \cdot \xi.$$

The probability of $\mathcal{C}$ outputting 1 in $\mathbf{H}_2$ differs from it outputting 1 in $\mathbf{H}_1$ by at most the probability of an adversary's advantage in the DDH game. This follows by using DDH assumption on tuples $\mathbf{g}_1, \mathbf{g}_1^b, \mathbf{g}_1^{b\cdot s}$ and either $\mathbf{g}_1^s$ (real DDH challenge) or $\mathbf{g}_1^{s+s'}$ (fake DDH challenge). The important thing to note is that b is not used at all in simulating group $\mathbb{G}_2$ elements in the CRS (as opposed to $\mathbf{H}_0$). When the DDH challenge is real, this sets up $\mathbf{H}_1$ (i.e. $\mathcal{C}$ and the code of PPT $\mathcal{A}$), and when the DDH challenge is fake this sets up $\mathbf{H}_2$.

In $\mathbf{H}_3$, $\mathcal{C}$ also picks a random and independent c' and computes ξ as follows: $\xi = e(\mathbf{g}_1^{c'}, \mathbf{x}_2 \cdot \mathbf{x}_1^{-a}) \cdot e(\mathbf{g}_1^{s'}, T \cdot \mathbf{x}_1^{-(d+\iota e)})$.

Rest of the computation remains same. Again, the probability of $\mathcal{C}$ outputting 1 in $\mathbf{H}_3$ differs from it outputting 1 in $\mathbf{H}_2$ by at most the probability of an adversary's advantage in the DDH game. This follows by using DDH assumption on tuples $\mathbf{g}_1, \mathbf{g}_1^{s'}, \mathbf{g}_1^c$ and either $\mathbf{g}_1^{s'c}$ (real DDH challenge) or $\mathbf{g}_1^{c'}$ (fake DDH challenge). The important thing to note is that c is not used at all in simulating group $\mathbb{G}_2$ elements in the CRS (as opposed to $\mathbf{H}_0$). When the DDH challenge is real, this sets up $\mathbf{H}_2$ (i.e. $\mathcal{C}$ and the code of PPT $\mathcal{A}$), and when the DDH challenge is fake this sets up $\mathbf{H}_3$.

Now, a simple information theoretic argument shows that the probability of $\mathcal{C}$ outputting 1 in $\mathbf{H}_3$ when $\mathbf{x}_2 \cdot \mathbf{x}_1^{-a} \neq \mathbf{1}_1$ or $T \cdot \mathbf{x}_1^{-(d+\iota e)} \neq \mathbf{1}_1$ is negligible. That completes the proof of soundness.

Partial-ZK: We first note that the semi-functional CRS generator does a statistically indistinguishable simulation of the real-world CRS generator. This is seen by noting that the former generator generates ρ identically to $\mathcal{D}$, and that b is non-zero with high probability.

Now we show, by induction over the order of calls to the two oracles, that $\mathcal{A}_1$ cannot computationally distinguish between oracle responses in the real-world and the partial-simulation world. First, looking at oracle calls to the first oracle (i.e. P or sfS), we first note that if the call in the real-world to P is with language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a valid witness r, then since this property is polynomial time verifiable given language parameter ρ, by induction, the adversary $\mathcal{A}_1$ in the partial simulation world also generates a $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ which belongs to L_ρ (with probability close to 1). Next, note that when $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ belongs to L_ρ (i.e. $\mathbf{x}_2 = \mathbf{x}_1^a$), the proof simulation is perfect; the membership bit β is 1, and hence $y = 0$, which leads to identical distribution for proof T, W in the two worlds.

We next show that for each call to the second oracle (i.e. V or pV), the difference in the probability of the oracle returning 1 in the two worlds is negligible. Since the CRS generated in the two worlds has statistically indistinguishable distribution, the probability of V (using real-world CRS) returning 1 in the real-world is non-negligibly close to the probability of the same V (with partial-simulation world CRS) returning 1. Thus, we just need to bound the difference in the probability of V and pV returning 1 in the partial-simulation world.

Consider the language $\overline{L}_\rho$ defined above. Now, from definition of pV it is clear that it accepts iff $\mathbf{x}_1, \mathbf{x}_2, T, \iota$ are in $\overline{L}_\rho$, and W is the unique "proof" matching this tuple (i.e $W = (\mathbf{w}_1 \mathbf{w}_2^\iota)^r$). It is also clear that V accepts the triplet and W if the triplet is in $\overline{L}_\rho$ and W is this unique "proof". Further, it is clear that V rejects if the quadruplet is in $\overline{L}_\rho$ and W is not this unique proof. Thus, we just need to show that if the quadruplet is not in $\overline{L}_\rho$ then (regardless of W) V rejects with high probability. But, indeed we showed in the proof of soundness above that W is the (single group element) proof of the language $\overline{L}_\rho$ which is sound under the verifier V and with adversary getting the full CRS generated by K_1 (and hence also the full CRS generated by sfK_1).

Unbounded partial-simulation (relative) soundness: This follows trivially, since the private verifier has access to a (see the private verifier trapdoor), and directly checks if the potential language member submitted is indeed a language member.

One-time full-ZK: We will show that the one-time full-ZK property holds statistically, i.e. against all powerful adversaries. Now we define a sequence of experiments, and show that they are statistically indistinguishable.

First note that the distribution of the CRS generated by the one-time full-simulation CRS generator is statistically indistinguishable from the distribution of the CRS generated by the semi-functional CRS generator. We next show, by induction over the order of calls of $\mathcal{A}_3$ and $\mathcal{A}_4$ to the two oracles and the one-time generation of proof π, that $\mathcal{A}_4$ cannot distinguish between the responses of the oracle calls and π in the partial-simulation world and this one-time full simulation world.

To this end, we will define a sequence of hybrid experiments and show that the view of $\mathcal{A}_4$ in every two consecutive experiments is computationally indistinguishable.

The first experiment $\mathbf{H}_0$ is identical to the partial-simulation world, but we describe it in detail here. The challenger $\mathcal{C}$ interacts with two adversaries $\mathcal{A}_3$ and $\mathcal{A}_4$ each with two different types of oracle calls. To start with, $\mathcal{C}$ produces a CRS to be used by both the adversaries. $\mathcal{A}_3$ also produces a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$, and a membership bit β. The challenger then produces π and $\mathcal{A}_4$ takes this as additional input, and finally outputs a bit.

$\mathcal{C}$ starts by picking a at random and sets $\rho = \langle \mathbf{g}_2, \mathbf{g}_2^a \rangle$. Next, it picks $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ randomly and independently from $\mathbb{Z}_q$, and computes $d = d_1 + a \cdot d_2$, $e = e_1 + a \cdot e_2$, $\Delta_1 = (\Delta_1' + d + a \cdot c)/b$, $\Delta_2 = (\Delta_2' + e)/b$, and sets the CRS to be $(\sigma_p, \sigma_v, \mathcal{H})$:

$\sigma_p = \{\rho,\ \mathbf{d} = \mathbf{g}_2^d,\ \mathbf{e} = \mathbf{g}_2^e,\ \mathbf{w}_1 = \mathbf{g}_2^{\Delta_1},\ \mathbf{w}_2 = \mathbf{g}_2^{\Delta_2}\}$,
$\sigma_v = \{\mathbf{g}_1,\ \mathbf{c} = \mathbf{g}_1^c,\ \mathbf{b} = \mathbf{g}_1^b,\ \mathbf{v}_1 = \mathbf{g}_1^{-\Delta_1'},\ \mathbf{v}_2 = \mathbf{g}_1^{-\Delta_2'}\}$.
It retains $c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ as proof simulator trapdoors τ, and retains a, c, b, d, e, Δ_1, Δ_2 as private-verifier trapdoors η.

The challenger $\mathcal{C}$ serves all calls to the first oracle using the algorithm sfS described above on trapdoors τ. It also produces π using the same algorithm sfS on $\mathcal{A}_3$ supplied input. It serves all calls to the second oracle using private verifier pV described above using trapdoors η.

In experiment $\mathbf{H}_1$, the challenger $\mathcal{C}$, instead of picking d_1, d_2 at random (and similarly e_2, e_2 at random), picks d and d_2 randomly and independently and sets $d_1 = d - a \cdot d_2$ (and $e_1 = e - a \cdot e_2$). Rest of the CRS computation remains the same, and it retains a, c, b, d, d_2, e, e_2, Δ_1', Δ_2', Δ_1, Δ_2 as trapdoors (for both oracles). It further replaces sfS by the following: on input a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a binary bit β as follows: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$; if $\beta = 1$ set $y = 0$, else choose y at random from $\mathbb{Z}_q$; Compute and output

$$T = \mathbf{x}_1^{d+\iota e} \cdot \mathbf{g}_2^y\ ,\ W = \mathbf{x}_1^{(\Delta_1' + d + \iota \cdot (\Delta_2' + e))/b} \cdot \mathbf{x}_2^{c/b} \cdot \mathbf{g}_2^{y/b}. \tag{2}$$

We show that when the membership bit β is valid for $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$, the view of the adversaries in Expt_0 and Expt_1 is identical. First note that the CRS is identically distributed in the two experiments. Next, let $\mathbf{x}_2 = \mathbf{x}_1^a \cdot z$. Thus, if $\beta = 1$ then $z = \mathbf{1}_2$, otherwise z is not one. When $\beta = 1$, $\mathcal{C}$ chooses $y = 0$ in $\mathbf{H}_1$ above, and hence this is identical to behavior of sfS. Otherwise, y is chosen randomly in both sfS and $\mathbf{H}_1$, and now note that

$$\mathbf{x}_1^{d+\iota e} \cdot \mathbf{g}_2^y = \mathbf{x}_1^{d_1 + \iota e_1} \cdot (\mathbf{x}_2 / z)^{d_2 + \iota e_2} \cdot \mathbf{g}_2^y\ ,$$
$$\mathbf{x}_1^{(\Delta_1' + d + \iota \cdot (\Delta_2' + e))/b} \cdot \mathbf{x}_2^{c/b} \cdot \mathbf{g}_2^{y/b} = \mathbf{x}_1^{(\Delta_1' + d_1 + \iota \cdot (\Delta_2' + e_1))/b} \cdot \mathbf{x}_2^{c/b} \cdot (\mathbf{x}_2/z)^{(d_2 + \iota e_2)/b} \cdot \mathbf{g}_2^{y/b}.$$

Since, y is chosen randomly and independently the experiments $\mathbf{H}_0$ and $\mathbf{H}_1$ produce identical distributions.

In the next experiment $\mathbf{H}_2$, the challenger simulates otfS by producing the proof π on the potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and binary bit β produced by $\mathcal{A}_3$ as follows (all calls to the first oracle remain as in previous experiment): it ignores the bit β, computes $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$, and next computes and outputs

$$T = \mathbf{x}_1^{d + \iota e - a \cdot (d_2 + \iota e_2)} \cdot \mathbf{x}_2^{d_2 + \iota e_2},\ W = \mathbf{x}_1^{(\Delta_1' + d + \iota \cdot (\Delta_2' + e) - a \cdot (d_2 + \iota e_2))/b} \cdot \mathbf{x}_2^{(d_2 + \iota e_2 + c)/b}. \tag{3}$$

Again, let $\mathbf{x}_2 = \mathbf{x}_1^a \cdot z$, and note the T and W being computed in this experiment can be written as

$$T = \mathbf{x}_1^{d + \iota e} \cdot z^{d_2 + \iota e_2}\ ,\ W = \mathbf{x}_1^{(\Delta_1' + d + \iota \cdot (\Delta_2' + e))/b} \cdot \mathbf{x}_2^{c/b} \cdot z^{(d_2 + \iota e_2)/b}.$$

Next, note that d_2 was chosen randomly and independently of d. Further, the CRS computation, all first oracle response computations, and all second oracle response (see pV) computations only need d and e (and not d_2). Thus, since $z = \mathbf{1}_2$ iff $\beta = 1$ (given that β is a valid membership bit), the distribution of π is identical in $\mathbf{H}_1$ and $\mathbf{H}_2$.

In the next experiment $\mathbf{H}_3$, the private-verifier pV is replaced by the following code: output 1 iff

$$T\ =\ \mathbf{x}_1^{d + \iota e - a \cdot (d_2 + \iota e_2)} \cdot \mathbf{x}_2^{(d_2 + \iota e_2)}\ \textbf{and}\ \ \mathbf{x}_1^{\Delta_1' + \iota \Delta_2'} \cdot \mathbf{x}_2^c \cdot T\ =\ W^b, \tag{4}$$

where $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$.

In order to show that the view of the adversary is indistinguishable in experiments $\mathbf{H}_3$ and $\mathbf{H}_2$, we define several hybrid experiments $\mathbf{H}_{2,i}$ (for $0 \leq i \leq N$, where N is the total number of calls to the second-oracle by $\mathcal{A}_3$ and $\mathcal{A}_4$ combined). Experiment $\mathbf{H}_{2,0}$ is identical to $\mathbf{H}_2$, and the intermediate experiments are defined inductively, by modifying the response of one additional second-oracle call starting with the last (N-th) second-oracle call, and ending with the changed response of first second-oracle call. The last hybrid experiment $\mathbf{H}_{2,N}$ will then be same as $\mathbf{H}_3$. The second-oracle call response in experiment $\mathbf{H}_{2,i+1}$ differs only in the $(N-i)$-th second-oracle call response in $\mathbf{H}_{2,i}$. In the latter experiment, this call is still served as in $\mathbf{H}_2$ (i.e. using pV). In the former experiment $\mathbf{H}_{2,i+1}$, the $(N-i)$-th call is responded to as defined in $\mathbf{H}_3$ above.

To show that the view of the adversary is statistically indistinguishable in $\mathbf{H}_{2,i}$ and $\mathbf{H}_{2,i+1}$, first note that the view of the adversary ($\mathcal{A}_3$ and A_4 combined) till it's $(N-i)$-th call in both experiments is identical. Moreover, this view is independent of d_2 and e_2 *except* for the information $d_2+\iota' e_2$, where $\iota' = \mathcal{H}(\mathbf{x}'_1, \mathbf{x}'_2)$, and $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$ is the potential language member on which $\mathcal{A}_3$ sought a proof π (if the $(N-i)$-th second-oracle call happens before $\mathcal{A}_3$ outputs this request, then even this information $d_2 + \iota' e_2$ is not available to the adversary). Now, consider the case that the adversary does not query this $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$ and π to the second-oracle (and hence to this $(N-i)$-th call).

First, suppose the $(N-i)$-th call has the same $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$, but a different proof π' then the proof π output by the challenger $\mathcal{C}$ (see definition of experiment $\mathbf{H}_2$ above). Then, we show that both experiments $\mathbf{H}_{2,i}$ and $\mathbf{H}_{2,i+1}$ reject this call (i.e. output 0). This follows from uniqueness of the valid proof, but to be rigorous, note that the proof π produced in both experiments is

$$T = (\mathbf{x}'_1)^{d+\iota' e - a\cdot(d_2+\iota' e_2)} \cdot (\mathbf{x}'_2)^{d_2+\iota' e_2},\ W = (\mathbf{x}'_1)^{(\Delta'_1+d+\iota'\cdot(\Delta'_2+e)-a\cdot(d_2+\iota' e_2))/b} \cdot (\mathbf{x}'_2)^{(d_2+\iota' e_2+c)/b},$$

where $\iota' = \mathcal{H}(\mathbf{x}'_1, \mathbf{x}'_2)$. Since, the $(N-i)$-th call has the same $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$, and hence the same hash $\mathcal{H}$-value, it is easy to inspect that pV accepts the above proof π, and in particular $T = (\mathbf{x}'_1)^{d+\iota' e}$ and $W = (\mathbf{x}'_1)^{\Delta_1+\iota'\Delta_2}$. Thus, any change in this T or W will be rejected by pV (and hence experiment $\mathbf{H}_{2,i}$). It is also easy to inspect that the response to $(N-i)$-th call in $\mathbf{H}_{2,i+1}$ with input $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$ and above proof π will be 1, i.e.

$$T \;=\; (\mathbf{x}'_1)^{d+\iota' e - a\cdot(d_2+\iota' e_2)} \cdot (\mathbf{x}'_2)^{(d_2+\iota' e_2)} \;\textbf{and}\;\; (\mathbf{x}'_1)^{\Delta'_1+\iota'\Delta'_2} \cdot (\mathbf{x}'_2)^{c} \cdot T \;=\; W^b,$$

Thus, any change in T will fail the first conjunct, and any change in just W (and not T) will fail the second conjunct, except for the negligible probability that $b = 0$. Thus, experiment $\mathbf{H}_{2,i+1}$ also outputs 0 on its $(N-i)$-th call.

Now, suppose that $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ in the $(N-i)$-th second-oracle call is different from $\langle \mathbf{x}'_1, \mathbf{x}'_2 \rangle$ on which the adversary sought proof π. Since $\mathcal{H}$ is a collision resistant hash function, no efficient adversary can choose these inputs to $\mathcal{H}$ such that their hash is the same. So, assume that $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$ is different from $\iota' = \mathcal{H}(\mathbf{x}'_1, \mathbf{x}'_2)$.

Thus, $d_2 + \iota e_2$ is random and independent given $d_2 + \iota' e_2$ (recall, d_2, e_2 are chosen randomly and independently of d and e from $\mathbf{H}_1$ on-wards). Now it is clear that if pV accepts $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and the submitted proof T, W, then experiment $\mathbf{H}_{2.i+1}$ also accepts the same. So, we next show that if $\mathbf{H}_{2,i+1}$ accepts the $(N-i)$-th call, then pV also accepts. Thus,

$$T \;=\; \mathbf{x}_1^{d+\iota e - a\cdot(d_2+\iota e_2)} \cdot \mathbf{x}_2^{(d_2+\iota e_2)} \;\textbf{and}\;\; \mathbf{x}_1^{\Delta'_1+\iota\Delta'_2} \cdot \mathbf{x}_2^{c} \cdot T \;=\; W^b \tag{5}$$

holds, and suppose pV rejects, i.e. one of the three conjuncts $\mathbf{x}_2 = \mathbf{x}_1^a$ and $T = \mathbf{x}_1^{d+\iota e}$ and $W = \mathbf{x}_1^{\Delta_1+\iota\Delta_2}$ fails. First, note that if the first two conjuncts hold and equation (5) holds, then the third

conjunct (involving W) holds. Also, note that if the first conjunct holds and equation (5) holds, then the second conjunct (involving T) holds. So, suppose the first conjunct does not hold, i.e. $\mathbf{x}_2 \neq \mathbf{x}_1^a$. Now, from equation 5, we get

$$d_2 + \iota e_2 = \frac{\log_{\mathbf{g}_2}(T/\mathbf{x}_1^{d+\iota e})}{\log_{\mathbf{g}_2}(\mathbf{x}_2/\mathbf{x}_1^a)} \tag{6}$$

But since $d_2 + \iota e_2$ is completely independent of the adversary's view till the $(N-i)$-th call and d, e, a, and hence the right-hand-side of equation (6), the probability of equation (6) holding is at most $1/q$. that completes the proof of computational indistinguishability of $\mathbf{H}_{2.i}$ and $\mathbf{H}_{2,i+1}$, and hence by induction of also $\mathbf{H}_2$ and $\mathbf{H}_3$. Note that computational restriction was only required for the collision-resistance property.

In then next experiment $\mathbf{H}_4$, the challenger $\mathcal{C}$ changes back the response to the first-oracle to as in sfS (from (9) in $\mathbf{H}_1$), i.e.: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$; if $\beta = 1$ set $y = 0$, else choose y at random from $\mathbb{Z}_q$; Compute and output

$$T = \mathbf{x}_1^{d_1+\iota e_1} \cdot \mathbf{x}_2^{d_2+\iota e_2} \cdot \mathbf{g}_2^{y} \ , \ W = \mathbf{x}_1^{(\Delta_1'+d_1+\iota\cdot(\Delta_2'+e_1))/b} \cdot \mathbf{x}_2^{(d_2+\iota e_2+c)/b} \cdot \mathbf{g}_2^{y/b}.$$

The argument that the view of the adversary in $\mathbf{H}_4$ is identical to the view of the adversary in $\mathbf{H}_3$ (when the membership bits β are valid) is same as the argument for $\mathbf{H}_1$ and $\mathbf{H}_0$ given earlier.

In the next experiment $\mathbf{H}_5$, the challenger $\mathcal{C}$ changes the response to the second-oracle (verifier) call (from equation (4) defined in $\mathbf{H}_3$) to: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$ and output 1 iff

$$(T \ = \ \mathbf{x}_1^{(d_1+\iota e_1)} \cdot \mathbf{x}_2^{(d_2+\iota e_2)}) \ \textbf{and} \ \ (\mathbf{x}_1^{\Delta_1'+\iota\Delta_2'} \cdot \mathbf{x}_2^{c} \cdot T \ = \ W^b),$$

It is straightforward to see that Adversary's view is identical in $\mathbf{H}_4$ and $\mathbf{H}_5$ (i.e. by the way d_1 and e_1 are defined).

In the next experiment $\mathbf{H}_6$, the challenger samples ρ, and then computes the CRS and the trapdoors exactly as in otfK_1. It is straightforward to see that the CRS (and the trapdoors) generated in $\mathbf{H}_6$ is identically distributed to the CRS (and the trapdoors) generated in $\mathbf{H}_5$.

At this point, the view of the adversaries $\mathcal{A}_3$ and $\mathcal{A}_4$ is exactly as in the one-time full ZK world, and that completes the proof. □

3.1 Revealing the Partial-Simulation Key

In some applications, the partial-simulation trapdoor τ can be revealed to the adversary, adaptively on adversary's demand. This event will be referred to as the *Reveal* event. In this case we would like to have the property that one-time full ZK simulation is still possible under the restriction that the adversary $\mathcal{A}_4$ has access to the second oracle (i.e. the verifier pV/sfV) only *till the reveal event*. The adversary $\mathcal{A}_3$ will continue to have access to the second oracle regardless of the reveal event.

To achieve this, we highlight an interesting property of the DSS-QA-NIZK of the Diffie-Hellman language from Section 2. We show that in the one-time full ZK property, there is an *intermediate* world, which uses a slightly different semi-functional CRS generator and simulator, but uses the private-verifier pV. Let's call these CRS generator and simulator sfK_1' and sfS' respectively, which

we define soon. Then, we have the following:

$$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); (\rho,\sigma,\tau,\eta) \leftarrow \mathsf{sfK}_1(\lambda);\ (x,\beta,s) \leftarrow \mathcal{A}_3^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot),\,\mathsf{pV}(\sigma,\eta,\cdot,\cdot)}(\sigma);$$
$$\pi \leftarrow \mathsf{sfS}(\sigma,\tau,x,\beta) : \mathcal{A}_4^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot),\,\mathsf{pV}(\sigma,\eta,\cdot,\cdot)}(\pi,s) = 1] \approx$$
$$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); (\rho,\sigma,\tau,\tau_1,\eta) \leftarrow \mathsf{sfK}'_1(\lambda);\ (x,\beta,s) \leftarrow \mathcal{A}_3^{\mathsf{sfS}'(\sigma,\tau,\cdot,\cdot),\,\mathsf{pV}(\sigma,\eta,\cdot,\cdot)}(\sigma);$$
$$\pi \leftarrow \mathsf{otfS}(\sigma,\tau_1,x) : \mathcal{A}_4^{\mathsf{sfS}'(\sigma,\tau,\cdot,\cdot),\,\mathsf{pV}(\sigma,\eta,\cdot,\cdot)}(\pi,s) = 1] \approx \quad (7)$$
$$\Pr[\lambda \leftarrow \mathsf{K}_0(1^m); \rho \leftarrow \mathcal{D}_\lambda;\ (\sigma,\tau,\tau_1,\eta) \leftarrow \mathsf{otfK}_1(\lambda,\rho);\ \ (x,\beta,s) \leftarrow \mathcal{A}_3^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot),\,\mathsf{sfV}(\sigma,\eta,\cdot,\cdot)}(\sigma);$$
$$\pi \leftarrow \mathsf{otfS}(\sigma,\tau_1,x) : \mathcal{A}_4^{\mathsf{sfS}(\sigma,\tau,\cdot,\cdot),\,\mathsf{sfV}(\sigma,\eta,\cdot,\cdot)}(\pi,s) = 1], \quad (8)$$

where β is a correct L_ρ-membership bit for x, **and** all calls to sfS also have correct L_ρ-membership bits **and** $\langle x,\pi\rangle$ is not queried to sfV/pV.

We now define sfK'_1 and sfS'. These actually come from the intermediate games in the proof of one-time full-ZK in Section!3. In particular, this intermediate world is just experiment $\mathbf{H}_2$. So, in particular sfK'_1 is the following algorithm: It is just like sfK_1 except that instead of picking d_1, d_2 at random (and similarly e_2, e_2 at random), it picks d and d_2 randomly and independently and sets $d_1 = d - a \cdot d_2$ (and $e_1 = e - a \cdot e_2$). Rest of the CRS computation remains the same. It outputs $c, b, d, e, , \Delta'_1, \Delta'_2$ as trapdoors τ (to be used by sfS'), and outputs $c, b, d_1, d_2, e_1, e_2, \Delta'_1, \Delta'_2$ as trapdoors τ_1 (to be used by otfS), and outputs a, c, b, d, e, Δ_1, Δ_2 as private-verifier trapdoors η.

The intermediate semi-functional simulator sfS' (using trapdoors $c, b, d, e, , \Delta'_1, \Delta'_2$) on input a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2\rangle$ and a binary bit β as follows: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$; if $\beta = 1$ set $y = 0$, else choose y at random from $\mathbb{Z}_q$; Compute and output

$$T = \mathbf{x}_1^{d+\iota e} \cdot \mathbf{g}_2^{y}\ ,\ W = \mathbf{x}_1^{(\Delta'_1 + d + \iota\cdot(\Delta'_2+e))/b} \cdot \mathbf{x}_2^{c/b} \cdot \mathbf{g}_2^{y/b}. \quad (9)$$

The proof of the above property ((7) and (8) then follows from the proof of Section 3.

We now show that the above property ((7) and (8) holds even if τ is revealed to the adversary (adaptively), as long as the following holds: (a) the second oracle (verifier) is not called after τ is revealed to the adversary, and (b) the switch between the partial-simulation world and the intermediate world above is undertaken only if x is a (Diffie-Hellman) language member (i.e. $\beta = 1$), and (c) the first oracle ($\mathsf{sfS}/\mathsf{sfS}'$) is only called with language members (i.e. $\beta = 1$) in all cases above.

To show this we need to verify the (computational or statistical) indistinguishability of various experiments in the proof of one-time full-ZK, i.e. games $\mathbf{H}_0$ through $\mathbf{H}_6$.

Between experiment $\mathbf{H}_0$ and $\mathbf{H}_1$ the CRS computation is statistically indistinguishable even with τ revealed. When $\beta = 1$ (which is the restriction (b) and (c) combined above), the generation of T and W is identical in the two games, i.e. sfS and sfS' behave identically.

In the indistinguishability of experiments $\mathbf{H}_1$ and $\mathbf{H}_2$, again restricting to $\beta = 1$ (by (b)), we notice there is no difference in the way sfS generates T, W (experiment $\mathbf{H}_1$) and how otfS (experiment $\mathbf{H}_2$) generates T, W.

In the indistinguishability of $\mathbf{H}_2$ and $\mathbf{H}_3$ where we replace pV with sfV, we go though various hybrid games $\mathbf{H}_{2,i}$. Since by restriction (a) above, no calls to the second oracle are allowed after τ, i.e. d_2, e_2 among others, is revealed to the adversary, the information-theoretic argument in equation 6 still holds.

The indistinguishability of $\mathbf{H}_3$ and $\mathbf{H}_4$ follows by noting that we restrict to $\beta = 1$ (i.e. restriction (c)). Indistinguishability of $\mathbf{H}_4, \mathbf{H}_5$ and $\mathbf{H}_6$ is straightforward.

We will see that the above properties of the DSS-QA-NIZK for the Diffie-Hellman language allows for a *keyed-homomorphic CCA2-secure encryption* scheme with CPA security maintained if the homomorphic-evaluation key is revealed (Section 4). The above properties ensure that CPA security holds along with a decryption oracle till reveal event.

3.2 Key-Reveal and CCA1-style Security

We can also extend th keyed-homomorphic scheme of Section 4 to maintain CCA1-security when the key is revealed, but the construction requires use of another group element as proof that provides CCA1-style security. Since this proof element is actually malleable it does not require additional trapdoor keys. This additional group element however provides CCA1 style-soundness of the Diffie-Hellman tuple, and hence the usual trick of switching between alternate ways of decryption can be accomplished up to the encryption query event.

To this end, we define a DSS-QA-NIZK for a new class of parameterized languages.

We consider the following class of languages parameterized by $\mathbf{g}_2$, $\mathbf{g}_2^a$, $\mathbf{g}_2^{\epsilon_1} \cdot \mathbf{g}_2^{\epsilon_2 \cdot a}$, ϵ_1, ϵ_2 ($\in \mathbb{G}_2^4 \times \mathbb{Z}_q^2$). The **augmented Diffie-Hellman language** corresponding to one such parameter ρ is

$$L_\rho \;=\; \{\langle \mathbf{g}_2^r, \mathbf{g}_2^{a \cdot r}, (\mathbf{g}_2^{\epsilon_1} \cdot \mathbf{g}_2^{\epsilon_2 \cdot a})^r \rangle \mid r \in \mathbb{Z}_q\}.$$

Note that the parameters *include* ϵ_1 and ϵ_2. In this respect, the hard component of the language (Diffie-Hellman) is due to a and that is not part of the parameters. The distribution $\mathcal{D}$ under which the DSS-QA-NIZK now picks ϵ_1 and ϵ_2 at random from $\mathbb{Z}_q$ as well.

We now describe the modifications required to the construction of Section 2. First, the component T of the proof will remain as before. The component W was just the QA-NIZK for the extended Diffie-Hellman language (see (1) in Section 3), and now it will be the single group element QA-NIZK [JR14] for the augmented and extended Diffie-Hellman language (and for this QA-NIZK the parameters ϵ_1, ϵ_2 are ignored by the CRS generator, and it just uses $\mathbf{g}_2$, $\mathbf{g}_2^a$, $\mathbf{g}_2 \epsilon_1 \cdot \mathbf{g}_2^{\epsilon_2 \cdot a}$). More precisely, this *doubly-augmented Diffie-Hellman* language is the following tag-based language

$$\overline{XL}_\rho = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, T, \iota \mid \exists r \; : \mathbf{x}_1 = \mathbf{g}_2^r, \mathbf{x}_2 = \mathbf{a}^r, \mathbf{x}_3 = (\mathbf{g}_2^{\epsilon_1} \cdot \mathbf{g}_2^{\epsilon_2 \cdot a})^r, T = (\mathbf{d}\mathbf{e}^\iota)^r\}. \tag{10}$$

Thus, the real-world CRS for the DSS-QA-NIZK is the real-world CRS for the QA-NIZK for this doubly-augmented Diffie-Hellman language $\overline{XL}_\rho$ (with ϵ_1, ϵ_2 dropped) (see [JR14]). The real-world prover P computes $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, l)$, and $T = (\mathbf{d} \cdot \mathbf{e}^\iota)^r$, and computes W using the prover for the QA-NIZK for $\overline{XL}_\rho$. The real-world verifier V just follows the verifier for the QA-NIZK for $\overline{XL}_\rho$.

In the partial-simulation world, sfK_1 itself generates the language parameters. However, it generates $\mathbf{g}_2^{\epsilon_1} \cdot \mathbf{g}_2^{\epsilon_2 \cdot a}$ differently by just picking an ϵ at random from $\mathbb{Z}_q$ and outputting $\mathbf{g}_2^\epsilon$. The value ϵ is made an additional part of the verifier trapdoor η, and hence the private verifier pV can directly check for validity of the language members. Crucially, ϵ is not part of the proof simulation trapdoor τ.

In the one-time full-ZK world, the CRS generator otfK_1 will make ϵ_1 and ϵ_2 as part of the verifier trapdoor η. Recall ϵ_1 and ϵ_2 are part of the parameters of the augmented Diffie-Hellman language.

Now, focusing on the proof security of the above scheme, the proof is similar to the proof of theorem 1. More importantly, now we also get a CCA1-style security even if the proof simulation

trapdoor is revealed. This is proven similarly to the proof sketch given above in Section 3.1, but now with the restriction of not calling the second (verifier) oracle after reveal *only limited* to $\mathcal{A}_4$. More precisely, we now claim that properties ((7) and (8) hold even if τ is revealed to the adversary (adaptively), as long as the following (modified restrictions) holds: (a') the second oracle (verifier) is not called by $\mathcal{A}_4$ after τ is revealed to the adversary, and (b) the switch between the partial-simulation world and the intermediate world above is undertaken only if x is a (Diffie-Hellman) language member (i.e. $\beta = 1$), and (c) the first oracle ($\mathsf{sfS}/\mathsf{sfS}'$) is only called with language members (i.e. $\beta = 1$) in all cases above.

The proof is similar to the proof sketch above in Section 3.1 except with the following change. In the indistinguishability of $\mathbf{H}_2$ and $\mathbf{H}_3$ where we replace pV with sfV, we go though various hybrid games $\mathbf{H}_{2,i}$. Since by restriction (a') above, no calls to the second oracle are allowed after τ, i.e. d_2, e_2 among others, is revealed to the adversary, the information-theoretic argument in equation 6 still holds. A similar information-theoretic argument continues to hold for $\mathcal{A}_3$'s oracle calls using β_2 as trapdoor to prove that if $\mathbf{x}_2 \neq \mathbf{x}_1^a$ holds then

$$\beta_2 = \frac{\log_{\mathbf{g}_2}(\mathbf{x}_3/\mathbf{x}_1^{\beta})}{\log_{\mathbf{g}_2}(\mathbf{x}_2/\mathbf{x}_1^{a})} \tag{11}$$

where β is chosen independently by challenger $\mathcal{C}$ in experiment $\mathbf{H}_2$ and $\mathbf{H}_3$, But, the probability of this holding is negligible, and that completes the proof sketch.

4 Keyed-Homomorphic CCA Encryption

Keyed-Homomorphic Encryption is a primitive, first developed in [EHO+13], which allows homomorphic operations with a restricted evaluation key, while preserving different flavors of semantic security, depending on whether access to the evaluation key is provided or not. To an adversary not having access to the evaluation key, the homomorphic operation should not be possible and this is ensured by requiring CCA security. However, if an adversary comes into possession of the evaluation key, CCA security can no longer be preserved and thus weaker forms of security, such as CCA1, are required. In [LPJY14], the authors gave improved constructions with better security guarantees.

Definition 2 (KH-PKE Scheme) *A KH-PKE scheme for a message space $\mathcal{M}$ with a binary operation $\odot$ defined from $\mathcal{M}^2 \to \mathcal{M}$ is a tuple of algorithms $(KeyGen, Enc, Dec, Eval)$:*

KeyGen: *This algorithm takes 1^λ as input, and returns a public key pk, a decryption key sk_d, and a homomorphic operation key sk_h.*

Enc: *This algorithm takes pk and a message $M \in \mathcal{M}$ as inputs and returns a ciphertext C.*

Dec: *This algorithm takes sk_d and a ciphertext C as input, and returns M or $\perp$.*

Eval: *This algorithm takes sk_h and two ciphertexts C_1 and C_2 as inputs, and returns a ciphertext C or $\perp$.*

The scheme is said to be correct if (i) for Enc we have $Dec(sk_d, C) = M$ and (ii) for Eval we have $Dec(sk_d, C) = Dec(sk_d, C_1) \odot Dec(sk_d, C_2)$.

Definition 3 (KH-CCA Security) *A KH-PKE scheme $(KeyGen, Enc, Dec, Eval)$ for a message space $\mathcal{M}$ with a binary operation $\odot$ defined from $\mathcal{M}^2 \to \mathcal{M}$ is said to be KH-CCA secure if no PPT adversary $\mathcal{A}$ has non-negligible advantage in winning the following game with a challenger:*

KeyGen: *The challenger takes 1^λ as input, and computes a public key pk, a decryption key sk_d, and a homomorphic operation key sk_h. It then gives pk to $\mathcal{A}$. It also initializes a list $D := \emptyset$. It also gives oracle access to $\mathcal{A}$ to the functions $Enc(.), Eval(sk_h, .), RevHK$ and $Dec(sk_d, .)$ defined as follows:*

Enc: *This query is performed once. On $\mathcal{A}$'s inputs $m_0, m_1 \in \mathcal{M}$, the challenger randomly samples a bit b and computes $C := Enc(pk, m_b)$ and returns C.*

Eval: *On receiving ciphertexts C_1, C_2 from $\mathcal{A}$, computes $C := Eval(sk_h, C_1, C_2)$ and returns C. In addition if $C_1 \in D$ or $C_2 \in D$, then sets $D := D \cup \{C\}$.*

RevHK: *Upon receiving this request, returns sk_h.*

Dec: *This oracle is not available after $\mathcal{A}$ has both requested $RevHK$ and obtained the challenge ciphertext C^* in any order. When available, it returns $\perp$ if $C \in D$ and $Dec(sk_d, C)$ on $\mathcal{A}$'s query C.*

The adversary $\mathcal{A}$ outputs a bit b' and wins if $b' = b$. Its advantage is defined to be $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$.

Construction. We present a construction of a KH-CCA secure encryption scheme which utilizes our DSS-QA-NIZK proofs. We start with the observation that a standard El Gamal encryption scheme $(\mathbf{g}^x, m \cdot \mathbf{f}^x)$ is multiplicatively homomorphic, but is not CCA secure due to the exact same reason. Extending it to be a CCA secure encryption scheme, as in Cramer-Shoup [CS02], destroys the homomorphic property. Our construction essentially takes an El Gamal scheme and attaches a DSS-QA-NIZK to it. Although the NIZK itself is not homomorphic, we can take advantage of the corresponding Semi-Functional Simulator sfS and simulate the proof of a multiplicatively generated El Gamal encryption while computing a homomorphic evaluation. We show that keeping the sfS trapdoor hidden from the adversary enables the scheme to be CCA secure, while also giving weaker semantic security if the sfS trapdoor is revealed. The semantic security continues to hold even with the availability of a decryption oracle till the time of reveal. Using the augmented Diffie Hellman language and its DSS-QA-NIZK as described in Section 3.2, we also get CCA1-security despite the key being revealed. Here we focus on the construction for the weaker notion where the decryption oracle is available till the reveal event.

KeyGen: Generate $\mathbf{g}$ and a, k randomly. Use sfK_1 to generate CRS σ and trapdoors τ and η for the language $(\mathbf{g}^w, \mathbf{g}^{aw})$. Set $pk = (\mathbf{g}, \mathbf{g}^a, \mathbf{g}^k, \sigma), \quad sk_h := \tau, \quad sk_d := k.$

Enc: Given plaintext m, generate $w \xleftarrow{\$} \mathbb{Z}_q$ and compute

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m \cdot \mathbf{g}^{kw}, \mathsf{P}(\sigma, (\mathbf{g}^w, \mathbf{g}^{aw}), w; \gamma)),$$

where $\gamma := m \cdot \mathbf{g}^{kw}$

Dec: Given ciphertext $c = (\rho, \hat{\rho}, \gamma, \pi)$, first check if $\mathsf{V}(\sigma, \pi, (\rho, \hat{\rho}); \gamma)$ holds, then compute $m := \gamma / \rho^k$.

Eval: Given ciphertexts $c_1 = (\rho_1, \hat{\rho}_1, \gamma_1, \pi_1)$ and $c_2 = (\rho_2, \hat{\rho}_2, \gamma_2, \pi_2)$, first check if $\mathsf{V}(\sigma, \pi, (\rho_1, \hat{\rho}_1); \gamma_1)$ and $\mathsf{V}(\sigma, \eta, \pi, (\rho_2, \hat{\rho}_2); \gamma_2)$ hold. Then compute: $\rho = \rho_1\rho_2, \hat{\rho} = \hat{\rho}_1\hat{\rho}_2, \gamma = \gamma_1\gamma_2$. Then compute $\pi := \mathsf{sfS}(\sigma, \tau, (\rho, \hat{\rho}), \beta = 1; \gamma)$. Output ciphertext $c := (\rho, \hat{\rho}, \gamma, \pi)$.

Theorem 4 (Security of Construction) *The above algorithms $(KeyGen, Enc, Dec, Eval)$ constitute a KH-CCA secure Keyed-Homomorphic Public Key Encryption scheme.*

Proof: We will prove the theorem using theorem 1 and its extension to the reveal event in Section 3.1. We prove the theorem by going through a sequence of games, where for any $m_0, m_1 \in \mathcal{M}$, the first game is a world where the challenger encrypts m_0 and the last game is a world where the challenger encrypts m_1.

Game 0:

KeyGen: Generate $\mathbf{g}$ and a, k randomly. Use sfK_1 to generate CRS σ and trapdoors τ and η for the language $(\mathbf{g}^w, \mathbf{g}^{aw})$. Set $pk = (\mathbf{g}, \mathbf{g}^a, \mathbf{g}^k, \sigma), \quad sk_h := \tau, \quad sk_d := k$.

Enc: Generate $w \xleftarrow{\$} \mathbb{Z}_q$ and compute

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_0 \cdot \mathbf{g}^{kw}, \mathsf{P}(\sigma, (\mathbf{g}^w, \mathbf{g}^{aw}), w; \gamma)),$$

where $\gamma := m_0 \cdot \mathbf{g}^{kw}$

Dec: Given ciphertext $c = (\rho, \hat{\rho}, \gamma, \pi)$, first check if $\mathsf{V}(\sigma, \pi, (\rho, \hat{\rho}); \gamma)$ holds, then compute $m := \gamma/\rho^k$.

Eval: Given ciphertexts $c_1 = (\rho_1, \hat{\rho}_1, \gamma_1, \pi_1)$ and $c_2 = (\rho_2, \hat{\rho}_2, \gamma_2, \pi_2)$, first check if $\mathsf{V}(\sigma, \pi, (\rho_1, \hat{\rho}_1); \gamma_1)$ and $\mathsf{V}(\sigma, \eta, \pi, (\rho_2, \hat{\rho}_2); \gamma_2)$ hold. Then compute: $\rho = \rho_1\rho_2, \hat{\rho} = \hat{\rho}_1\hat{\rho}_2, \gamma = \gamma_1\gamma_2$. Then compute $\pi := \mathsf{sfS}(\sigma, \tau, (\rho, \hat{\rho}); \gamma)$. Output ciphertext $c := (\rho, \hat{\rho}, \gamma, \pi)$.

RevHK: Output τ.

Game 1: Same as Game 0, but now during decryption and eval the challenger switches to the private verifier pV. Also the challenge encryption is performed as

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_0 \cdot \mathbf{g}^{kw}, \mathsf{sfS}(\sigma, \tau, (\mathbf{g}^w, \mathbf{g}^{aw}), \beta = 1; \gamma)),$$

where $\gamma := m_0 \cdot \mathbf{g}^{kw}$.

Indistinguishability follows due to the partial-ZK property.

Game 2: Same as Game 1, but now KeyGen generates k_0, k_1 randomly and sets $k = k_0 + ak_1$ and gives $sk_d = (k_0, k_1)$. The challenge encryption is performed as

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw}, \mathsf{sfS}(\sigma, \tau, (\mathbf{g}^w, \mathbf{g}^{aw}), \beta = 1; \gamma)),$$

where $\gamma := m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw}$. Decryption is performed as $m := \gamma/(\rho^{k_0}\hat{\rho}^{k_1})$.

This step is indistinguishable from Game 1, as unbounded partial-simulation (relative) soundness holds in the semi-functional world, therefore $\hat{\rho} = \rho^a$.

Game 3: Same as Game 2, but now KeyGen switches to the otfK_1 to generate CRS σ and trapdoors $\sigma, \tau, \tau_1, \eta$. Now sfS in Eval continues to use σ, τ and the pV is switched to sfV which uses σ, η. The Challenge encryption is performed as

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw}, \mathsf{otfS}(\sigma, \tau_1, (\mathbf{g}^w, \mathbf{g}^{aw}); \gamma)),$$

where $\gamma := m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw}$.

Indistinguishability holds due to the one-time full-ZK property. Note that the membership bits β in the calls to sfS (in Eval) in game 2 are with $\beta = 1$, and they are correct memebership bits as the verifier pV (note the swtich in game 1) is unbounded partial-simulation sound. Since, the one-time full-ZK property is proven against all powerful adversaries (see Section 3, it suffices to employ the one-time full-ZK property when the membership bits are known to be correct only in the partial-simulation world.

Game 4: Same as Game 3, but now the challenge encryption is performed as follows: Generate $w, w' \xleftarrow{\$} \mathbb{Z}_q$ and compute:

$$c := (\mathbf{g}^w, \mathbf{g}^{aw'}, m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw'}, \mathsf{otfS}(\sigma, \tau_1, (\mathbf{g}^w, \mathbf{g}^{aw'}); \gamma)),$$

where $\gamma := m_0 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw'}$.

Indistinguishability holds by DDH.

Game 5: Now we switch to the intermediate world (see Section 3.1), where the KeyGen uses sfK'_1 to generate $\sigma, \tau, \tau_1, \eta$. The Eval function now uses sfS' to simulate proofs. Decryption is performed as follows: First switch to pV, while checking the NIZK in the ciphertext and then compute $m := \gamma/\rho^{k_0 + ak_1}$ with decryption key $sk_d = k = k_0 + ak_1$.

This step is indistinguishable from Game 4 as we could switch to pV for decryption, as in the intermediate world, and as simulation soundness holds with pV as the verifier, therefore $\hat{\rho} = \rho^a$.

Game 6: Now we choose a, k'_0, k'_1, w, w' such that $\log(m_0) + k_0 w + k_1 aw' = \log(m_1) + k'_0 w + k'_1 aw'$ and $k_0 + ak_1 = k'_0 + ak'_1$. It can be seen that the distribution of a, k'_0, k'_1, w, w' is identical as a, k_0, k_1, w, w' in Game 5. Hence these two games are indistinguishable.

Now the challenge encryption is performed as

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw'}, \mathsf{otfS}(\sigma, \tau, (\mathbf{g}^w, \mathbf{g}^{aw'}); \gamma)),$$

where $\gamma := m_1 \cdot \mathbf{g}^{k_0 w}\mathbf{g}^{k_1 aw'}$. Decryption is performed as $m := \gamma/\rho^{k'_0 + ak'_1}$ with decryption key $sk_d = k = k'_0 + ak'_1$.

Game 7: In this game, decryption is switched back to $m := \gamma/(\rho^{k'_0}\hat{\rho}^{k'_1})$ with decryption key $sk_d = (k'_0, k'_1)$. Indistinguishability holds due to unbounded partial-simulation soundness.

Game 8: Now we switch back to using otfK_1 in KeyGen. he challenge encryption is performed as follows: Generate $w, w' \xleftarrow{\$} \mathbb{Z}_q$ and compute:

$$c := (\mathbf{g}^w, \mathbf{g}^{aw'}, m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw'}, \mathsf{otfS}(\sigma, \tau_1, (\mathbf{g}^w, \mathbf{g}^{aw'}); \gamma)),$$

where $\gamma := m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw'}$. In addition, we switch back to sfV for verification before decryption.

Indistinguishability holds due to switching from the intermediate world to the OTFS world.

Game 9: Now the Challenge encryption is performed as follows: Generate $w \xleftarrow{\$} \mathbb{Z}_q$ and compute:

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw}, \mathsf{otfS}(\sigma, \tau_1, (\mathbf{g}^w, \mathbf{g}^{aw}); \gamma)),$$

where $\gamma := m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw}$.

Indistinguishability holds due to DDH.

Game 10: Now we switch back to the semi-functional world. The challenge encryption is performed as

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw}, \mathsf{sfS}(\sigma, \tau, (\mathbf{g}^w, \mathbf{g}^{aw}), \beta = 1; \gamma)),$$

where $\gamma := m_1 \cdot \mathbf{g}^{k'_0 w}\mathbf{g}^{k'_1 aw}$.

This step is indistinguishable from Game 9 due to one-time full ZK.

Game 11: Now instead of giving k'_0, k'_1 as sk_d, KeyGen gives $k = k'_0 + ak'_1$. The challenge encryption is computed as:

$$c := (\mathbf{g}^w, \mathbf{g}^{aw}, m_1 \cdot \mathbf{g}^{kw}, \mathsf{sfS}(\sigma, \tau, (\mathbf{g}^w, \mathbf{g}^{aw}), \beta = 1; \gamma)),$$

where $\gamma := m_q \cdot \mathbf{g}^{kw}$. Decryption is switched back to $m := \gamma/\rho^k$.

Indistinguishability follows due to unbounded partial simulation soundness.

Game 12: In this game we switch back to the real prover for encryption, the real verifier for decryption and the semi-functional simulator for Eval. The only difference from Game 0 is that now m_1 is encrypted instead of m_0. Thus we are done. □

5 A Single Round UC Password-Based Key Exchange Protocol

5.1 Universally Composable Security

The Universally Composable (UC) framework [Can01] is a formal system for proving security of computational systems such as cryptographic protocols. The framework describes two probabilistic games: The *real world* that captures the protocol flows and the capabilities of an attacker, and the *ideal world* that captures what we think of as a secure system. The notion of security asserts that these two worlds are essentially equivalent.

5.2 UC Functionality for Password-Based Key Exchange

The essential elements of the Universal Composability framework can be found in [Can01]. We adopt the definition for password-based key exchange from Canetti et al [CHK+05]. The formal description is given in Figure 1.

The real-world protocol we provide is also shown to be secure when different sessions use the same common reference string (CRS). To achieve this goal, we consider the *universal Composability with joint state* (JUC) formalism of Canetti and Rabin [CR03]. This formalism provides a "wrapper layer" that deals with "joint state" among different copies of the protocol. In particular, defining

Functionality $\mathcal{F}_{\text{pake}}$

The functionality $\mathcal{F}_{\text{PAKE}}$ is parameterized by a security parameter k. It interacts with an adversary S and a set of parties via the following queries:

Upon receiving a query $(\mathsf{NewSession}, sid, P_i, P_j, pw, role)$ **from party** P_i**:**
Send $(\mathsf{NewSession}, sid, P_i, P_j, role)$ to S. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, pw'), then record (P_i, P_j, pw) and mark this record fresh.

Upon receiving a query $(\mathsf{TestPwd}, sid, P_i, pw')$ **from the adversary** S**:**
If there is a record of the form (P_i, P_j, pw) which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to S with "correct guess". If $pw \neq pw'$, mark the record interrupted and reply with "wrong guess".

Upon receiving a query $(\mathsf{NewKey}, sid, P_i, sk)$ **from** S**, where** $|sk| = k$**:**
If there is a record of the form (P_i, P_j, pw), and this is the first NewKey query for P_i, then:

- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, sk) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, pw') with $pw' = pw$, and a key sk' was sent to P_j, and (P_j, P_i, pw) was fresh at the time, then output (sid, sk') to P_i.
- In any other case, pick a new random key sk' of length k and send (sid, sk') to P_i.

Either way, mark the record (P_i, P_j, pw) as completed.

Upon receiving $(\mathsf{Corrupt}, sid, P_i)$ **from** S**:** if there is a (P_i, P_j, pw) recorded, return pw to S, and mark P_i corrupted.

Figure 1: The password-based key-exchange functionality $\mathcal{F}_{\text{PAKE}}$

a functionality $\mathcal{F}$ also implicitly defines the multi-session extension of $\mathcal{F}$(denoted by $\hat{\mathcal{F}}$)g: $\hat{\mathcal{F}}$ runs multiple independent copies of $\mathcal{F}$, where the copies are distinguished via sub-session IDs ssid. The JUC theorem [CR03] asserts that composing a protocol π that uses multiple independent copies of $\mathcal{F}$, with a single copy of a protocol that realizes $\hat{\mathcal{F}}$, preserves the security of π.

5.3 Main Idea of the UC Protocol using DSS-QA-NIZK

For the sake of exposition, let's call one party in the session the server and the other the client (there is no such distinction required in the actual protocol). The common reference string (CRS) defines a Diffie-Hellman language, i.e. $\rho = \mathbf{g}_2, \mathbf{g}_2^a$. The client picks a fresh Diffie-Hellman tuple by picking a witness r and computing $\langle \mathbf{x}_1 = \mathbf{g}_2^r, \mathbf{x}_2 = \mathbf{g}_2^{a \cdot r} \rangle$. It also computes a DSS-QA-NIZK proof π on this tuple, i.e. T, W as described in Section 2. It next modifies the Diffie-Hellman tuple using the password pwd it possesses. Essentially, it multiplies $\mathbf{x}_2$ by pwd to get a modified group element which we will denote by S. It next sends this modified Diffie-Hellman tuple, i.e. $\mathbf{x}_1, S$, and the T component of the proof π to the server. It retains W for later use. At this point it can erase the witness r.

As a first step, we intend to utilize an interesting property of the real-world verifier V of the DSS-QA-NIZK. Note, V on input $\mathbf{x}_1, \mathbf{x}_2$ and proof T, W, computes $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2)$, and outputs true

CRS : $\mathbf{g}_1,\ \mathbf{c}=\mathbf{g}_1^c,\ \mathbf{b}=\mathbf{g}_1^b,\ \mathbf{v}_1=\mathbf{g}_1^{-\Delta_1 b+d+c\cdot a},\ \mathbf{v}_2=\mathbf{g}_1^{-\Delta_2 b+e},$
$\mathbf{g}_2,\ \mathbf{a}=\mathbf{g}_2^a,\ \mathbf{d}=\mathbf{g}_2^d,\ \mathbf{e}=\mathbf{g}_2^e,\ \mathbf{w}_1=\mathbf{g}_2^{\Delta_1},\ \mathbf{w}_2=\mathbf{g}_2^{\Delta_2},$
where $\mathbf{g}_1 \xleftarrow{\$} \mathbb{G}_1,\ \mathbf{g}_2 \xleftarrow{\$} \mathbb{G}_2,$ and $c,b,a,d,e,\Delta_1,\Delta_2 \xleftarrow{\$} \mathbb{Z}_q$

Party P_i		Adv $\mathcal{A}$
Input $(\texttt{NewSession}, sid, \mathsf{ssid}, P_i, P_j, \text{pwd}, \textit{initiator/responder})$		
Choose $r_1, s_1 \xleftarrow{\$} \mathbb{Z}_q$.		
Set $R_1=\mathbf{g}_2^{r_1}$, $S_1=\text{pwd}\cdot\mathbf{a}^{r_1}$, $T_1=(\mathbf{d}\cdot\mathbf{e}^{i_1})^{r_1}$, $\hat{\rho}_1=\mathbf{b}^{s_1}$,	$\xrightarrow{R_1,S_1,T_1,\hat{\rho}_1}$	$\mathcal{A}$
$W_1=(\mathbf{w}_1\mathbf{w}_2^{i_1})^{r_1}$, where $i_1=\mathcal{H}(sid,\mathsf{ssid},P_i,P_j,R_1,S_1,\hat{\rho}_1)$,		
and erase r_1. Send R_1,S_1,T_1 and $\hat{\rho}_1$, and retain W_1.		
	$\xleftarrow{R_2',S_2',T_2',\hat{\rho}_2'}$	$\mathcal{A}$
If any of $R_2',S_2',T_2',\hat{\rho}_2'$ is not in their respective group or is 1,		
set $\text{sk}_1 \xleftarrow{\$} G_T$, else		
compute $i_2'=\mathcal{H}(sid,\mathsf{ssid},P_j,P_i,R_2',S_2',\hat{\rho}_2')$,		
$\rho_1=\mathbf{g}_1^{s_1}$, $\theta_1=\mathbf{c}^{s_1}$, $\gamma_1=(\mathbf{v}_1\mathbf{v}_2^{i_2'})^{s_1}$.		
Finally compute $\text{sk}_1=e(\rho_1,T_2')\cdot e(\theta_1,S_2'/\text{pwd})\cdot e(\gamma_1^{-1}.R_2')\cdot e(\hat{\rho}_2',W_1)$		
Output $(sid,\mathsf{ssid},\text{sk}_1)$.		

Figure 2: Single round UC-secure Password-authenticated KE under SXDH Assumption.

iff

$$e((\mathbf{v}_1\mathbf{v}_2^{\iota}),\mathbf{x}_1^{-1})\cdot e(\mathbf{c},\mathbf{x}_2)\cdot e(\mathbf{g}_1,T)\ =\ e(\mathbf{b},W).$$

Thus, it outputs true iff the left-hand-size (LHS) equals the right-hand-side (RHS) of the above equation. Note that the client sent $\mathbf{x}_1, S$ (i.e. $\mathbf{x}_2$ linearly modified by pwd) and T to the server. Assuming the server has the same password pwd, it can un-modify the received message and get $\mathbf{x}_2 = S/\text{pwd}$, and hence can compute this LHS (using the CRS). The client retained W, and can compute the RHS (using the CRS).

The intuition is that unless an adversary out-right guesses the password, it cannot produce a different $\mathbf{x}_1'$, S', T', such that $\mathbf{x}_1'$, $\tilde{S}'/\text{pwd}$, T' used to compute the LHS will match the RHS above. While we make this intuition rigorous later by showing a UC simulator, to complete the description of the protocol, and using this intuition, the client and server actually compute the LHS and RHS respectively of the following equation (for a fresh random $s \in \mathbb{Z}_q$ picked by the server):

$$e((\mathbf{v}_1\mathbf{v}_2^{\iota})^s,\mathbf{x}_1^{-1})\cdot e(\mathbf{c}^s,\mathbf{x}_2)\cdot e(\mathbf{g}_1^s,T)\ =\ e(\mathbf{b}^s,W). \tag{12}$$

Now note that for the client to be able to compute the RHS, it must have $\mathbf{b}^s$, since s was picked by the server afresh. For this purpose, the protocol requires that the server send $\mathbf{b}^s$ to the client (note this can be done independently and asynchronously of the message coming from the client). It is not difficult to see, from completeness of the prover and verifier of the DSS-QA-NIZK, that both parties compute the same quantity. Given the definition of the real-world CRS of the DSS-QA-NIZK for the DH language, the LHS of equation (12) can also be written as

$$e(\mathbf{g}_1^s,\mathbf{x}_1^{(\Delta_1'+\iota\Delta_2')})\cdot e(\mathbf{g}_1^s,\mathbf{x}_2^c)\cdot e(\mathbf{g}_1^s,T).$$

Now, by DDH assumption holding in group $\mathbb{G}_1$, $\mathbf{g}_1^s$ is random and independent given $\mathbf{b}^s$ (and $\mathbf{g}_1$, $\mathbf{b}$). Thus, the (identical) computation of the two parties is random and independent of their message flows as long as $\mathbf{x}_1^{(\Delta_1'+\iota\Delta_2')}\cdot\mathbf{x}_2^c\cdot T$ is not one (which happens with high probability).

Each pair of parties actually run two versions of the above protocol, where-in each party plays the part of client in one version, and the part of server in the other version. Each party then outputs the product of the LHS of (12) computation (in the server version) and the RHS of (12) computation (in the client version) as the session-key. This is the final UC-PAKE protocol described in Fig. 2 (with the parties identities, session identifiers and $\mathbf{b}^s$ from its server version, used as label). The quantity $\mathbf{x}_1$ is called R in the protocol, as subscripts will be used for other purposes.

5.4 Main Idea of the UC Simulator

We first *re-define* the various verifiers in the DSS-QA-NIZK for the DH language described in Section 2, to bring them in line with the above description. In particular, V is defined equivalently to be: the verifier V takes as input $\mathbf{CRS}_v$, a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$, and a proof $\pi = (T, W)$, computes $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$, picks a fresh random $s \in \mathbb{Z}_q$, and outputs true iff

$$e((\mathbf{v}_1 \mathbf{v}_2^{\iota}), \mathbf{x}_1^{-1})^s \cdot e(\mathbf{c}, \mathbf{x}_2)^s \cdot e(\mathbf{g}_1, T)^s \ = \ e(\mathbf{b}^s, W).$$

This is equivalent as long as $s \neq 0$.

The partial-simulation world private-verifier pV is now defined as: it checks a potential language member $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ and a proof T, W as follows: compute $\iota = \mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, l)$; pick s and s' randomly and independently from $\mathbb{Z}_q$, and if $\mathbf{x}_2 = \mathbf{x}_1^a$ and $T = \mathbf{x}_1^{d+\iota e}$ then set $\xi = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s'}$ and output true iff

$$e((\mathbf{v}_1 \mathbf{v}_2^{\iota}), \mathbf{x}_1^{-1})^s \cdot e(\mathbf{c}, \mathbf{x}_2)^s \cdot e(\mathbf{g}_1, T)^s \cdot \xi \ = \ e(\mathbf{b}^s, W). \tag{13}$$

This is equivalent to the earlier definition of pV with high probability by an information-theoretic argument, if the trapdoors used were generated by the semi-functional CRS generator sfK_1.

The UC simulator $\mathcal{S}$ will generate the CRS for $\widehat{\mathcal{F}}_{\text{PAKE}}$ using the semi-functional CRS generator sfK_1 for the Diffie-Hellman language. Note, $\mathcal{S}$ simulating the server can compute the LHS of equation (13), except $\mathcal{S}$ does not have direct access to pwd and hence cannot get $\mathbf{x}_2$ from the modified $\hat{S}$ that it receives. However, it can do the following: Use the $\mathsf{TestPwd}$ functionality of the ideal functionality $\widehat{\mathcal{F}}_{\text{PAKE}}$ with a pwd$'$ computed as $\hat{S}/\mathbf{x}_1^a$. If this pwd$'$ does not match the pwd recorded in $\widehat{\mathcal{F}}_{\text{PAKE}}$ for this session and party, then $\widehat{\mathcal{F}}_{\text{PAKE}}$ anyway outputs a fresh random session key. This is same as $\mathbf{x}_2 \, (= S/\text{pwd}) \neq \mathbf{x}_1^a$, which would also have resulted in the same computation on the LHS (note s', and hence ξ in this case, is random and independent of the transcript). If the pwd$'$ matched the pwd, the simulator is notified the same, and hence it can now do the following: if $T = \mathbf{x}_1^{d+\iota e}$ then set $\xi = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s'}$. Call $\widehat{\mathcal{F}}_{\text{PAKE}}$'s NewKey with session key $e((\mathbf{v}_1 \mathbf{v}_2^{\iota}), \mathbf{x}_1^{-1})^s \cdot e(\mathbf{c}, \mathbf{x}_1^a)^s \cdot e(\mathbf{g}_1, T)^s \cdot \xi$ (multiplied by a RHS computation of (13) in the simulation of the party as a client).

The UC Simulator $\mathcal{S}$ must also simulate $\mathbf{g}_2^r, \text{pwd} \cdot (\mathbf{g}_2^a)^r$ and the T component of the DSS-QA-NIZK, as that is the message sent out to the adversary by the real party. However, $\mathcal{S}$ does not have access to pwd. It can just generate a fake tuple $\mathbf{g}_2^r, \mu \cdot (\mathbf{g}_2^a)^r \cdot \mathbf{g}_2^{r'}$ (for some constant or randomly chosen group element μ, and some random and independent $r' \in \mathbb{Z}_q$). Now, the semi-functional (proof) simulator sfS of the DSS-QA-NIZK of Section 2 has an interesting property that when the tuple $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ does not belong to the language, the T component of the simulated proof does not even depend on $\mathbf{x}_2$. In the simulation, with high probability $\mathbf{x}_2$ is not going to be consistent regardless of μ and pwd, because of the way r' is chosen. Thus, $\mathcal{S}$ can simulate T from just $\mathbf{x}_1 \, (= \mathbf{g}_2^r)$.

The simulator also needs W, but only in the session-key computation phase. As mentioned above, if the password pwd' "decrypted" from the incoming message is not correct then the key is anyway set to be random, and hence a proper W is not even required. However, if the pwd' is correct, the simulator is notified of same, and hence it can compute W component of the proof by passing $\mathbf{x}_2 = \mu \cdot (\mathbf{g}_2^a)^r \cdot \mathbf{g}_2^{r'}/\mathrm{pwd}'$ along with $\mathbf{x}_1$ $(= \mathbf{g}_2^r)$ to sfS.

5.5 Main Idea of the Proof of UC Realization

The proof that the simulator $\mathcal{S}$ described above simulates the Adversary in the real-world protocol, follows essentially from the properties of the DSS-QA-NIZK, although not generically since the real-world protocol and the simulator use the verifiers V and pV (resp.) in a split fashion. However, as described above the proof is very similar and we give a broad outline here. The proof will describe various experiments between a challenger $\mathcal{C}$ and the adversary, which we will just assume to be the environment $\mathcal{Z}$ (as the adversary $\mathcal{A}$ can be assumed to be just dummy and following $\mathcal{Z}$'s commands). In the first experiment the challenger $\mathcal{C}$ will just be the combination of the code of the simulator $\mathcal{S}$ above and $\widehat{\mathcal{F}}_{\mathrm{PAKE}}$. In particular, after the environment issues a $\mathsf{NewSession}$ request with a password pwd, the challenger gets that password. So, while in the first experiment, the challenger (copying $\mathcal{S}$) does not use pwd directly, from the next experiment on-wards, it can use the pwd. Thus, the main goal of the ensuing experiments is to modify the fake tuples $\mathbf{g}_2^r, \mu \cdot (\mathbf{g}_2^a)^r \cdot \mathbf{g}_2^{r'}$ by real tuples (as in real-world) $\mathbf{g}_2^r, \mathrm{pwd} \cdot (\mathbf{g}_2^a)^r$, since the challenger has access to pwd. This is accomplished by a hybrid argument, modifying one instance at a time using DDH assumption in group $\mathbb{G}_2$ and using one-time full-ZK property (and using the otfS proof simulator for that instance). A variant of the one-time full-ZK semi-functional verifier sfV (just as the variants for pV and V described above) is easily obtained.

Once all the instances are corrected, i.e. R, S generated as $\mathbf{g}_2^r, \mathrm{pwd} \cdot (\mathbf{g}_2^a)^r$, the challenger can switch to the real-world because the tuples $R, S/\mathrm{pwd}$ are now Diffie-Hellman tuples. This implies that the session keys are generated using the V variant described above, which is exactly as in the real-world.

5.6 Dynamic Corruption

The UC protocol described above is also UC-secure against dynamic corruption of parties by the Adversary in the erasure model. In the real-world when the adversary corrupts a party (with a $\mathsf{Corrupt}$ command), it gets the internal state of the party. Clearly, if the party has already been invoked with a $\mathsf{NewSession}$ command then the password pwd is leaked at the minimum, and hence the ideal functionality $\mathcal{F}_{\mathrm{PAKE}}$ leaks the password to the Adversary in the ideal world. In the protocol described above, the Adversary also gets W and s, as this is the only state maintained by each party between sending $R, S, T, \hat{\rho}$, and the final issuance of session-key. Simulation of s is easy for the simulator $\mathcal{S}$ since it also generates s exactly as in the real world. For generating W, which $\mathcal{S}$ had postponed to computing till it received an incoming message from the adversary, it can now use the pwd which it gets from $\widehat{\mathcal{F}}_{\mathrm{PAKE}}$ by issuing a $\mathsf{Corrupt}$ call to $\widehat{\mathcal{F}}_{\mathrm{PAKE}}$. More precisely, it issues the $\mathsf{Corrupt}$ call, and gets pwd, and then calls the semi-functional simulator with $\mathbf{x}_2 = \mu \cdot (\mathbf{g}_2^a)^r \cdot \mathbf{g}_2^{r'}/\mathrm{pwd}$ along with $\mathbf{x}_1$ $(= \mathbf{g}_2^r)$ to get W.

Without loss of generality, we can assume that in the real-world if the Adversary (or Environmanet $\mathcal{Z}$) corrupts an instance before the session key is output then the instance does not output

any session key. This is so because the Adversary (or $\mathcal{Z}$) either sets the key for that session or can compute it from the internal state it broke into.

5.7 Realization of the UC-PAKE functionality

Theorem 5 *Assuming the existence of SXDH-hard groups, the protocol in Fig 2 securely realizes the $\widehat{\mathcal{F}}_{\text{PAKE}}$ functionality in the $\mathcal{F}_{\text{CRS}}$ hybrid model, in the presence of dynamic corruption adversaries.*

6 Proof of Realization of the UC-PAKE Functionality

In this section we state and prove that the protocol in Fig. 2 realizes the multi-session ideal functionality $\widehat{\mathcal{F}}_{\text{PAKE}}$.

Theorem 6 *Assuming the existence of SXDH-hard groups, the protocol in Fig 2 securely realizes the $\widehat{\mathcal{F}}_{\text{PAKE}}$ functionality in the $\mathcal{F}_{\text{CRS}}$ hybrid model, in the presence of dynamic corruption adversaries.*

We start by defining the UC simulator in detail.

6.1 The Simulator for the UC Protocol

We will assume that the adversary $\mathcal{A}$ in the UC protocol is dummy, and essentially passes back and forth commands and messages from the environment $\mathcal{Z}$. Thus, from now on we will use environment $\mathcal{Z}$ as the real adversary, which outputs a single bit. The simulator $\mathcal{S}$ will be the ideal world adversary for $\widehat{\mathcal{F}}_{\text{PAKE}}$. It is a universal simulator that uses $\mathcal{A}$ as a black box.

For each instance (session and a party), we will use subscript 2 along with a prime, to refer to variables received in the message from $\mathcal{Z}$(i.e $\mathcal{A}$), and use subscript 1 to refer to variables computed in the instance under consideration. We will call a message *legitimate* if it was not altered by $\mathcal{Z}$, and delivered in the correct session, and to the correct party.

The simulator $\mathcal{S}$ picks the CRS just as semi-functional CRS generator sfK_1 picks the CRS for the DSS-QA-NIZK. This is statistically same as the real-world CRS in the UC protocol, except $\mathcal{S}$ sets $d = d_1 + a \cdot d_2$, and $e = e_1 + a \cdot e_2$, where d_1, d_2, e_1, e_2 are chosen randomly and independently from $\mathbb{Z}_q$. It retains $a, c, b, d_1, d_2, e_1, e_2, \Delta_1', \Delta_2'$ as trapdoors.

The next main difference in the simulation of the real world parties is that $\mathcal{S}$ uses a dummy message μ instead of the real password which it does not have access to. Further, it decrypts the incoming message R_2', S_2', T_2' to compute a pwd$'$, which it uses to call the ideal functionality's test function. If the test succeeds, it produces a sk (see below) and sends it to the ideal functionality to be output to the party concerned.

6.1.1 New Session: Sending a message to $\mathcal{Z}$.

On message (NewSession, *sid*, ssid, i, j, role) from $\widehat{\mathcal{F}}_{\text{PAKE}}$, S starts simulating a new instance of the protocol Π for party P_i, peer P_j, session identifier ssid, and CRS set as above. We will denote this instance by (P_i, ssid). To simulate this instance, S chooses r_1, r_1', r_1'', s_1 at random, and sets $R_1 = \mathbf{g}_2^{r_1}$, $S_1 = \mu \cdot \mathbf{a}^{r_1} \cdot \mathbf{g}_2^{r_1'}$, $T_1 = \mathbf{g}_2^{r_1 \cdot (d_1 + \iota_1 e_1)} \cdot \mathbf{g}_2^{r_1''}$, $\hat{\rho}_1 = \mathbf{b}^{s_1}$ where $\iota_1 = \mathcal{H}(sid, \mathsf{ssid}, P_i, P_j, R_1, S_1, \hat{\rho}_1)$ (note the use of μ instead of pwd). Note this is what sfS would compute for T_1. It retains r_1, r_1', r_1'', s_1 (and μ if chosen randomly). It then hands $R_1, S_1, T_1, \hat{\rho}_1$ to $\mathcal{Z}$ on behalf of this instance.

6.1.2 On Receiving a Message from $\mathcal{Z}$.

On receiving a message $R'_2, S'_2, T'_2, \hat{\rho}'_2$ from $\mathcal{Z}$ intended for this instance (P_i, ssid), the simulator S makes the real world protocol checks, namely group membership and non-triviality. If any of the checks fail, it issues a TestPwd call to $\widehat{\mathcal{F}}_{\text{PAKE}}$ with the dummy password μ, followed by a NewKey call with a random session key, which leads to the functionality issuing a random and independent session key to the party P_i (regardless of whether the instance was interrupted or compromised).

Otherwise, if the message received from $\mathcal{Z}$ is same as message sent by $\mathcal{S}$ on behalf of peer P_j in session ssid, then $\mathcal{S}$ just issues a NewKey call for P_i.

Else, it computes pwd′ by decrypting S'_2, i.e. setting it to $S'_2/(R'_2)^a$. $\mathcal{S}$ then calls $\widehat{\mathcal{F}}_{\text{PAKE}}$ with (TestPwd, ssid, P_i, pwd′). Regardless of the reply from $\mathcal{F}$, it then issues a NewKey call for P_i with key computed as follows (recall, $R_1, S_1, \iota_1, r'_1, r''_1$ from earlier in this instance when the message was sent to $\mathcal{Z}$).

$$\iota'_2 = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R'_2, S'_2, \hat{\rho}'_2),\ \rho_1 = \mathbf{g}_1^{s_1},\ \theta_1 = \mathbf{c}^{s_1},\ \gamma_1 = (\mathbf{v}_1\mathbf{v}_2^{\iota'_2})^{s_1},$$
$$W_1 = R_1^{(\Delta'_1 + d_1 + \iota_1(\Delta'_2 + e_1))/b} \cdot (S_1/\text{pwd}')^{c/b} \cdot \mathbf{g}_2^{r''_1/b}.$$

if $T'_2 = (R'_2)^{d+\iota'_2 e}$ then set $\xi_1 = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s'_1}$, where s'_1 is a fresh random value. Call $\widehat{\mathcal{F}}_{\text{PAKE}}$'s NewKey with session key

$$e(\gamma_1^{-1}, R'_2) \cdot e(\theta_1, (R'_2)^a) \cdot e(\rho_1, T'_2) \cdot e(\hat{\rho}'_2, W_1) \cdot \xi_1.$$

Note that this is how sfS would compute W_1. By definition of $\widehat{\mathcal{F}}_{\text{PAKE}}$, this has the effect that if the pwd′ was same as the actual pwd previously recorded in $\widehat{\mathcal{F}}_{\text{PAKE}}$ (for this instance) then the session key is determined by the Simulator as above, otherwise the session key is set to a random and independent value.

6.1.3 Corruption

On receiving a Corrupt call from $\mathcal{Z}$ for instance P_i in session ssid, the simulator $\mathcal{S}$ calls the Corrupt routine of $\widehat{\mathcal{F}}_{\text{PAKE}}$ to obtain pwd. If $\mathcal{S}$ had already output a message to $\mathcal{Z}$, and not output sk_1 it computes

$$W_1 = R_1^{(\Delta'_1 + d_1 + \iota_1(\Delta'_2 + e_1))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r''_1/b}$$

and outputs this W_1 along with pwd, and ρ_1, θ_1, γ_{11}, γ_{12} as internal state of P_i. Note that this computation of W_1 is identical to the computation of W_1 in the computation of sk_1 (which is really output to $\mathcal{Z}$ when pwd′ = pwd).

As explained in Section 5.6, if $\mathcal{Z}$ corrupts an instance before it has output its session key, we will assume w.l.o.g. that the session-key is not output at all by that instance.

6.2 Proof of Indistinguishability - Series of Experiments

We now describe a series of experiments between a probabilistic polynomial time challenger $\mathcal{C}$ and the environment $\mathcal{Z}$, starting with Expt_0 which we describe next. We will show that the view of $\mathcal{Z}$ in Expt_0 is same as its view in UC-PAKE ideal-world setting with $\mathcal{Z}$ interacting with $\widehat{\mathcal{F}}_{\text{PAKE}}$ and the UC-PAKE simulator $\mathcal{S}$ described above in Section 6.1. We end with an experiment which is identical to the real world execution of the protocol in Fig 2. We will show that the environment

has negligible advantage in distinguishing between these series of experiments, leading to a proof of realization of $\mathcal{F}_{\text{PAKE}}$ by the protocol Π.

Here is the complete code in Expt_0 (stated as it's overall experiment with $\mathcal{Z}$):

1. The challenger $\mathcal{C}$ picks the CRS just as in the real world, except it sets $d = d_1 + a \cdot d_2$, and $e = e_1 + a \cdot e_2$, where d_1, d_2, e_1, e_2 are chosen randomly and independently from $\mathbb{Z}_q$. It retains $a, c, b, d1, d2, e1, e2, \Delta_1, \Delta_2$ as trapdoors.

2. On receiving $\texttt{NewSession}, sid, \mathsf{ssid}, P_i, P_j, \text{pwd}, role$ from $\mathcal{Z}$, $\mathcal{C}$ generates $R_1, S_1, T_1, \hat{\rho}_1$ by choosing r_1, r_1', r_1'', s_1 at random, and setting $R_1 = \mathbf{g}_2^{r_1}$, $S_1 = \mu \cdot \mathbf{a}^{r_1} \cdot \mathbf{g}_2^{r_1'}$, $T_1 = \mathbf{g}_2^{r_1 \cdot (d_1 + \iota_1 e_1)} \cdot \mathbf{g}_2^{r_1''}$, $\hat{\rho}_1 = \mathbf{b}^{s_1}$. It sends these values to $\mathcal{Z}$.

3. On receiving $R_2', S_2', T_2', \hat{\rho}_2'$ from $\mathcal{Z}$, intended for session ssid and party P_i (and assuming no corruption of this instance)

   (a) if the received elements are either not in their respective groups, or are trivially 1, output sk_1 chosen randomly and independently from $\mathbb{G}_T$.

   (b) Otherwise, if the message received is identical to message sent by $\mathcal{C}$ in the same session (i.e. same SSID) on behalf of the peer, then output $\text{sk}_1 \xleftarrow{\$} \mathbb{G}_T$ (unless the simulation of peer also received a legitimate message and its key has already been set, in which case the same key is used to output sk_1 here).

   (c) Else, compute $\text{pwd}' = S_2'/(R_2')^a$. If $\text{pwd}' \neq \text{pwd}$ (note pwd was given in `NewSession` request), then output sk_1 randomly and independently from $\mathbb{G}_T$.

   (d) Else, compute $\iota_2' = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2', S_2', \hat{\rho}_2')$, $\rho_1 = \mathbf{g}_1^{s_1}$, $\theta_1 = \mathbf{c}^{s_1}$, $\gamma_1 = (\mathbf{v}_1 \mathbf{v}_2^{\iota_2'})^{s_1}$, $W_1 = R_1^{(\Delta_1' + d_1 + \iota_1(\Delta_2' + e_1))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r_1''/b}$. if $T_2' = (R_2')^{d + \iota_2' e}$ then set $\xi_1 = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s_1'}$, where s_1' is a fresh random value. Output

$$e(\gamma_1^{-1}, R_2') \cdot e(\theta_1, S_2'/\text{pwd}) \cdot e(\rho_1, T_2') \cdot e(\hat{\rho}_2', W_1) \cdot \xi_1.$$

4. On a $\mathsf{Corrupt}$ call, if step 2 has already happened then output s_1, pwd and $W_1 = R_1^{(\Delta_1' + d_1 + \iota_1(\Delta_2' + e_1))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r_1''/b}$.

All outputs of sk_1 are also accompanied with sid, ssid (but are not mentioned above for ease of exposition).

Note that each instance has two asynchronous phases: a phase in which $\mathcal{C}$ outputs $R_1, S_1, ...$ to $\mathcal{Z}$, and a phase where it receives a message from $\mathcal{Z}$. However, $\mathcal{C}$ cannot output sk_1 until it has completed both phases. These orderings are dictated by $\mathcal{Z}$. We will consider two different kinds of temporal orderings. A temporal ordering of different instances based on the order in which $\mathcal{C}$ outputs sk_1 in an instance will be called **temporal ordering by key output**. A temporal ordering of different instances based on the order in which $\mathcal{C}$ outputs its first message (i.e. $R_1, S_1, ...$) will be called **temporal ordering by message output**. It is easy to see that $\mathcal{C}$ can dynamically compute both these orderings by maintaining a counter (for each ordering).

It is straightforward to inspect that the view of $\mathcal{Z}$ in Expt_0 is identical to its view in its combined interaction with $\widehat{\mathcal{F}}_{\text{PAKE}}$ and $\mathcal{S}$, as $\mathcal{C}$ has just combined the code of $\widehat{\mathcal{F}}_{\text{PAKE}}$ and $\mathcal{S}$ (noting that in step 3(d), $\text{pwd} = \text{pwd}'$)

Expt_1 : In this experiment step 3(c) is dropped altogether and step 3(d) is altered as follows: In step 3(d) in Expt_0, ξ_1 is set based on the condition $T_2' = (R_2')^{d+\iota_2' e}$. In Expt_1 in step 3(d), the computation of ξ_1 is now changed to: if $(S_2' = \text{pwd} \cdot (R_2')^a)$ and $(T_2' = (R_2')^{d+\iota_2' e})$ then set $\xi_1 = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s_1'}$, where s_1' is a fresh random value. Rest of the computation of sk_1 in step 3(d) remains the same.

We claim that the view of $\mathcal{Z}$ is statistically identical in Expt_0 and Expt_1. This follows by noting that the computation of sk_1 has the extra factor ξ_1. Since s_1' is chosen independently of the rest of the transcript, this will be a random and independent factor in $\mathbb{G}_T$ if $S_2' \neq \text{pwd} \cdot (R_2')^a$, which is equivalent to the condition $\text{pwd}' \neq \text{pwd}$ in Expt_0. The condition $S_2' = \text{pwd} \cdot (R_2')^a$ held in step 3(d) in Expt_0, as that step was only reached if this condition held.

Expt_2 : In this experiment step 3(b) is modified as follows:
Step 3(b): Otherwise, if the message received is identical to message sent by $\mathcal{C}$ in the same session (i.e. same SSID) on behalf of the peer, then

- if simulation of peer also received a legitimate message and its key has already been set, then output that same key here. Else, got to step 3(d).

If the message received is identical to the message sent by $\mathcal{C}$ in the same session, then clearly S_2' received is not equal to $\text{pwd} \cdot (R_2')^a$, since $\mathcal{C}$ had produced S_1 with a random $\mathbf{g}_2^{r_1'}$ factor. Thus, step 3(d) will output a random sk_1 in Expt_2 as well (because of the way ξ_1 is defined).

At this point, Expt_2 can equivalently be written as follows:

1. The challenger $\mathcal{C}$ picks the CRS just as in the real world, except it sets $d = d_1 + a \cdot d_2$, and $e = e_1 + a \cdot e_2$, where d_1, d_2, e_1, e_2 are chosen randomly and independently from $\mathbb{Z}_q$. It retains $a, c, b, d_1, d_2, e_1, e_2, \Delta_1, \Delta_2$ as trapdoors.

2. On receiving $\texttt{NewSession}, sid, \mathsf{ssid}, P_i, P_j, \text{pwd}, role$ from $\mathcal{Z}, \mathcal{C}$ generates $R_1, S_1, T_1, \hat{\rho}_1$ by choosing r_1, r_1', r_1'', s_1 at random, and setting $R_1 = \mathbf{g}_2^{r_1}$, $S_1 = \mu \cdot \mathbf{a}^{r_1} \cdot \mathbf{g}_2^{r_1'}$, $T_1 = \mathbf{g}_2^{r_1 \cdot (d_1 + \iota_1 e_1)} \cdot \mathbf{g}_2^{r_1''}$, $\hat{\rho}_1 = \mathbf{b}^{s_1}$. It sends these values to $\mathcal{Z}$.

3. On receiving $R_2', S_2', T_2', \hat{\rho}_2'$ from $\mathcal{Z}$(and assuming no corruption of this instance),

   (a) if the received elements are either not in their respective groups, or are trivially 1, output sk_1 chosen randomly and independently from $\mathbb{G}_T$.

   (b) Otherwise, if the message received is identical to message sent by $\mathcal{C}$ in the same session (i.e. same SSID) on behalf of the peer **and** if simulation of peer also received a legitimate message and its key has already been set, then output that same key here.

   (c) -

   (d) compute $\iota_2' = \mathcal{H}(sid, \mathsf{ssid}, P_j, P_i, R_2', S_2', \hat{\rho}_2')$, $\rho_1 = \mathbf{g}_1^{s_1}$, $\theta_1 = \mathbf{c}^{s_1}$, $\gamma_1 = (\mathbf{v}_1 \mathbf{v}_2^{\iota_2'})^{s_1}$, $W_1 = R_1^{(\Delta_1' + d_1 + \iota_1(\Delta_2' + e_1))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r_1''/b}$. if $(S_2' = \text{pwd} \cdot (R_2')^a)$ and $(T_2' = (R_2')^{d+\iota_2' e})$ then set $\xi_1 = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s_1'}$, where s_1' is a fresh random value. Output

$$e(\gamma_1^{-1}, R_2') \cdot e(\theta_1, S_2'/\text{pwd}) \cdot e(\rho_1, T_2') \cdot e(\hat{\rho}_2', W_1) \cdot \xi_1.$$

4. On a Corrupt call, if step 2 has already happened then output s_1, pwd and $W_1 = R_1^{(\Delta'_1+d_1+\iota_1(\Delta'_2+e_1))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r''_1/b}$.

Expt$_3$: In this experiment the challenger in step 2 computes S_1 in each instance as $\text{pwd} \cdot \mathbf{a}^{r_1} \cdot \mathbf{g}_2^{r'_1}$. Note the use of pwd instead of μ.

This is statistically the same, as in each instance the challenger picks a fresh and random r'_1, and it is not used anywhere else.

Expt$_4$: In each instance T_1 is computed as follows: $T_1 = \mathbf{g}_2^{r_1 \cdot d+\iota_1 e} \cdot \mathbf{g}_2^{r''_1}$. Further, later on in the instance, on receiving a message from $\mathcal{Z}$, W_1 is computed as $W_1 = R_1^{(\Delta'_1+d+\iota_1(\Delta'_2+e))/b} \cdot (S_1/\text{pwd})^{c/b} \cdot \mathbf{g}_2^{r''_1/b}$.

This is statistically identical as T_1 has a random factor $\mathbf{g}_2^{r''_1}$ and W_1 has a random factor $\mathbf{g}_2^{r''_1/b}$, and r''_1 is not used anywhere else. This change is equivalent to experiment $\mathbf{H}_1$ in the proof of one-time full-ZK in Section 3 (see 9).

Expt$_5$: In this experiment the challenger $\mathcal{C}$ generates the CRS slightly differently. It picks Δ'_1 and Δ'_2 at random, and sets $\Delta_1 = (\Delta'_1 + d + c \cdot a)/b$, and $\Delta_2 = (\Delta'_2 + e)/b$. Rest of the CRS computation remains the same. Note that a is not used anymore in the computation of group $\mathbb{G}_1$ CRS elements.

The view of the adversary in Expt_4 and Expt_5 is statistically the same.

Expt$_6$: In each instance S_1 is computed as follows: $\text{pwd} \cdot \mathbf{a}^{r_1}$. Further, T_1 is computed as follows: $T_1 = R_1^{d+\iota_1 e}$. Further, later on in the instance, on receiving a message from $\mathcal{Z}$, W_1 is computed as $W_1 = R_1^{(\Delta'_1+d+\iota_1(\Delta'_2+e))/b} \cdot (S_1/\text{pwd})^{c/b}$. In other words, the computation of S_1, T_1 and W_1 is as in the real-world (this can be seen from the definition of $\mathbf{w}_1$ and $\mathbf{w}_2$ in the CRS).

To show that Expt_5 is computationally indistinguishable from Expt_6, we define several hybrid experiments $\mathsf{Expt}_{5,i}$ inductively. Experiment $\mathsf{Expt}_{5,0}$ is identical to Expt_5. If there are a total of N instances, $\mathsf{Expt}_{5,N}$ will be identical to Expt_6. Experiment $\mathsf{Expt}_{5,1+1}$ differs from experiment $\mathsf{Expt}_{5,i}$ in only (temporally ordered by message output) the $(i+1)$-th instance. While in $\mathsf{Expt}_{5,i}$, the $(i+1)$-th instance is simulated by $\mathcal{C}$ as in Expt_5, in $\mathsf{Expt}_{5,i+1}$ this instance is simulated as in Expt_6.

Lemma 7 *For all $i : 0 \leq i \leq N$, the view of $\mathcal{Z}$ in experiment $\mathsf{Expt}_{5,i+1}$ is computationally indistinguishable from the view of $\mathcal{Z}$ in $\mathsf{Expt}_{5,i}$.*

Proof: The proof will follow the proof of one-time full ZK in Section 3, except that we will also employ DDH to replace the fake tuples with real tuples.

We define several hybrid experiments. Experiment $\mathbf{H}_0$ is identical to $\mathsf{Expt}_{5,i}$.

In experiment $\mathbf{H}_1$, the CRS is picked differently by $\mathcal{C}$. instead of picking d_1, d_2 at random (and similarly e_2, e_2 at random), picks d and d_2 randomly and independently and sets $d_1 = d - a \cdot d_2$ (and $e_1 = e - a \cdot e_2$). Rest of the CRS computation remains the same, and it retains a, c, b, d, d_2, e, e_2, Δ'_1, Δ'_2. This is statistically the same distribution.

In $\mathbf{H}_2$, in the $(i+1)$-th instance T and W are computed differently:

$$T = R_1^{d+\iota_1 e - a\cdot(d_2+\iota_1 e_2)} \cdot (S_1/\text{pwd})^{d_2+\iota_1 e_2},\ W = R_1^{(\Delta_1'+d+\iota_1\cdot(\Delta_2'+e)-a\cdot(d_2+\iota_1 e_2))/b} \cdot (S_2/\text{pwd})^{(d_2+\iota_1 e_2+c)/b}. \tag{14}$$

This is statistically the same as d_2 was chosen randomly and independently of d. All other instances are only using d and e in their computation. Thus, the r_1'' in this instance can be replaced by $d_2 + \iota_1 e_2$.

In the next experiment $\mathbf{H}_3$, in step 3(d) of each instance, the computation of ξ_1 is done as follows: if $(T_2' = (R_2')^{d_1+\iota_2' e_1} \cdot (S_2'/\text{pwd})^{d_2+\iota_2' e_2})$ then set $\xi_1 = \mathbf{1}_T$ else set $\xi = \epsilon(\mathbf{g}_1, \mathbf{g}_2)^{s_1'}$, where s_1' is a fresh random value. Rest of the computation of sk_1 remains the same. This is the same as using the sfV instead of pV, and the proof of $\mathbf{H}_3$ being statistically indistinguishable from $\mathbf{H}_2$ is exactly the same as in proof of $\mathbf{H}_2$ and $\mathbf{H}_3$ in one-time full-ZK property proof of theorem 1 noting that $sid, \mathsf{ssid}, P, P_j$ and $\hat{\rho}$ are used as label.

In the next experiment $\mathbf{H}_4$, the computation of each of the instances (except the $(I+1)$-th instance undergoes the following change: in the j-th instance, if $j < (i+1)$, then set $r_1'' = 0$, else choose r_1'' at random. Compute $T_1 = R_1^{d_1+\iota_1 e_1} \cdot (S_1/\text{pwd})^{d_2+\iota_1 e_2} \cdot \mathbf{g}_2^{r_1''}$. Further, later on in the instance, on receiving a message from $\mathcal{Z}$, W_1 is computed as $W_1 = R_1^{(\Delta_1'+d_1+\iota_1(\Delta_2'+e_1))/b} \cdot (S_1/\text{pwd})^{(d_2+\iota_1 e_2+c)/b} \cdot \mathbf{g}_2^{r_1''/b}$.

Experiments $\mathbf{H}_4$ and $\mathbf{H}_3$ are statistically identical, and the proof is same as for experiments Expt_3 and Expt_4.

In the next experiment $\mathbf{H}_5$, the challenger produces the rest of the CRS using $\rho = \mathbf{g}_2, \mathbf{g}_2^a$, where a is chosen randomly from $\mathbb{Z}_q$. In particular, it chooses $d_1, d_2, e_1, e_2, \Delta_1', \Delta_2', b, c$ at random, and defines the CRS using ρ and these values. This is statistically the same.

In the next experiment $\mathbf{H}_6$, the challenger generates the S_1 in the $(i+1)$-th instance as follows: $S_1 = \text{pwd}\cdot\mathbf{a}^{r_1}$. That the view of $\mathcal{Z}$ in experiments $\mathbf{H}_5$ and $\mathbf{H}_6$ are computationally indistinguishable follows from the DDH assumption in group $\mathbb{G}_2$.

Now, we unwind our way back to $\mathsf{Expt}_{5,i+1}$ following all of the above hybrid games back-wards, and that completes the proof. □

Expt_7 : In this experiment, the challenger $\mathcal{C}$ sets the CRS by just directly choosing Δ_1, Δ_2 at random, instead of first choosing Δ_1', Δ_2' and then setting Δ_1 etc. in terms of the primed quantities and c, b, a, d and e. This means $\mathbf{v}_1$ is now again computed as in the real world CRS. Note, now c and b are not used in simulation of group $\mathbb{G}_2$ CRS elements.

It is easy to see that the view of $\mathcal{Z}$ is unchanged in going from Expt_8 to Expt_7.

Expt_8 : In this experiment in step 3(d), in each instance $\mathcal{C}$ picks another fresh random value $c_1' \in \mathbb{Z}_q$ (independent of s_1'), and computes ξ_1 differently as follows (all other values are computed as in Expt_4):
$\xi_1 = e(\mathbf{g}_1, T_2' \cdot (S_2'/\text{pwd})^{c_1'})^{s_1'} \cdot e(\mathbf{g}_1, R_2')^{-(d+\iota_2' e + c_1'\cdot a)s_1'}$.

This computation of ξ_1 is statistically indistinguishable by a simple information-theoretic argument.

Expt_9 : In this experiment, in each instance ξ_1 is computed without using a fresh c_1', but instead using c itself, i.e. $\xi_1 = e(\mathbf{g}_1, T_2' \cdot (S_2'/\text{pwd})^{c})^{s_1'} \cdot e(\mathbf{g}_1, R_2')^{-(d+\iota_2' e + ca)s_1'}$.