Strong Cryptography from Weak Secrets - Building Efficient PKE and IBE from Distributed Passwords

Xavier Boyen, Céline Chevalier, Georg Fuchsbauer, David Pointcheval

This is the full version of the extended abstract which appears in Africacrypt '10 (3 – 6 may 2010, Stellenbosch, South Africa)
D. Bernstein and T. Lange Eds., Springer-Verlag, LNCS 6055, pages 297–315.

Strong Cryptography from Weak Secrets

Building Efficient PKE and IBE from Distributed Passwords

Xavier Boyen[1], Céline Chevalier[2], Georg Fuchsbauer[3], and David Pointcheval[3]

[1] Université de Liège, Belgium
[2] Telecom ParisTech, Paris, France
[3] École Normale Supérieure, CNRS-INRIA, Paris, France

Abstract. Distributed-password public-key cryptography (DPwPKC) allows the members of a group of people, each one holding a small secret password only, to help a leader to perform the private operation, associated to a public-key cryptosystem. Abdalla *et al.* recently defined this tool [1], with a practical construction. Unfortunately, the latter applied to the ElGamal decryption only, and relied on the DDH assumption, excluding any recent pairing-based cryptosystems. In this paper, we extend their techniques to support, and exploit, pairing-based properties: we take advantage of pairing-friendly groups to obtain efficient (simulation-sound) zero-knowledge proofs, whose security relies on the Decisional Linear assumption. As a consequence, we provide efficient protocols, secure in the standard model, for ElGamal decryption as in [1], but also for Linear decryption, as well as extraction of several identity-based cryptosystems [6, 4]. Furthermore, we strenghten their security model by suppressing the useless testPwd queries in the functionality.

1 Introduction

Recently, Abdalla *et al.* [1] proposed the notion of distributed-password public-key cryptography (DPwPKC), which allows the members of a group of people, each one holding a small independent secret password, to act collectively (for the benefit of one of them, who "owns" the group) as the custodian of a private key in some ordinary public-key cryptosystem — without relying on any secure (secret and/or authentic) storage — as long as each member remembers his or her password. Precisely, in DPwPKC, the members initially create a "virtual" key pair $(\mathsf{sk}, \mathsf{pk})$, by engaging in some distributed protocol over adversarial channels, where only pk is revealed, while sk is implicitly determined by the collection of passwords. Third parties can perform the public-key operation(s) of the underlying system using pk. Members can help the leader of the group perform private-key operation(s) in a distributed manner, by engaging in some protocol using only their knowledge of their respective passwords.

Password-based public-key cryptography is generally considered infeasible because password-based secret-key spaces are easy to enumerate, and the knowledge of the public key makes it possible to test the correct key from that space, without interacting with anyone (offline dictionary attack). In DPwPKC, there are as many passwords as participants, and (unlike in virtually all applications of passwords) the passwords are not meant to be shared: they are chosen independently by each player. Since the passwords need not be related, they will likely be diverse, and the min-entropy of their *combination* ought to grow linearly with the number of participants, even if every single password is itself minuscule. For instance, with ten players each holding a random 20-bit password, the virtual secret key will be a random 200-bit string, which is more than enough to build a secure public-key system for usual values of the security parameter. This is what makes sk in DPwPKC resistant to brute-force off-line dictionary attacks, even though the corresponding pk is public.

The main contribution of [1] was to define general functionalities for distributed password-based key generation and private computation in the UC model, and to give a construction for ElGamal decryption as a proof of concept. However, the construction proposed in [1] was merely illustrative because it required generic simulation-sound non-interactive zero-knowledge (SSNIZK) proofs for NP languages, which can only be performed efficiently in the random oracle model [3]. Furthermore, their distributed private computation protocol could only perform the task of computing c^{sk} from the implicit secret key sk, and the security of their protocol relied on the DDH assumption. Together, these restrictions limited its applicability to ElGamal decryption.

In this work, we first improve and strengthen the ideal functionalities defined in [1], by further restricting the information that the adversary can gain from an attack. This will make any protocol that we can prove to realize those functionalities stronger, since the simulation will have to work without this information. (Recall that in the UC model, the functionalities are supposed to capture everything that we allow the adversary (and thus the simulator) to learn.)

Then, we extend the techniques from [1] to support a much broader class of private-key operations in discrete-log-hard groups, including operations involving random ephemerals and/or operations in bilinear groups. More precisely, our construction still targets the distributed computation of c^{sk}, but under the Decision Linear assumption, which makes the proof more intricate since the DDH is now verifiable: we had to change the workings of the protocol to introduce secret values. Furthermore, the construction works for several values of c at once, and now allows to share random ephemerals in the exponent. It thus allows a much greater variety of public-key cryptosystems to be converted to distributed password-based cryptosystems, including extraction of identity-based private keys — thus giving us the new interesting notion of "password-based distributed identity-based encryption" (DPwIBE). Contrarily to regular IBE, the "central" key extraction authority is now distributed among a group of people (sufficiently many of them trusted), with the "master key" being implicitly contained in the collections of short independent passwords held by those users.

In the process of strengthening and generalizing the protocols, we also make them much more efficient. To do so, we develop special-purpose *simulation-sound* non-interactive zero-knowledge proofs (SSNIZK) for our languages of interest, *in the standard model*, and show how to use them instead of the inefficient general SSNIZK considered in [1]. We do this using bilinear maps, in the CRS model, relying on a classic decisional hardness assumption for bilinear groups. The SSNIZK proofs we construct revisit the techniques of [12] and use efficient proofs inspired by the recent Groth-(Ostrovsky)-Sahai sequence of efficient NIZK construction in bilinear groups [14], but do not trivially follow from them.

In summary, we revisit the recent and intriguing notion of DPwPKC, which we strengthen and extend in several directions. Our general results are, more precisely: 1) to propose new, simpler and stronger versions of the DPwPKC functionalities defined in [1] by getting rid of the testPwd query; 2) to show how we can realize the general functionalities much more efficiently, without random oracles, by constructing new SSNIZK proof techniques from bilinear pairings, inspired from the NIZK proof system proposed by Groth and Sahai [14] under the Decisional Linear assumption [5]; 3) to show how to apply our protocol to implement a very broad class of public-key cryptosystems from discrete-log-type hardness assumptions in bilinear groups; and 4) in particular, to show how to apply our technique to transform the pairing-based Boneh-Franklin [6] and Boneh-Boyen [4] identity-based cryptosystems (and most of the latter's many extensions), to realize the notion of distributed password-based identity-based encryption (DPwIBE). In DPwIBE, contrarily to regular IBE, the "central" key extraction authority is now distributed among a group of people (sufficiently many of them trusted), with the "master key" being implicitly contained in the collections of short independent passwords held by those users.

In order to do all this, and in particular to get the efficient pairing-based approach working, a number of new technical challenges had to be solved. We specifically mention the following: 1) the use of pairings not only helps us make efficient zero-knowledge proofs for various languages, it would also help the adversary verify the result of the private computation c^{sk} in the basic DPwPKC protocol from [1]. Since the UC model requires that the simulation be carried out until the end on both correct and incorrect inputs, this will make our new security reduction somewhat more intricate since the result sent at the end of the simulation is random and we do not want the adversary to become aware of it. 2) In connection with the stronger and simpler functionality definitions we propose, the adversary is no longer allowed to conduct *explicit* password compatibility tests prior to the private-key operation. This should intuitively further complicate the simulation, though we remarkably note that these queries were indeed useless in the proofs and thus getting rid of them has no negative impact. 3) Generally speaking, we achieved much of our security and efficiency gains over [1], by succeeding to make our protocols

being *fully robust* by the use of public verifications (computations of pairings) rather than intermediate validity tests (SSNIZK proofs, relying on the random oracle model in [1]). This is generally both more efficient (no more SSNIZK proofs) and more secure than testing, but it can lead to significantly more complex simulations owing to the ideal functionality being less "helpful".

2 Security Model

Split Functionalities. Throughout this paper, we assume basic familiarity with the universal composability framework [8]. See Appendix A for a short introduction of some UC notions we shall use in this work. Without any strong authentication mechanisms, the adversary can always partition the players into disjoint subgroups and execute independent sessions of the protocol with each subgroup, playing the role of the other players. Such an attack is unavoidable since players cannot distinguish the case in which they interact with each other from the case where they interact with the adversary. The authors of [2] addressed this issue by proposing a new model based on *split functionalities* which guarantees that this attack is the only one available to the adversary.

The split functionality is a generic construction based upon an ideal functionality: Its description can be found in Figure 5. In the initialization stage, the adversary $\mathcal{A}$ adaptively chooses disjoint subsets of the honest parties (with a unique session identifier that is fixed for the duration of the protocol). During the computation, each subset H activates a separate instance of the functionality $\mathcal{F}$. All these functionality instances are independent: The executions of the protocol for each subset H can only be related in the way $\mathcal{A}$ chooses the inputs of the players it controls. The parties $P_i \in H$ provide their own inputs and receive their own outputs, whereas $\mathcal{A}$ plays the role of all the parties $P_j \notin H$.

Note that the use of these split functionalities already allows the adversary to try some passwords for users by choosing subgroups of size 1 and trying a password for each of them while impersonating the other players. They are thus enough to model on-line dictionary attacks. In [1], additional TestPwd queries were available to the adversary, thus allowing additional password trials. In this paper, we limit the adversary against the ideal functionality (i.e. the simulator), to the unavoidable on-line dictionary attack but in the strict sense, and thus without any additional TestPwd queries. This means that we give less power to the simulator. Both the constructions in [1] and ours do not need them in the security proofs, which means that a stronger security level is reached.

In the sequel, as we describe our two general functionalities $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ and $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$ (the complete descriptions can be found in Figures 3 and 4), one has to keep in mind that an attacker controlling the communication channels can always choose to view them as the split functionalities $s\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ and $s\mathcal{F}_{\mathsf{pwDistPrivateComp}}$, which implicitly consist of multiple instances of $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ and $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$ for non-overlapping subsets of the original players. Furthermore, one cannot prevent $\mathcal{A}$ from keeping some flows, which will never arrive. This is modelled in our functionalities by a bit b, which specifies whether the flow is really sent or not.

The Players and the Group Leader. We denote by n the number of users involved in a given execution of the protocol. All the computation is done for the benefit of only one of them, denoted as the *group leader*. The role of all the other ones, the *players*, is to help it in its use of the group's virtual key. A group is thus formed arbitrarily and is defined by its composition, which cannot be changed: a leader, which is the only one to receive the result of a private computation in the end, and a (ordered or not, according to the secret key computation from the passwords) set of players to assist it.

The Aim of the Functionalities. The functionalities are intended to capture distributed-password protocols for (the key-generation and private-key operation of) an arbitrary public-key primitive, but taking into consideration the unavoidable on-line dictionary attacks. More precisely, the aim of the distributed key generation functionality $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ (Figure 3, page 17) is to provide a public key to the users, computed according to their passwords with respect to a function PublicKeyGen given as parameter. Moreover, it ensures that the group leader never receives an incorrect key in the end, whatever the adversary does.

In the distributed private computation functionality $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$ (Figure 4, page 18), the aim is to perform a private computation for the sole benefit of the group leader, which is responsible for the correctness of the computation; in addition, it is the only user to receive the end result. This functionality will thus compute a function of some supplied input in, depending on a set of passwords that must define a secret key corresponding to a given public key. More precisely, it will be able to check the compatibility of the passwords with the public key thanks to a verification function PublicKeyVer, and if it is correct it will then compute the secret key sk from the passwords with the help of a function SecretKeyGen, and from there evaluate PrivateComp(sk, in) and give the result to the leader. The function PrivateComp could be the decryption function Dec of a public-key encryption scheme, or the signing function Sign in a signature scheme, or the identity-based key extraction function Extract in an IBE system.

Note that SecretKeyGen and PublicKeyVer are naturally related to the function PublicKeyGen called by the former functionality. In all generality, unless SecretKeyGen and PublicKeyGen are both assumed to be deterministic, we need the predicate PublicKeyVer in order to verify that a public key is "correct" without necessarily being "equal" (to some canonical public key). Also note that the function SecretKeyGen is not assumed to be injective, lest it unduly restrict the number of users and the total size of their passwords. The distributed computations should not reveal more information than the non-distributed ones, and thus the ideal functionalities can make use of these functions as black-boxes.

The Functionalities. We only recall here the main points of the functionalities, referring the interested reader to [1] for details. But, importantly, as in [9], the functionalities are not in charge of providing the passwords to the participants. The passwords are chosen by the environment which then hands them to the parties as inputs. This guarantees security even in the case where an honest user executes the protocol with an incorrect password: This models, for instance, the case where a user mistypes its password. It also implies that the security is preserved for all password distributions (not necessarily the uniform one) and in all situations where related passwords are used in different protocols.

The private-computation functionality fails directly at the end of the initialization phase if the users do not share the same (public) inputs. In principle, after the initialization stage (the NewSession queries) is over, the eligible users are ready to receive the result. However the functionality waits for the adversary $\mathcal{S}$ to send a compute message before proceeding. This allows $\mathcal{S}$ to decide the exact moment when the result should be sent to the users and, in particular, it allows $\mathcal{S}$ to choose the exact moment when corruptions should occur (for instance $\mathcal{S}$ may decide to corrupt some party P_i before the result is sent but after P_i decided to participate to a given session of the protocol; see [15]). Also, although in the key generation functionality all users are normally eligible to receive the public key, in the private computation functionality it is important that only the group leader receives the output (though he may choose to reveal it afterwards to others, outside of the protocol, depending on the application). In both cases, after the result is computed, $\mathcal{S}$ can choose whether the group leader indeed receives it. If delivery is denied ($\mathsf{b} = 0$), then nobody gets it, and it is as if it was never computed. Otherwise, in the first functionality, the other players may be allowed to receive it too, according to a schedule chosen by $\mathcal{S}$.

Note that given the public key, if the adversary knows/controls sufficiently many passwords so that the combined entropy of the remaining passwords is low enough, he will be able to recover these remaining passwords by brute force attack. This is unavoidable and has nothing to do with the fact that the system is distributed: off-line attacks are always possible in principle in public-key systems, and become feasible as soon as a sufficient portion of the private key is known.

3 Notations and Building Blocks

The authors of [1] propose a protocol that deals with a particular case of unauthenticated distributed private computation [2], as captured by their functionalities recalled in the former section. Informally,

assuming s to be a secret key, the aim of the protocol is to compute a value c^s given an element c of the group. They claim that this computation can be used to perform distributed BLS signatures [7], ElGamal decryptions [11], linear decryptions [5], and BF or BB1 identity-based key extraction [6, 4] but they only focus on ElGamal decryptions, relying on the DDH assumption.

Here, we show how to really achieve such results, by constructing a protocol relying on the Decision Linear assumption [5] for compatibility with bilinear groups. This protocol will easily enable "password-based" Boneh-Franklin IBE scheme [6]. In the following section, we show how to modify the protocol to obtain "password-based" Boneh-Boyen (BB_1) IBE scheme [4] and linear decryptions [5].

Notations. Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p and g_3 a generator of $\mathbb{G}$. The linear encryption works as follows: The private key is a pair of scalars, $\mathsf{sk}_{\mathsf{lin}} = (x_1, x_2)$, and the public key, $\mathsf{pk}_{\mathsf{lin}} = (g_1, g_2, g_3)$, where $g_1 = {g_3}^{1/x_1}$, $g_2 = {g_3}^{1/x_2}$. In order to encrypt $M \in \mathbb{G}$, one chooses $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$, and the ciphertext consists of $C = \mathcal{E}_{\mathsf{pk}_{\mathsf{lin}}}(M; r_1, r_2) = (C_1, C_2, C_3) = ({g_1}^{r_1}, {g_2}^{r_2}, M{g_3}^{r_1+r_2})$. The decryption process consists of $M = \mathcal{D}_{\mathsf{pk}_{\mathsf{lin}}}(C) = C_3/({C_1}^{x_1}{C_2}^{x_2})$.

This encryption scheme is secure under the Decisional Linear (DLin) assumption, first presented in [5] and stated here for completeness: For random $x, y, r, s \in \mathbb{Z}_p^*$ and $(g, f = g^x, h = g^y, f^r, h^s) \in \mathbb{G}^5$, it is computationally intractable given g^d to distinguish between the case where $d = r + s$ or d is random. More precisely, a triple (f^r, h^s, g^d) is named a *linear triple* in basis (f, h, g) if $d = r + s$. We also consider a one-time signature scheme consisting of the three algorithms $(\mathsf{SKG}, \mathsf{Sign}, \mathsf{Ver})$.

Passwords, Public Key and Private Key. Each user P_i owns a privately selected password pw_i, to act as the i-th share of the secret key sk (see below). For convenience, we write $\mathsf{pw}_i = \mathsf{pw}_{i,1} \dots \mathsf{pw}_{i,\ell} \in \{0, \dots, 2^\ell - 1\}$, i.e., we further divide each password pw_i into ℓ bits $\mathsf{pw}_{i,j}$, where $p < 2^\ell$ (p is the order of the group $\mathbb{G}$). Notice that although we allow full-size passwords of up to ℓ bits (the size of p), users are of course permitted to choose shorter passwords.

The authors of [1] discussed the use of such passwords to combine properly into a private key sk: the combination should be reproducible, it should allow to recover either of the passwords from the key and the other passwords, and it should preserve the joint entropy of the set of paswords. They also discussed possible cancellation or aliasing effects of the passwords. The preferable solution is to do standard pre-processing using hashing, i.e. that each user independently transforms his or her true password pw_i^* into an effective password pw_i by applying a suitable extractor $\mathsf{pw}_i = \mathsf{H}(i, \mathsf{pw}_i^*, Z_i)$ where Z_i is any relevant public information. We can then safely take $\mathsf{sk} = \sum_i \mathsf{pw}_i$ and be assured that the entropy of sk will closely match the joint entropy of the vector $(\mathsf{pw}_1^*, \dots, \mathsf{pw}_n^*)$.

The discrete-log-based key pair $(\mathsf{sk}, \mathsf{pk} = g^{\mathsf{sk}})$ is then defined as follows:

$$\mathsf{sk} = \mathsf{SecretKeyGen}(\mathsf{pw}_1, \dots, \mathsf{pw}_n) \stackrel{\mathsf{def}}{=} \sum_{i=1}^n \mathsf{pw}_i$$
$$\mathsf{pk} = \mathsf{PublicKeyGen}(\mathsf{pw}_1, \dots, \mathsf{pw}_n) \stackrel{\mathsf{def}}{=} g^{\sum \mathsf{pw}_i}$$

The password/public-key verification function is then

$$\mathsf{PublicKeyVer}(\mathsf{pw}_1, \dots, \mathsf{pw}_n, \mathsf{pk}) \stackrel{\mathsf{def}}{=} \left(\mathsf{pk} \stackrel{?}{=} g^{\sum \mathsf{pw}_i}\right).$$

In the following, we focus on a specific format for the $\mathsf{PrivateComp}$ function, defined by $(\mathsf{sk}, c) \mapsto m = c^{\mathsf{sk}}$. We show how to perform it in a distributed way, and how to use if for decryption processes, and private key extraction in IBE.

Building Blocks.

Extractable Homomorphic Commitments. As in [1], the first step of our distributed decryption protocol is for each user to commit to his password (the details are given in the following section). The commitment needs to be extractable, homomorphic, and compatible with the shape of the public key. Generally speaking, one needs a commitment $\mathsf{Commit}(\mathsf{pw}, R)$ that is additively homomorphic on pw and with certain properties on R. Instead of ElGamal's scheme [11] used in [1], we focus here on linear commitments $\mathsf{Commit}_g(\mathsf{pw}, r, s) = (U_1{}^{\mathsf{pw}}{g_1}^r, U_2{}^{\mathsf{pw}}{g_2}^s, g^{\mathsf{pw}}{g_3}^{r+s})$, where $(U_1, U_2, U_3 = g)$ is *not* a linear triple in basis (g_1, g_2, g_3) in order to provide extractability, or encryptions $\mathsf{Encrypt}_g(\mathsf{pw}, r, s) = ({g_1}^r, {g_2}^s, g^{\mathsf{pw}}{g_3}^{r+s})$ (here, g_1, g_2 and g_3 are defined as before and g is a generator of $\mathbb{G}$). In both cases, the

hiding property or the semantic security rely on the DLin assumption. Extractability is possible granted the private/decryption key (x_1, x_2), such that $g_3 = {g_1}^{x_1} = {g_2}^{x_2}$, and recalling that the users commit to bits. Denoting by (c_1, c_2, c_3) the commitment, it is thus enough to check that $c_3/({c_1}^{x_1}{c_2}^{x_2}) = 1$ or $(c_3/g)/((c_1/U_1)^{x_1}(c_2/U_2)^{x_2}) = 1$.

Proofs of Membership. For the robustness and soundness of the protocols, we need some proofs of honest computations. We use witness-indistinguishable and SSNIZK proofs/arguments. The difficulty consists in designing such *simulation-sound* proofs without random oracles: they are described in Section 6. Along these lines, we use the following kinds of *non-interactive* proofs:

- $\mathsf{CDH}(g, G, h, H)$, to prove that (g, G, h, H) lies in the CDH language: there exists a common exponent x such that $G = g^x$ and $H = h^x$. Granted pairing-friendly groups, this can be easily done by simple pairing computations;
- $\mathsf{WIProofBit}(C)$, to prove that the commitment or the ciphertext C contains a bit. We will use a WI proof from [13], which basically proves that either C or C divided by the basis is a linear 3-tuple;
- $\mathsf{SSNIZKEq}_{g,c}(C_1, C_2)$, to prove that the ciphertexts/commitments C_1 and C_2 contain the same value, possibly in the different bases g and c, that is, C_1 encrypts/commits to g^a and C_2 encrypts/commits to c^a, with the same a. We use a SSNIZK argument, following the overall approach by Groth [12] to obtain simulation soundness, but using the Groth-Sahai proof system [14] for efficiency (see Section 6 – the proof is omitted, but very similar to [12]).

4 Description of the Protocols

The Distributed Key Generation Protocol. This protocol is described in Figure 1 and realizes the functionality $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$. All the users are provided with a password pw_i and want to obtain a public key pk. One of them is the leader of the group, denoted by P_1, and the others are $P_2, \ldots, P_n$.

The protocol starts with a round of commitments of these passwords. Each user sends a commitment C_i of pw_i (divided into ℓ blocks $\mathsf{pw}_{1,1}, \ldots, \mathsf{pw}_{i,\ell}$ of length L — here, $L = 1$): it computes $C_{i,j} = (C_{i,j}^{(1)}, C_{i,j}^{(2)}, C_{i,j}^{(3)}) = (U_1{}^{\mathsf{pw}_{i,j}} g_1{}^{r_{i,j}}, U_2{}^{\mathsf{pw}_{i,j}} g_2{}^{s_{i,j}}, g^{\mathsf{pw}_{i,j}} g_3{}^{r_{i,j}+s_{i,j}})$ for $j = 1, \ldots, \ell$ and random values $r_{i,j}$ and $s_{i,j}$, and publishes $\mathbf{C_i} = (C_{i,1}, \ldots, C_{i,\ell})$, with a set of proofs $\mathsf{WIProofBit}(C_{i,j})$ that each commitment indeed commits to an L-bit block. As we see in the proof (see Appendix B), this commitment needs to be extractable so that the simulator is able to recover the passwords used by the adversary, which is the reason why we segmented all the passwords and make commitments of bits, along with a $\mathsf{WIProofBit}$ that the committed value is actually a bit. Each user also runs the signature key generation algorithm to obtain a signature key SK_i and a verification key VK_i. The users will be split according to the values received in this first flow (i.e. the commitments, the proofs and the verification keys), as we see in the second flow where they send a signature of all they have received up to this point. Thus, the protocol cannot continue past this point if some players do not share the same values as the others (i.e. one of the signatures σ_i will be rejected later on and at least a user will abort).

Once this first step is done, the users commit again to their passwords (by encrypting them, for efficiency reasons), but this time in a single block: $C'_i = (C_i'^{(1)}, C_i'^{(2)}, C_i'^{(3)}) = (g_1{}^{t_i}, g_2{}^{u_i}, g^{\mathsf{pw}_i} g_3{}^{t_i+u_i})$ (with random values t_i and u_i) and publish it along with a SSNIZK proof that the passwords committed are the same in the two commitments: $\mathsf{SSNIZKEq}_{g,g}(C_i, C'_i)$, C_i roughly being the product of the $C_{i,j}$, i.e. a commitment of pw_i. The new encryptions C'_i will be the ones used in the rest of the protocol. They need not be segmented (since we will not extract anything from them, but just make computations on encrypted values), but we ask the users to prove that they are compatible with the former commitments.

Each user P_i computes $H = \mathcal{H}(\mathbf{C_1}, \ldots, \mathbf{C_n})$, and sends a signature of the values that identifies this execution, under an ephemeral one-time signature key, to avoid malleability and replay from previous sessions: $\sigma_i = \mathsf{Sign}(H; \mathsf{SK}_i)$. This allows the protocol to realize the split functionality by ensuring that

everybody has received the same values in the first round (more precisely, the players have been split according to what they received in the first round, so that we can assume that they have all received the same values). Note that the protocol will fail if the adversary drops or modifies a flow received by a user, even if everything was correct. This situation is modeled by the bit b of the key delivery queries in the functionality, for when everything goes well but some of the players do not obtain the result.

The need for an additional extractable commitment C_i of g^{pw_i} (and a proof that the password used is the same, and that everybody received the same value) is a requirement of the UC model, as in [9]. Indeed, we show later on that $\mathcal{S}$ needs to be able to simulate everything without knowing any passwords: Thus, he recovers the passwords by extracting them from the commitments $\mathbf{C_i}$ made by the adversary in the first round, enabling him to adjust his own values before the subsequent encryptions C'_i, so that all the passwords are compatible with the public key (if they should be in the situation at hand).

After these rounds of commitments/encryptions, the players check the signatures and abort if one of them is not valid. A computation step then allows them to compute the public key. Note that everything has become publicly verifiable.

Computation starts from the ciphertexts C'_i, and involves two "blinding rings" to raise sequentially the values $\prod_i C_i'^{(3)} = g^{\sum_i \mathsf{pw}_i} g_3^{\sum_i (t_i+u_i)}$, g_1, g_2 and g_3 to some distributed random exponent $\alpha = \sum_i \alpha_i$. The players then broadcast ${g_3}^{\alpha(t_i+u_i)}$ (the values g_1 and g_2 are only here to check the consistency of the values t_i and u_i and avoid cheating), leaving every player able to compute $g^{\alpha \sum_i \mathsf{pw}_i}$. A final "unblinding" allows for the recovery of $g^{\sum_i \mathsf{pw}_i} = \mathsf{pk}$. We stress that every user is able to check the validity of this computation (at each step, it checks the CDH values to ensure that the same exponent was used each time): A dishonest execution cannot continue without an honest user becoming aware of it (and aborting). Note however that an honest execution can also be stopped by a user if the adversary modifies a flow, as reflected by the bit b in the functionality.

The Distributed Private Computation Protocol. This protocol is presented in Figure 2 and realizes $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$. Here, in addition to their passwords, the users are also provided a public key pk and a group element $c \in \mathbb{G}$. For this given $c \in \mathbb{G}$, the leader wants to obtain $m = c^{\mathsf{sk}}$. A big difference with the previous protocol is that this result will be private to the leader. But before computing it, everybody wants to be sure that all the users are honest, or at least that the combination of the passwords is compatible with the public key.

This verification step is exactly the same as the computation step in the previous protocol. The protocol starts by verifying that they will be able to perform this computation, and thus that they indeed know a representation of the secret key into shares. Each user sends a commitment $\mathbf{C_i} = \{C_{i,j}\}_j$ of its password as before, and the associated set of $\mathsf{WIProofBit}(C_{i,j})$.

As in the former protocol, once this first step (which enables the users to be split into subgroups according to what values they have received) is done, the users commit again to their passwords in the value C'_i, which will be the ones used in the rest of the protocol, and also send a signature which enables them to check that they share the same public key pk, the same group element c, and have received the same values in the first round. It thus avoids situations in which a group leader with an incorrect key obtains a correct private computation result, contrary to the ideal functionality. The protocol will thus fail if all these values are not the same to everyone, which is the result required by the functionality.

Next, the users make yet another encryption A_i of their passwords, but this time they do a linear encryption of pw_i in base c instead of in base g (in the above C'_i ciphertext): $A_i = \mathsf{Encrypt}_c(\mathsf{pw}_i, v_i, w_i) = ({g_1}^{v_i}, {g_2}^{w_i}, c^{\mathsf{pw}_i} {g_3}^{v_i+w_i})$. The ciphertexts C'_i will be used to check the possibility of the private computation (i.e. that the passwords are consistent with the public key $\mathsf{pk} = g^{\mathsf{sk}}$), whereas the ciphertexts A_i will be used to actually compute the expected result c^{sk}, hence the two different bases g and c in C'_i and A_i, respectively. All the users send these last two ciphertexts to everybody, along with a SSNIZK argument that the same password was used each time: $\Pi_i^2 = \mathsf{SSNIZKEq}_{g,c}(C'_i, A_i)$.

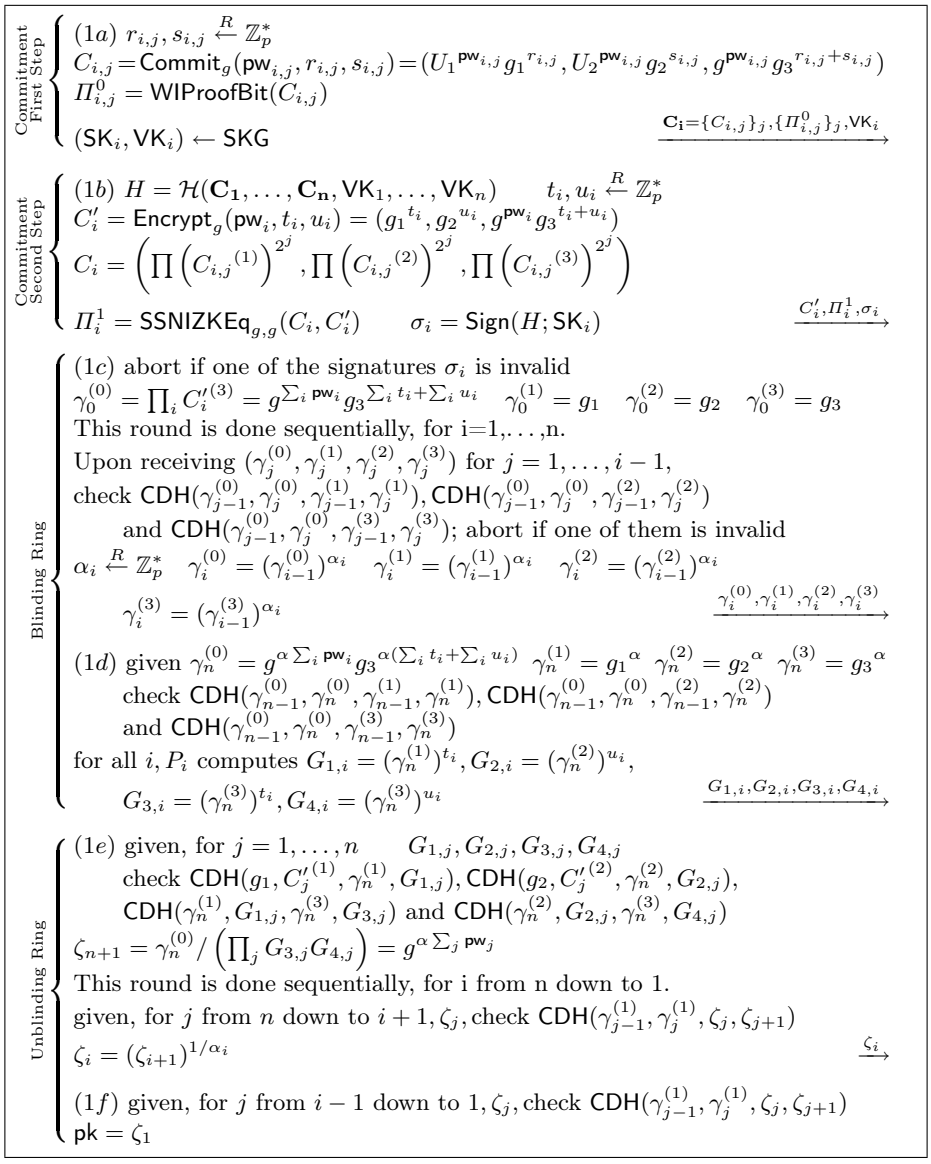

Fig. 1. Individual steps of the distributed key generation protocol

After these rounds of commitments/encryptions, a verification step allows for all the players to check whether the public key and the passwords are compatible. Note that at this point, everything has become publicly verifiable so that the group leader will not be able to cheat and make the other players believe that everything is correct when it is not. Verification starts from the ciphertexts C'_i, and involves a blinding and an unblinding ring as described above. This ends with a decision by the group leader on whether to abort the protocol (when the passwords are incompatible) or go on to the computation step. Every user is able to check the validity of the group leader's decision, as in the former protocol.

If the group leader decides to go on, the players assist it in the computation of c^{sk}, again with the help of a blinding and an unblinding rings, starting from the ciphertexts A_i. However, note that this time, the group leader does not reveal the values $G'_{1,1} = (\delta_n^{(1)})^{v_1}$, $G'_{2,1} = (\delta_n^{(2)})^{w_1}$, $G'_{3,1} = (\delta_n^{(3)})^{v_1}$ and $G'_{4,1} = (\delta_n^{(3)})^{w_1}$ at the end of the blinding ring, but it is the only one able to compute $c^{\beta \sum_j \mathsf{pw}_j}$. Instead of revealing it to the others, it chooses at random an exponent $x \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and broadcasts the

value $c^{\beta x \sum_j \mathsf{pw}_j}$. The unblinding ring then takes place as before, leading to a public value $c^{\beta_1 x \sum_j \mathsf{pw}_j}$ that the environment cannot distinguish from random thanks to the random exponent x. Furthermore, the whole process is robust, which means that nobody can make the decryption result become incorrect. Except of course the group leader itself who broadcasts any value it wants as ζ'_{n+1}, without having to prove anything. But this does not help it to obtain a computation which it could not do alone, except the result c^{sk}.

Note that if at some point a user fails to send its value (denial of service attack) or if the adversary modifies a flow (man-in-the-middle attack), the protocol will fail. In the ideal world this means that the simulator makes a computation delivery query to the functionality with a bit b set to zero. Because of the public verifications of the CDH values, in these blinding/unblinding rounds exactly the same sequence of passwords as in the first rounds has to be used by the players. This necessarily implies compatibility with the public key, but may be an even stronger condition.

As a side note, observe that all the blinding rings in the verification and computation steps could be made concurrent instead of sequential, to simplify the protocol. Notice however that the final unblinding ring of c^{sk} in the computation step should only be carried out after the public key and the committed passwords are known to be compatible, and the passwords to be the same in both sequences of commitments/encryptions, i.e. after the verification step succeeded.

All the witness-indistinguishable and SSNIZK proofs and arguments will be described in Section 6. We show in Appendix B that we can *efficiently* simulate these computations without the knowledge of the pw_i's, so that they do not reveal anything more about the pw_i's than pk already does. More precisely, we show that such computations are indistinguishable to $\mathcal{A}$ under the DLin assumption.

Security Theorems. Assuming that the proofs of membership $\mathsf{WIProofBit}$ and $\mathsf{SSNIZKEq}$ are instantiated as described in Section 6 (relying on the CDH), we have the following results, provided that DLin is infeasible in $\mathbb{G}$ and $\mathcal{H}$ is collision-resistant. The proofs of these theorems can be found in Appendix B.

Theorem 1 *Let $\widehat{\mathcal{F}}_{\mathsf{pwDistPublicKeyGen}}$ be the concurrent multi-session extension of $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$. The distributed key generation protocol in Figure 1 securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwDistPublicKeyGen}}$ for ElGamal key generation, in the CRS model, in the presence of static adversaries.*

Theorem 2 *Let $\widehat{\mathcal{F}}_{\mathsf{pwDistPrivateComp}}$ be the concurrent multi-session extension of $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$. The distributed decryption protocol in Figure 2 securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwDistPrivateComp}}$ for ElGamal decryption, in the CRS model, in the presence of static adversaries.*

As stated above, our protocols are only proven secure against static adversaries. Unlike adaptive ones, static adversaries are only allowed to corrupt protocol participants prior to the beginning of the protocol execution.

5 Extensions of the Protocols

Boneh-Franklin IBE Scheme [6]. We need to compute $d_{\mathsf{id}} = H(\mathsf{id})^{\mathsf{sk}}$ where $H(\mathsf{id})$ is a public hash of a user's identity. This is analogous to c^{sk}, and thus our protocol works as is.

Boneh-Boyen ($\mathbf{BB_1}$) IBE Scheme [4]. Here, d_{id} is randomized and of the form $({h_0}^{\mathsf{sk}}(h_1^{\mathsf{id}}h_2)^r, {h_3}^r)$. Since $({h_0}^{\mathsf{sk}})$ is a private value, the protocol can be adapted as follows: 1) In the commitment steps, the user also commits (once) in $(2a)$ to a value r_i, which will be its share of r. 2) Up to $(2f)$, everything works as before in order to check pk (there is no need to check r, constructed on the fly). 3) The blinding rings are made in parallel, one for $({h_0}^{\mathsf{sk}})^\beta$, one for $((h_1^{\mathsf{id}}h_2)^r)^\beta$, and one for $({h_3}^r)^\beta$, the CDH being checked to ensure that the same r and β_i are used each time. 4) The players obtain $({h_0}^{\mathsf{sk}}(h_1^{\mathsf{id}}h_2)^r)^\beta$ and the unblinding ring is made globally for this value. An unblinding ring is also done for $({h_3}^r)^\beta$, with the same verification for the exponents β_i.

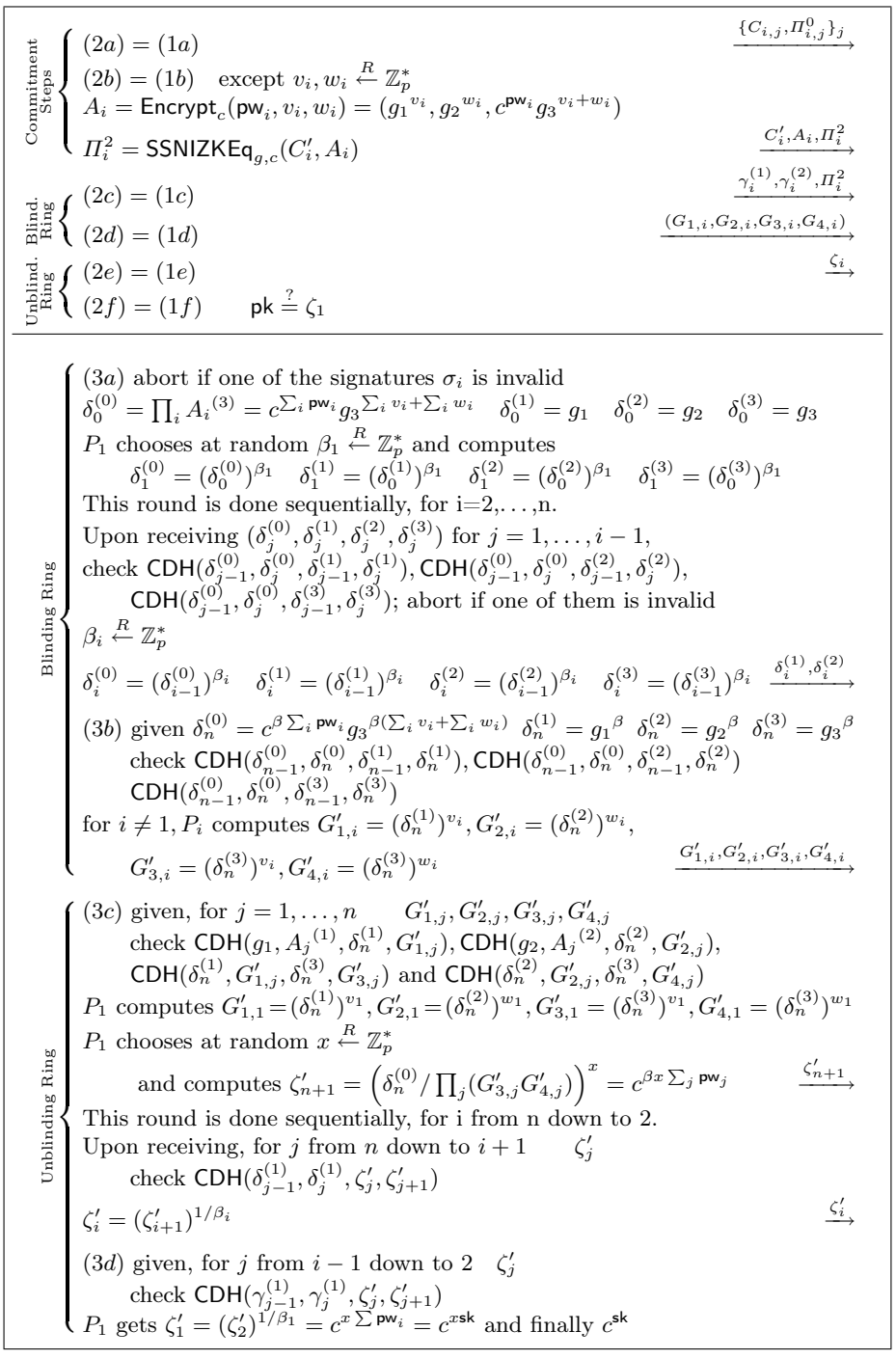

Fig. 2. Individual steps of the distributed decryption protocol

Linear Decryptions [5]. Let $(f = g^{1/x}, g, h = g^{1/y})$ be the public key of a linear encryption scheme, (x, y) being the private key. Assuming $z = y/x$, these keys can be seen as $\mathsf{pk} = (h^z, h^y, h)$ and $\mathsf{sk} = (y, z)$. Using these notations,

$$c = \mathcal{E}_{\mathsf{pk}}(m; r) = (c_1, c_2, c_3) = (f^r, h^s, mg^{r+s})$$
$$m = \mathcal{D}_{\mathsf{sk}}(c) = c_3(c_1{}^x c_2{}^y)^{-1} = mg^{r+s}g^{-r}g^{-s}$$

In the first protocol, the players need to use two passwords z_i and y_i to create the public key pk. In the second one, the commitment steps are doubled to commit to both z_i and y_i. As soon as pk is checked, the blinding rings are made separately, one for $(c_1{}^x)^\beta$ and one for $(c_2{}^y)^\beta$. The players obtain $(c_1{}^x c_2{}^y)^\beta$ and the unblinding ring can be made globally for this value. In both rings, the CDH is checked to ensure that the same β_i is used each time.

6 Employed Proof Systems

6.1 GOS WI Proof of Commitments Being to Bits

Let $(g_1, g_2, g_3) \in \mathbb{G}^3$ be a "basis" and let $(U_1, U_2, g) \in \mathbb{G}^3$ be a commitment key (which is in general non-linear w.r.t. (g_1, g_2, g_3), but for simulation purposes it will be linear). Let $C = (U_1^x g_1^r, U_2^x g_2^s, g^x g_3^{r+s})$ be a commitment to x using randomness (r, s). Groth *et al.* [13] construct a WI proof system to show that one of two triples is linear. Applying it to (C_1, C_2, C_3) and $(C_1U_1^{-1}, C_2U_2^{-1}, C_3g^{-1})$ yields a proof that $x \in \{0, 1\}$, thus implements $\mathsf{WIProofBit}$, in an efficient way and without random oracles. A proof consists of 6 group elements and is verified by 6 equations using 18 pairings in total. To simulate, one chooses (U_1, U_2, g) to be linear, thus the commitment C is linear whether x is 0 or 1. Witness indistinguishability of the proof guarantees that the case where the commitments and the proofs where constructed for $x = 0$ is perfectly indistinguishable from the case $x = 1$.

We provide below more details how to implement $\mathsf{WIProofBit}$: choose $t \leftarrow \mathbb{Z}_p$ and set

$$\pi_{1,1} := U_1^{(2x-1)r} g_1^{r^2} \qquad \pi_{1,2} = U_2^{(2x-1)r} g_2^{t+rs} \qquad \pi_{1,3} := g^{-(2x-1)r} g_3^{-t-(r+s)r}$$
$$\pi_{2,1} := U_1^{(2x-1)s} g_1^{-t+rs} \qquad \pi_{2,2} = U_2^{(2x-1)s} g_2^{s^2} \qquad \pi_{2,3} := g^{-(2x-1)s} g_3^{t-(r+s)s}$$

A proof $\pi = (\pi_{1,1}, \ldots, \pi_{2,3})$ for a commitment (C_1, C_2, C_3) is verified by checking:

$$e(g_1, \pi_{1,1}) = e(C_1, C_1U_1^{-1}) \qquad e(g_1, \pi_{1,2})e(g_2, \pi_{2,1}) = e(C_1, C_2U_2^{-1})e(C_2, C_1U_1^{-1})$$
$$e(g_2, \pi_{2,2}) = e(C_2, C_2U_2^{-1}) \qquad e(g_1, \pi_{1,3})^{-1}e(g_3, \pi_{1,1}\pi_{2,1}) = e(C_1, C_3g^{-1})e(C_3, C_1U_1^{-1})$$
$$e(g_3, \pi_{1,3}\pi_{2,3})^{-1} = e(C_3, C_3g^{-1}) \qquad e(g_2, \pi_{2,3})^{-1}e(g_3, \pi_{1.2}\pi_{2,2}) = e(C_2, C_3g^{-1})e(C_3, C_2U_2^{-1})$$

Correctness. We show correctness of two exemplary equations. The first one: $e(g_1, \pi_{1,1}) = e(g_1^r, U_1^{2x-1}g_1^r)$ which is $e(C_1, C_1U_1^{-1})$ when $x = 0$ and $e(C_1U_1^{-1}, C_1)$ when $x = 1$. The fourth equation:

$$\begin{aligned} e(g_1, \pi_{1,3})^{-1}e(g_3, \pi_{1,1}\pi_{2,1}) &= e(g_1, g^{(2x-1)r} g_3^{t+(r+s)r})e(g_3, U_1^{(2x-1)r} g_1^{r^2} U_1^{(2x-1)s} g_1^{-t+rs}) \\ &= e(g_1^r, g^{2x-1})e(g_1, g_3^t)e(g_1^r, g_3^{r+s})e(g_3^r, U_1^{2x-1})e(g_3^r, g_1^r)e(g_3^s, U_1^{2x-1})e(g_3, g_1^{-t})e(g_3^s, g_1^r) \\ &= e(g_1^r, g^{2x-1}g_3^{r+s})e(g_3^{r+s}, U_1^{2x-1})e(g_3^{r+s}, g_1^r) = e(g_1^r, g^{2x-1}g_3^{r+s})e(g_3^{r+s}, U_1^{2x-1}g_1^r) \\ &= e(C_1, C_3g^{-1})e(C_3, C_1U_1^{-1}) \end{aligned}$$

The remaining equations are shown to hold analogously.

Soundness. Let (U_1, U_2, g) be a binding commitment key, let $C = (U_1^x g_1^r, U_2^x g_2^s, g^x g_3^{r+s})$ (note that for any C, there exist such x, r, s), and let π be a proof for C that passes verification.

Define $\rho := \log_{g_1} U_1$, $\sigma := \log_{g_2} U_2$, $\tau := \log_{g_3} g$, define $m_{i,j} := \log_{g_j} \pi_{i,j}$ Then from the verification relation we get

$$\begin{aligned}
m_{1,1} &= (\rho x + r)(\rho(x-1) + r) \\
m_{1,2} + m_{2,1} &= (\rho x + r)(\sigma(x-1) + s) + (\sigma x + s)(\rho(x-1) + r) \\
m_{2,2} &= (\sigma x + s)(\sigma(x-1) + s) \\
-m_{1,3} + m_{1,1} + m_{2,1} &= (\rho x + r)(\tau(x-1) + r + s) + (\tau x + r + s)(\rho(x-1) + r) \\
-m_{1,3} - m_{2,3} &= (\tau x + r + s)(\tau(x-1) + r + s) \\
-m_{2,3} + m_{1,2} + m_{2,2} &= (\sigma x + s)(\tau(x-1) + r + s) + (\tau x + r + s)(\sigma(x-1) + s)
\end{aligned}$$

If we add the first, second, third and fifth equation and substract the fourth and sixth, we get 0 on the left-hand side, whereas the right-hand side simplifies to $(\rho^2 + \sigma^2 + \tau^2 + 2\rho\sigma - 2\rho\tau - 2\sigma\tau)x(x-1)$. Thus $(\rho + \sigma - \tau)^2 x(x-1) = 0$, which implies $x = 0$ or $x = 1$, since for a binding commitment key (a non-linear triple), we have $\rho + \sigma \neq \tau$.

Witness Indistinguishability. If the commitment key is witness hiding, i.e., (U_1, U_2, g) is linear w.r.t. (g_1, g_2, g_3), then C is linear, be it computed for $x = 0$ or $x = 1$. Moreover, we show that the two cases entail the same proof.

Let $U_1 = g_1^{\rho}$, $U_2 = g_2^{\sigma}$, $g = g_3^{\rho+\sigma}$. Let the commitment and proof be produced for $x = 0$, i.e., $C = (g_1^r, g_2^s, g_3^{r+s})$ and

$$\begin{aligned}
\pi_{1,1} &= U_1^{-r} g_1^{r^2} & \pi_{1,2} &= U_2^{-r} g_2^{t+rs} & \pi_{1,3} &= g^r g_3^{-t-(r+s)r} \\
\pi_{2,1} &= U_1^{-s} g_1^{-t+rs} & \pi_{2,2} &= U_2^{-s} g_2^{s^2} & \pi_{2,3} &= g^s g_3^{t-(r+s)s}
\end{aligned} \tag{1}$$

However, C could also be the result of "committing" to 1 and using randomness $(r' = r - \rho, s' = s - \sigma)$, since $C = (g_1^r, g_2^s, g_3^{r+s}) = (U_1 g_1^{r'}, U_2 g_1^{s'}, g g_3^{r'+s'})$. We show that using witness $x = 1$ and (r', s'), and randomness $t' := t + \rho s - \sigma r$ for the proof generates the same proofs as in (1):

$$\begin{aligned}
\pi'_{1,1} &= U_1^{r'} g_1^{(r')^2} = g_1^{\rho(r-\rho)+(r^2-2\rho r+\rho^2)} = g_1^{-\rho r + r^2} = U_1^{-r} g_1^{r^2} = \pi_{1,1} \\
\pi'_{1,2} &= U_2^{r'} g_2^{t'+r's'} = g_2^{\sigma(r-\rho)+(t+\rho s-\sigma r)+(rs-\rho s-\sigma r+\rho\sigma)} = g_2^{-\sigma r + t + rs} = U_2^{-r} g_2^{t+rs} = \pi_{1,2} \\
\pi'_{1,3} &= g^{-r'} g_3^{-t'-(r'+s')r'} = g_3^{-(\rho+\sigma)(r-\rho)-(t+\rho s-\sigma r)-(r+s-\rho-\sigma)(r-\rho)} = g_3^{-t-r^2-sr+\sigma r+\rho r} = \pi_{1,3} \\
\pi'_{2,1} &= U_1^{s'} g_1^{-t'+r's'} = g_1^{\rho(s-\sigma)-(t+\rho s-\sigma r)+(rs-\rho s-\sigma r+\rho\sigma)} = g_1^{-t+rs-\rho s} = \pi_{2,1} \\
\pi'_{2,2} &= U_2^{s'} g_2^{(s')^2} = g_2^{\sigma(s-\sigma)+(s^2-2\sigma s+\sigma^2)} = g_2^{\sigma s + s^2} = \pi_{2,2} \\
\pi'_{2,3} &= g^{-s'} g_3^{t'-(r'+s')s'} = g_3^{-(\rho+\sigma)(s-\sigma)+(t+\rho s-\sigma r)-(r+s-\rho-\sigma)(s-\sigma)} = g_3^{t-rs-s^2+\rho s_\sigma s} = \pi_{2,3}
\end{aligned}$$

6.2 Simulation-Sound NIZK Arguments for Relations of Ciphertexts and Commitments

We construct two simulation-sound NIZK argument systems implementing the proof $\mathsf{SSNIZKEq}$. Given two ciphertexts, the first proves that the encrypted messages m_1 and m_2 are in CDH w.r.t. some fixed basis (c, d), i.e., $m_1 = c^{\mu}$ and $m_2 = d^{\mu}$ for some μ. The second SSNIZK proves that for a given linear commitment to x and a linear encryption of g^y it holds that $x = y$. We follow the overall approach by Groth [12] to obtain simulation soundness, but using the Groth-Sahai proof system [14] we get an efficient result: the proofs themselves are efficient, and we need not *encrypt* some of the witnesses in order to guarantee extractability, as the employed Groth-Sahai proofs are witness extractable.

Overview. We start with some intuition on how [12] constructs simulation-sound proofs for *satisfiability of a set of pairing product equations* (PPEs) $\{E_k\}_{k=1}^{K_E}$ (and later show how to express the

statements we want to prove this way). Let Σ_{ot} be a strong one-time signature scheme[1] and let Σ_{cma} be a signature scheme that is existentially unforgeable under chosen message attack (EUF-CMA), and whose signatures σ on a message M are verified by checking a set of PPEs over a verification key vk and M, denoted $\{V_k(\mathsf{vk}, M, \sigma)\}_{k=1}^{K_V}$.

The common reference string (CRS) of our argument system will contain a verification key vk for Σ_{cma} (whose corresponding signing key serves as simulation trapdoor). When making an argument, one first chooses a key pair $(\mathsf{vk}_{\text{ot}}, \mathsf{sk}_{\text{ot}})$ for Σ_{ot}, proves a statement and, at the end, adds a signature under vk_{ot} on the instance and the proof. The statement one actually proves is the following: to either know a witness satisfying Equations $\{E_k\}$ *or* to know a signature on vk_{ot} valid under vk. Groth [12] shows how to construct a new set of equations which is satisfiable iff $\{E_k\}$ or $\{V_k(\mathsf{vk}, \mathsf{vk}_{\text{ot}}, \cdot)\}$ are satisfiable. Moreover, knowing witnesses for either of them, one can compute witnesses of the new set of equations. Using the techniques of [14], one then commits to the witnesses and proves that the committed values satisfy the new PPEs in a witness-indistinguishable (WI) way.

To simulate an argument, after choosing a pair $(\mathsf{vk}_{\text{ot}}, \mathsf{sk}_{\text{ot}})$, one uses the trapdoor to produce a signature σ on vk_{ot} valid under vk and uses σ as a witness for $\{V_k(\mathsf{vk}, \mathsf{vk}_{\text{ot}}, \cdot)\}$. (It follows from WI of the Groth-Sahai proof that this is indistinguishable from using a witness for $\{E_k\}$.) Even after seeing many proofs of this kind, no adversary is able to produce one for a new false statement: Since it has to sign the instance and the argument at the end, it must choose a *new* pair $(\mathsf{vk}_{\text{ot}}^*, \mathsf{sk}_{\text{ot}}^*)$ (by one-time security of Σ_{ot}). Soundness of Groth-Sahai proofs imposes that to prove a false statement (meaning that the first clause of the disjunction is not satisfiable), it must use a witness for the second clause, thus know a signature on vk_{ot}. This however is infeasible by EUF-CMA of Σ_{cma} (since we can extract the witnesses and thus a forged signature). We start by instantiating the mentioned building blocks.

Building Blocks. The main motivation for our choices of instantiations of these blocks is that their security is implied by DLin only. We insist that by admitting more exotic assumptions, the efficiency of our proof system could be improved.

The Strong One-Time Signature Scheme Σ_{OT}. We pick the scheme described in [12] (but any other would equally do), since its security follows from the discrete-log assumption which is implied by DLin.

The Waters Signature Scheme. The signature scheme from [16] suits our purposes, it requires no additional assumption and—more importantly—signatures are verified by checking PPEs.

Setup. In a bilinear group $(p, \mathbb{G}, \mathbb{G}_T, e, g)$, define parameters $f \leftarrow \mathbb{G}^*$ and $\boldsymbol{h} := (h_0, h_1, \ldots, h_\ell) \leftarrow \mathbb{G}^{\ell+1}$. A secret key $x \leftarrow \mathbb{Z}_p$ defines a public key $X := g^x$. For ease of notation, define $\mathcal{W}(M) := h_0 \prod_{i=1}^{\ell} h_i^{M_i}$.

Signing. To sign a message $M \in \{0,1\}^\ell$, choose $r \leftarrow \mathbb{Z}_p$ and define a signature as $\sigma := (f^x \mathcal{W}(M)^r, g^{-r})$.

Verification. A signature $\sigma = (\sigma_1, \sigma_2)$ is accepted for message M iff

$$e(\sigma_1, g)\, e(\mathcal{W}(M), \sigma_2) = e(f, X) \tag{2}$$

Security. EUF-CMA follows from the computational Diffie-Hellman assumption which is implied by DLin.

The Groth-Sahai Proof System. Consider a set of *pairing product equations* $\{E_k\}_{k=1}^{K_E}$ on variables $\{X_i\}_{i=1}^{n}$ in $\mathbb{G}$ of the form

$$\prod_{i=1}^{n} e(A_{k,i}, X_i) \prod_{i=1}^{n} \prod_{j=1}^{n} e(X_i, X_j)^{\gamma_{k,i,j}} = T_k \tag{E_k}$$

for given $A_{k,i} \in \mathbb{G}$, $\gamma_{k,i,j} \in \mathbb{Z}_p$, and $T_k \in \mathbb{G}_T$. Groth and Sahai [14] build a non-interactive witness-indistinguishable proof of satisfiability of $\{E_k\}$ from which—given a trapdoor—can be extracted the witnesses X_i (we will use their instantiation with DLin): the common reference string is a (binding) key for linear commitments to group elements. The proof consists of commitments to each X_i and 9 elements

[1] A signature scheme is *strong one-time* if no adversary, after getting a signature σ on *one* message m of his choice, can produce a valid pair $(m^*, \sigma^*) \neq (m, \sigma)$.

of $\mathbb{G}$ per equation proving that it is satisfied by the committed values. By DLin, replacing the CRS by a *hiding* commitment key is indistinguishable. In this setting now every witness $\{X_i\}_{i=1}^n$ satisfying the equations generates the same distribution of proofs, which implies witness-indistinguishability of the proofs.

Moreover, we assume a collision-resistant hash function $\mathcal{H}$ that maps strings of elements of $\mathbb{G}$ to elements in $\mathbb{Z}_p$ which we identify with their bit-representation in $\{0,1\}^{\lceil \log p \rceil}$. Thus, when we say we sign a vector of group elements, we actually mean that we sign their hash values.

Equations for Proof of Plaintexts Being in CDH. Let $c, d \in \mathbb{G}$ be fixed and let (g_1, g_2, g_3) be a linear encryption key. Given two ciphertexts $C = (g_1^r, g_2^s, m_1 g_3^{r+s})$ and $D = (g_1^t, g_2^u, m_2 g_3^{t+u})$, we give a set of PPEs that are satisfiable by a witness a if and only if there exists $\mu \in \mathbb{Z}_p$ such that $m_1 = c^\mu$ and $m_2 = d^\mu$.

$$\begin{array}{lll} e(C_1, g_3) = e(g_1, a_1) & e(C_2, g_3) = e(g_2, a_2) & \\ e(D_1, g_3) = e(g_1, a_3) & e(D_2, g_3) = e(g_2, a_4) & e(C_3 a_1^{-1} a_2^{-1}, d) = e(c, D_3 a_3^{-1} a_4^{-1}) \end{array} \tag{3}$$

The witness satisfying them is $a := (g_3^r, g_3^s, g_3^t, g_3^u)$. The first four equations prove that the logarithms of the a_i's are those of C_1, C_2, D_1, D_2 w.r.t. their respective bases. Thus, $C_3 a_1^{-1} a_2^{-1} = m_1$ and $D_3 a_3^{-1} a_4^{-1} = m_2$ and the last equation shows that (m_1, m_2) is in CDH w.r.t. (c, d).

Disjunction of Equations. Following [12] (and optimizing since the pairings have variables in common), we define a set of equations which we can prove satisfiable if we have witnesses for either (3) or (2), i.e., if we either know a satisfying (3) or σ satisfying (2). We first introduce the following new variables:

$$\chi_1, \chi_2 \qquad \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 \qquad \psi_1, \psi_2, \psi_3$$

We define the following 15 equations expressing a disjunction of (3) and (2), therefore termed "$(3 \vee 2)$".

Equation for Disjunction:
$$e(g^{-1}\chi_1\chi_2, g) = 1$$

From (2):
$$e(\chi_2, \psi_1^{-1}\sigma_1) = 1 \qquad e(\chi_2, \psi_2^{-1}\mathcal{W}(M)) = 1 \qquad e(\chi_2, \psi_3^{-1} f) = 1$$
$$e(\psi_1, g)\, e(\psi_2, \sigma_2)\, e(\psi_3, X)^{-1} = 1$$

From (3):
$$\begin{array}{lll} e(\chi_1, \phi_1^{-1} g_1) = 1 & & e(\chi_1, \phi_2^{-1} g_2) = 1 \\ e(\chi_1, \phi_3^{-1} g_3) = 1 & e(\chi_1, \phi_4^{-1} c) = 1 & e(\chi_1, \phi_5^{-1} d) = 1 \\ e(C_1, \phi_3)\, e(\phi_1, a_1)^{-1} = 1 & & e(C_2, \phi_3)\, e(\phi_2, a_2) = 1 \\ e(D_1, \phi_3)\, e(\phi_1, a_3)^{-1} = 1 & & e(D_2, \phi_3)\, e(\phi_2, a_4) = 1 \end{array}$$
$$e(C_3 a_1^{-1} a_2^{-1}, \phi_5)\, e(\phi_4, D_3 a_3^{-1} a_4^{-1}) = 1$$

Completeness. To produce a proof we proceed as follows: If we have an assignment a for (3), we choose $\chi_1 := g$, $\chi_2 := 1$, satisfying thus the first equation. Moreover, set $\phi_1 := g_1$, $\phi_2 := g_2$, $\phi_3 := g_3$, $\phi_4 := c$, $\phi_5 := d$. Thus the equations of the block for (3) are satisfied, because a is a witness for (3). Since $\chi_2 = 1$, we can set $\psi_i := 1$ (for all i) as well, which satisfies the block for (2), no matter what value we set σ.

On the other hand, if we know a signature σ satisfying (2), we choose $\chi_1 := \phi_i := 1$ (for all i) and $\chi_2 := g$, $\psi_1 := \sigma_1$, $\psi_2 := \mathcal{W}(M)$, $\psi_3 := f$ and get a satisfying assignment for any choice of a.

Soundness. We show that if $(3 \vee 2)$ is satisfied then either a satisfies (3) or σ satisfies (2): From the first equation we have that either χ_1 or χ_2 must be non-trivial, which either confines the values of the ϕ_i's to (g_1, g_2, g_3, c, d) or those of the ψ_i's to $(\sigma_1, \mathcal{W}(M), f)$. Now this imposes that either a satisfies (3) (by the last five equations of the block for (3)) or σ satisfies (2) (by the last equation of the block for (2)).

Equations for Proof of Commitment and Ciphertext Containing the Same Value. Let (g_1, g_2, g_3) be a key for linear encryption, and let (U_1, U_2, g) be an associated commitment key. Let

$C = (U_1^x g_1^r, U_2^x g_2^s, g^x g_3^{r+s})$ be a commitment to x and $D = (g_1^v, g_2^w, g^y g_3^{v+w})$ be an encryption of g^y. We prove that $x = y$: the witness is $(a_1 = U_1^x, a_2 = U_2^x, a_3 = g^x, a_4 = g_3^r, a_5 = g_3^v)$ satisfying

$$\begin{array}{lll} e(a_1, U_2) = e(U_1, a_2) & e(C_1 a_1^{-1}, g_3) = e(g_1, a_4) & e(D_1, g_3) = e(g_1, a_5) \\ e(a_1, g) = e(U_1, a_3) & e(C_2 a_2^{-1}, g_3) = e(g_2, C_3 a_3^{-1} a_4^{-1}) & e(D_2, g_3) = e(g_2, D_3 a_3^{-1} a_5^{-1}) \end{array} \quad (4)$$

The equations in the first column show that $a_1 = U_1^z$, $a_2 = U_2^z$, $a_3 = g^z$ for some z, the second column proves that $(C_1 a_1^{-1}, C_2 a_2^{-1}, C_3 a_2^{-1})$ is linear (i.e., C commits to z) and the third that D is an encryption of $a_3 = g^z$.

Transformation. Transforming Equations (4) and (2) to a set $(4 \vee 2)$ analogously to the construction of $(3 \vee 2)$, we get a set of 16 equations we can prove satisfiable adding 10 new witnesses if either we have a witness for C being a commitment to some x and D an encryption of g^x, or we know a signature. (Associate the ϕ_i's to U_1, a_1, g_1, g_2 and g_3.)

Assembling the Pieces. We describe the SSNIZK proof system for "plaintexts in CDH". The one for "commitment and ciphertext contain the same value" is obtained by replacing $(3 \vee 2)$ by $(4 \vee 2)$.

Common Reference String. Generate a key pair $(\mathsf{vk}, \mathsf{sk})$ for Waters' signature scheme, and a CRS $\mathsf{crs}_{\mathrm{GS}}$ for the Groth-Sahai proof system. Let $\mathsf{crs} := (\mathsf{vk}, \mathsf{crs}_{\mathrm{GS}})$ and let the simulation trapdoor be sk.

Proof. Let $(C, D) \in \mathbb{G}^6$ be an instance and a a witness satisfying (3). Generate a key pair $(\mathsf{vk}_{\mathrm{ot}}, \mathsf{sk}_{\mathrm{ot}})$ for Σ_{ot}; using witness a, make a Groth-Sahai proof π_{GS} w.r.t. $\mathsf{crs}_{\mathrm{GS}}$ of satisfiability of $(3 \vee 2)$ with $M := \mathsf{vk}_{\mathrm{ot}}$; produce a signature σ_{ot} on $(C, D, \mathsf{vk}_{\mathrm{ot}}, \pi_{\mathrm{GS}})$ using $\mathsf{sk}_{\mathrm{ot}}$. The proof is $\pi := (\mathsf{vk}_{\mathrm{ot}}, \pi_{\mathrm{GS}}, \sigma_{\mathrm{ot}})$

Verification. Given π, verify σ_{ot} on $(C, D, \mathsf{vk}_{\mathrm{ot}}, \pi_{\mathrm{GS}})$ under $\mathsf{vk}_{\mathrm{ot}}$, and π_{GS} on the respective equations.

Simulation. Proceed as in Proof, but using sk produce σ on $\mathsf{vk}_{\mathrm{ot}}$ and use that as a witness for $(3 \vee 2)$.

Theorem 3 *Under the DLin assumption, the above is a simulation-sound NIZK argument for the encryptions of two linear ciphertexts forming a CDH-pair.*

Using the ideas given in the overview, the proof is analogous to that in [12] except that we do not require perfect soundness and that we use the extraction key for $\mathsf{crs}_{\mathrm{GS}}$ to extract a forged signature on $\mathsf{vk}_{\mathrm{ot}}$ directly rather than adding encryptions to the proof.

Acknowledgments

This work was supported in part by the European Commission through the ICT Program under Contract ICT-2007-216676 ECRYPT II and by the French ANR-07-SESU-008-01 PAMPA Project.

References

1. M. Abdalla, X. Boyen, C. Chevalier, and D. Pointcheval. Distributed public-key cryptography from weak secrets. In *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, *LNCS* 5443, pages 139–159. Springer, Mar. 2009.
2. B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *Advances in Cryptology – CRYPTO 2005*, *LNCS* 3621, pages 361–377. Springer, Aug. 2005.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, Nov. 1993.
4. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT 2004*, *LNCS* 3027, pages 223–238. Springer, May 2004.
5. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, *LNCS* 3152, pages 41–55. Springer, Aug. 2004.

6. D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology – CRYPTO 2001*, *LNCS* 2139, pages 213–229. Springer, Aug. 2001.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – ASIACRYPT 2001*, *LNCS* 2248, pages 514–532. Springer, Dec. 2001.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145. IEEE Computer Society Press, Oct. 2001.
9. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology – EUROCRYPT 2005*, *LNCS* 3494, pages 404–421. Springer, May 2005.
10. R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, Aug. 2003.
11. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO'84*, *LNCS* 196, pages 10–18. Springer, Aug. 1985.
12. J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *Advances in Cryptology – ASIACRYPT 2006*, *LNCS* 4284, pages 444–459. Springer, Dec. 2006.
13. J. Groth, R. Ostrovsky, and A. Sahai. Non-interactive zaps and new techniques for NIZK. In *Advances in Cryptology – CRYPTO 2006*, *LNCS* 4117, pages 97–111. Springer, Aug. 2006.
14. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology – EUROCRYPT 2008*, *LNCS* 4965, pages 415–432. Springer, Apr. 2008.
15. J. Katz and J. S. Shin. Modeling insider attacks on group key-exchange protocols. In *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 180–189. ACM Press, Nov. 2005.
16. B. R. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology – EUROCRYPT 2005*, *LNCS* 3494, pages 114–127. Springer, May 2005.

A The UC Framework

The aim of the UC security model is to ensure that UC-secure protocols will continue to behave in the ideal way even if executed in arbitrary environments. This model relies on the indistinguishability between two worlds, the ideal world and the real world. In the ideal world, the security is provided by an ideal functionality $\mathcal{F}$.

One can think of it as a trusted party in the context of multi-party computation: this functionality interacts with n users having to compute a function f. These users give their inputs to $\mathcal{F}$, which gives them back their outputs. We stress that there is no communication between the users. $\mathcal{F}$ ensures that the computation is correct and that the users learn nothing more than their own inputs and outputs. Security is then guaranteed since an adversary $\mathcal{A}$ can only learn and thus modify the data of corrupted users.

In order to prove that a protocol Π emulates $\mathcal{F}$, one considers an environment $\mathcal{Z}$ that provides inputs to the users and acts as a distinguisher between the real world (with actual users and a real adversary that can control some of them and also the communication among them) and the ideal world (with dummy users interacting only with the ideal functionality $\mathcal{F}$, and a simulated adversary also interacting with $\mathcal{F}$). We then say that the protocol Π realizes $\mathcal{F}$ if for all polynomial adversaries $\mathcal{A}$, there exists a polynomial simulator $\mathcal{S}$ such that no polynomial environment $\mathcal{Z}$ can distinguish between the two worlds (one with $\mathcal{F}$ and $\mathcal{S}$, the other with Π and $\mathcal{A}$) with a significant advantage.

Since there are several copies of a functionality $\mathcal{F}$ running in parallel, each one has a unique session identifier *sid*. All the messages must contain the SID of the copy they are intended for. As in [9], we assume for simplicity that each protocol realizing $\mathcal{F}$ has inputs containing its SID. We also assume that each user starts a session by specifying the SID of $\mathcal{F}$, its identity P_i, its password pw_i, and the identities of the other users *Pid*.

A shortcoming of the UC theorem is that is says nothing about protocols sharing state and randomness (it ensures the security of a single unit only). Here, since we need a common reference string for all instances of the protocol, we need a stronger result, provided by Canetti and Rabin in [10] and called *universal composability with joint state*. Informally, they define the multi-session extension $\widehat{\mathcal{F}}$ of $\mathcal{F}$, which basically runs multiple instances of $\mathcal{F}$, where each of them is identified by a sub-session identifier *ssid*. $\widehat{\mathcal{F}}$ has to be executed with *sid* and *ssid*. When it receives a message m containing *ssid*, it hands m to the copy of $\mathcal{F}$ having the SSID *ssid* (or invokes a new one if such copy does not exist).

For the sake of generality, we shall describe all the functionalities in the context of *adaptive* adversaries, that are allowed to corrupt users whenever they like to. For simplicity, however, we shall only prove the security of our constructions in presence of *static* adversaries, that have to choose which users to corrupt before the beginning of the execution of the protocol, either implicitly in the key generation protocol (when the adversary starts playing as one of the parties, choosing by himself the password), or explicitly in the decryption protocol (asking a corruption before a new decryption session).

In the UC model, a corruption implies a complete access to the internal memory of the users (which here means the password and the internal state); in addition, the adversary takes the entire control of the corrupted user, and can modify its behavior for the remaining of the protocol.

B Proof of the Security Theorems

In this section we give the proof that the protocols described on Figures 1 and 2 (in Section 4) respectively realize the functionalities $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ and $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$. These ideal functionalities have been described in Section 2 and can be found in Figures 3 and 4. The split functionality is given in Figure 5.

The functionality $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ is parametrized by a security parameter k and an efficiently computable function $\mathsf{PublicKeyGen} : (\mathsf{pw}_1, \mathsf{pw}_2, \ldots, \mathsf{pw}_n) \mapsto \mathsf{pk}$ that derives a public key pk from a set of passwords. Denote by *role* either *player* or *leader*. The functionality interacts with an adversary $\mathcal{S}$ and a set of parties $P_1,\ldots,P_n$ via the following queries:

Initialization. Upon receiving a query $(\mathsf{newSession}, sid, Pid, P_i, \mathsf{pw}_i, role)$ from user P_i for the first time, where Pid is a set of at least two distinct identities containing P_i, ignore it if $role = leader$ and if there is already a record of the form $(sid, Pid, *, *, leader)$. Record $(sid, Pid, P_i, \mathsf{pw}_i, role)$ and send $(sid, Pid, P_i, role)$ to $\mathcal{S}$. Ignore any subsequent query $(\mathsf{newSession}, sid, Pid', *, *, *)$ where $Pid' \neq Pid$.

If there are already $|Pid| - 1$ recorded tuples $(sid, Pid, P_j, \mathsf{pw}_j)$ for $P_j \in Pid \setminus \{P_i\}$, and exactly one of them such that $role = leader$, then while recording the $|Pid|$-th tuple, also record $(sid, Pid, \mathsf{ready})$ and send this to $\mathcal{S}$. Otherwise, record $(sid, Pid, \mathsf{error})$ and send $(sid, Pid, \mathsf{error})$ to $\mathcal{S}$.

Key Computation. Upon receiving a message $(\mathsf{compute}, sid, Pid)$ from the adversary $\mathcal{S}$ where there is a recorded tuple $(sid, Pid, \mathsf{ready})$, then compute $\mathsf{pk} = \mathsf{PublicKeyGen}(\mathsf{pw}_1, \ldots, \mathsf{pw}_n)$ and record (sid, Pid, pk).

Leader Key Delivery. Upon receiving a message $(\mathsf{leaderDeliver}, sid, Pid, \mathsf{b})$ from the adversary $\mathcal{S}$ for the first time, where there is a recorded tuple (sid, Pid, pk) and a record $(sid, Pid, P_i, pw_i, leader)$, send (sid, Pid, pk) to P_i and to $\mathcal{S}$ if $\mathsf{b} = 1$, or $(sid, Pid, \mathsf{error})$ otherwise. Record $(sid, Pid, \mathsf{sent})$ and send this to $\mathcal{S}$.

Player Key Delivery. Upon receiving a message $(\mathsf{playerDeliver}, sid, Pid, \mathsf{b}, P_i)$ from $\mathcal{S}$ where there are recorded tuples (sid, Pid, pk), $(sid, Pid, P_i, pw_i, player)$ and $(sid, Pid, \mathsf{sent})$, send (sid, Pid, pk) to P_i if $\mathsf{b} = 1$, or $(sid, Pid, \mathsf{error})$ otherwise.

User Corruption. If $\mathcal{S}$ corrupts $P_i \in Pid$ where there is a recorded tuple $(sid, Pid, P_i, \mathsf{pw}_i)$, then reveal pw_i to $\mathcal{S}$. If there also is a recorded tuple (sid, Pid, pk) and if (sid, Pid, pk) has not yet been sent to P_i, send (sid, Pid, pk) to $\mathcal{S}$.

Fig. 3. The Distributed Key Generation Functionality $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$

The proof of the distributed key generation protocol is similar to that of the distributed private computation given below, with the added simplification that there is no verification step and the difference that all the users receive the result in the end (which corresponds exactly to what happens in the private computation protocol at the end of the verification step, where everyone also receives the result). Thus, we refer to the proof of the private computation protocol for the workings of the simulation. The additional simplification implied by the adversary receiving the result in the end will be given in remarks. The structure of the proof is similar to that of Abdalla *et al.* in [1].

B.1 Methodology of the Proof

The objective of the proof is to construct, from a real-world adversary $\mathcal{A}$, an ideal-world simulator $\mathcal{S}$, so that the behavior of $\mathcal{A}$ in the real world and that of $\mathcal{S}$ in the ideal world are indistinguishable to

The functionality $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$ is parametrized by a security parameter k and three functions. $\mathsf{PublicKeyVer}$ is a boolean function defined by $\mathsf{PublicKeyVer} : (\mathsf{pw}_1, \mathsf{pw}_2, \ldots, \mathsf{pw}_n, \mathsf{pk}) \mapsto b$, where $b = 1$ if the passwords and the public key are compatible, $b = 0$ otherwise. $\mathsf{SecretKeyGen}$ is a function $\mathsf{SecretKeyGen} : (\mathsf{pw}_1, \mathsf{pw}_2, \ldots, \mathsf{pw}_n) \mapsto \mathsf{sk}$, where sk is the secret key obtained from the passwords. Finally, $\mathsf{PrivateComp}$ is a private-key function $\mathsf{PrivateComp} : (\mathsf{sk}, c) \mapsto m$, where sk is the secret key, c is the function input (e.g., a ciphertext) and m the private result of the computation (e.g., the decrypted message). Denote by *role* either *player* or *leader*. The functionality interacts with an adversary $\mathcal{S}$ and a set of parties $P_1, \ldots, P_n$ via the following queries:

- **Initialization.** Upon receiving a query $(\mathsf{newSession}, sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, role)$ from user P_i for the first time, where Pid is a set of at least two distinct identities containing P_i, ignore it if $role = leader$ and if there is already a record of the form $(sid, Pid, *, *, *, *, leader)$. Record $(sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, role)$, mark it fresh, and send $(sid, Pid, P_i, \mathsf{pk}, c, role)$ to $\mathcal{S}$. Ignore any subsequent query $(\mathsf{newSession}, sid, Pid', *, *, *, *, *)$ where $Pid' \neq Pid$.
  If there are already $|Pid| - 1$ recorded tuples $(sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, role)$, and exactly one of them such that $role = leader$, then after recording the $|Pid|$-th tuple, verify that the values of c and pk are the same for all the users. If the tuples do not fulfill all of these conditions, report $(sid, Pid, \mathsf{error})$ to $\mathcal{S}$ and stop. Otherwise, record $(sid, Pid, \mathsf{pk}, c, \mathsf{ready})$ and send it to $\mathcal{S}$.
- **Private Computation.** Upon receiving a message $(\mathsf{compute}, sid, Pid)$ from $\mathcal{S}$ where there is a recorded tuple $(sid, Pid, \mathsf{pk}, c, \mathsf{ready})$, then
  - If all records are fresh and $\mathsf{PublicKeyVer}(\mathsf{pw}_1, \ldots, \mathsf{pw}_n, \mathsf{pk}) = 1$, then compute $\mathsf{sk} = \mathsf{SecretKeyGen}(\mathsf{pw}_1, \ldots, \mathsf{pw}_n)$ and $m = \mathsf{PrivateComp}(\mathsf{sk}, c)$, and store (sid, Pid, m). Next, for all $P_i \in Pid$ mark the record $(sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, role)$ as $\mathsf{complete}$.
  - In any other case, store $(sid, Pid, \mathsf{error})$.

  When the computation result is set, report the outcome (either error or $\mathsf{complete}$) to $\mathcal{S}$.
- **Leader Computation Delivery.** Upon receiving a message of the form $(\mathsf{leaderDeliver}, sid, Pid, \mathsf{b})$ from the adversary $\mathcal{S}$, where both there is a recorded tuple (sid, Pid, m) such that $m \in \{well\text{-}formed\ messages\} \cup \{\mathsf{error}\}$, and there exists a record $(sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, leader)$, send (sid, Pid, m) to P_i if b is equal to 1, or send $(sid, Pid, \mathsf{error})$ if b is equal to 0. If the group leader P_i is corrupted, then send (sid, Pid, m) to $\mathcal{S}$ as well (note that $\mathcal{S}$ gets m automatically if P_j is corrupted).
- **User Corruption.** If $\mathcal{S}$ corrupts $P_i \in Pid$ where there is a recorded tuple $(sid, Pid, P_i, \mathsf{pk}, c, \mathsf{pw}_i, role)$, then reveal pw_i to $\mathcal{S}$. If $role = leader$, if there also is a recorded tuple (sid, Pid, m), and if (sid, Pid, m) has not yet been sent to P_i, then also send (sid, Pid, m) to $\mathcal{S}$.

Fig. 4. The Distributed Private Computation Functionality $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$

Given a functionality $\mathcal{F}$, the split functionality $s\mathcal{F}$ proceeds as follows:

Initialization:

- Upon receiving (Init, sid) from party P_i, send $(\mathsf{Init}, sid, P_i)$ to the adversary.
- Upon receiving a message $(\mathsf{Init}, sid, P_i, H, sid_H)$ from $\mathcal{A}$, where H is a set of party identities, check that P_i has already sent (Init, sid) and that for all recorded $(H', sid_{H'})$, either $H = H'$ and $sid_H = sid_{H'}$ or H and H' are disjoint and $sid_H \neq sid_{H'}$. If so, record the pair (H, sid_H), send $(\mathsf{Init}, sid, sid_H)$ to P_i, and invoke a new functionality $(\mathcal{F}, sid_H)$ denoted as $\mathcal{F}_H$ and with set of honest parties H.

Computation:

- Upon receiving (Input, sid, m) from party P_i, find the set H such that $P_i \in H$ and forward m to $\mathcal{F}_H$.
- Upon receiving $(\mathsf{Input}, sid, P_j, H, m)$ from $\mathcal{A}$, such that $P_j \notin H$, forward m to $\mathcal{F}_H$ as if coming from P_j.
- When $\mathcal{F}_H$ generates an output m for party $P_i \in H$, send m to P_i. If the output is for $P_j \notin H$ or for the adversary, send m to the adversary.

Fig. 5. Split Functionality $s\mathcal{F}$

any environment $\mathcal{Z}$. The ideal functionality is specified in Figure 4 and described in Section 2. Since we use the joint state version of the UC theorem, we implicitly consider the multi-session extension of this functionality, and thus replace all sid by $(\mathsf{sid}, \mathsf{ssid})$. Note that the passwords of the users depend on the sub-session considered. For sake of simplicity, we denote them by pw_i, but one should implicitly understand $\mathsf{pw}_{i,\mathsf{ssid}}$.

B.2 Details of the Proof

In this proof, we incrementally define a sequence of games starting from the one describing a real execution of the protocol and ending up with game $\mathbf{G}_{10}$ which we prove to be indistinguishable with respect to the ideal experiment.

The simulator $\mathcal{S}$ is incrementally defined in the games, ending up to be completely defined in $\mathbf{G}_{10}$ (though we do not rewrite him entirely in this game since his behavior was described in the previous games). This final game will then be proven to be indistinguishable to the ideal world, showing that we indeed have constructed an ideal simulator to the real-world adversary $\mathcal{A}$.

In the first games, the simulator has actually access to all the information given to the users by the environment, in particular their passwords. In the last game, we nearly are in the ideal game so that the users do not exist anymore: $\mathcal{S}$ has only access to the information transmitted by his queries to the functionality (not to the passwords, for instance) and he has to simulate the users entirely by himself. Between these two situations, the simulator lives in a world which is not really real, not really ideal.

Following [1], in order to formally model this situation, we chose to consider two hybrid queries that $\mathcal{S}$ can ask to the functionality all along the games. The CompatiblePwd query checks whether the passwords of the users are compatible with the passwords of the other users and the public key of the group leader. And the Delivery query gives the result to the group leader–and to the adversary if the former is corrupted.

Note that since in the first games, the simulator has access to the users' inputs, he knows their passwords. In such a case a CompatiblePwd or Delivery query can be easily implemented by letting the simulator look at the passwords owned by the users. When the users are entirely simulated, without the knowledge of their passwords, $\mathcal{S}$ will replace the queries above with the real Private Computation and Leader Computation Delivery queries to the functionality.

We say that a flow is *oracle-generated* if it was sent by an honest user and arrives without any alteration to the user it was meant to. We say it is *non-oracle-generated* otherwise, that is either if it was sent by an honest user and modified by the adversary, or if it was sent by a corrupted user or a user impersonated by the adversary (more generally denoted by *attacked* user, that is, a user whose password is known to the adversary).

Game $\mathbf{G}_0$: Real game. In the real game, we know that the protocol cannot continue past the two initial commitment rounds if there is any inconsistency between the passwords pw_i used in all the commitments. Any inconsistency will violate the SSNIZK language, and because the proof system with honest setup is assumed to be sound, it is not feasible for anyone to prove a false statement. Similarly, the copies of pk and c held by all the users and the commitments received in the first round are assumed to be the same for all users, thanks to the one-time signatures. Finally, the two rounds of blinding and unblinding serve to verify the consistency of pk with the pw_i's, and to compute the final output c^{sk}, respectively. To be precise, the security of these rounds follows from our assumptions: cheating in the computation of the blinding/unblinding rounds without getting caught is impossible since the CDH are publicly verifiable. Finally note that the players are not granted the right to check that the group leader uses the correct exponents when it publishes ζ'_{n+1}. This could only be a sensible issue when the group leader is corrupted, but we will show that it cannot get any additional information.

Game $\mathbf{G}_1$: Simulation of the SSNIZK proofs. This game modifies how the zero-knowledge proofs are performed. Specifically, instead of using the honest-prover strategy, all the proofs Π_j^1 and Π_j^2 in which the prover is an honest user are simulated using the zero-knowledge simulator (also note that the common reference string γ is also simulated once and for all). Since the proofs are concurrent zero-knowledge, the environment cannot distinguish between the two games $\mathbf{G}_1$ and $\mathbf{G}_0$. That is, if an environment could distinguish between these hybrids (ie become aware that the two commitments possibly commit to/encrypt different passwords when the simulator claim the passwords are the same),

one could construct an adversary that breaks the zero-knowledge property of the proof protocol. We assume that he still knows the passwords of the players so that everything else is done correctly.

Game $\mathbf{G}_2$: Use of a *linear* basis (U_1, U_2, U_3). In this game, the simulator is allowed to swap the basis (U_1, U_2, U_3) for a linear one used for $\mathbf{C_i}$. This will allow him to simulate properly, and in particular to be able to extract the password committed in the values $\mathbf{C_i}$'s. The adversary cannot become aware of this change thanks to the DLin assumption. The adversary still knows the passwords of the players and computes everything honestly. Furthermore, the soundness of the SSNIZK proofs still holds thanks to the indistiguishability of the 3-tuples (relying on the DLin assumption). Thus, this game is indistinguishable from the previous one.

Game $\mathbf{G}_3$: Simulation of the first round of commitments. The passwords are still known to the simulator, but he uses dummy passwords for the first round of commitments. He then uses the real ones for the second rounds of commitments (C'_i and A_i). Thanks to the simulation-soundness of the proofs, the adversary cannot become aware that the simulator is proving a wrong statement in $(1b)$. Furthermore, thanks to the witness indistinguishability of the first proof (since the basis is linear), the environment cannot become aware that we are using dummy passwords in $\mathbf{C_i}$. This game is thus indistinguishable from the previous one.

Game $\mathbf{G}_4$: Use of a non-linear basis. Everything works as before except that the simulator comes back with a non-linear basis in the first step. The simulator is then allowed to program (once and for all) the totality of the common reference string. This is indistinguishable from the previous game due to the DLin assumption.

Game $\mathbf{G}_5$: Simulation of the first round of commitments. From this game on, $\mathcal{S}$ simulates the first rounds of commitments in the following way. We suppose that he still knows the passwords of the players. Let n_h be the number of honest users, i.e. the users $\mathcal{S}$ has to simulate. $\mathcal{S}$ chooses at random n_h dummy passwords $\widetilde{\mathsf{pw}}_{i_1}, \ldots, \widetilde{\mathsf{pw}}_{i_{n_h}}$ on behalf of each one of these users. Once all these values are set, $\mathcal{S}$ computes the first-round commitments and send them to everybody.

$\mathcal{S}$ then learns from the functionality whether the users all share the same c and pk or not. In the second case, $\mathcal{S}$ aborts the game. In the first case, $\mathcal{S}$ goes on the execution of the protocol in a honest way, using the real passwords of the users. Note that he will have to prove false statements, which is not a problem since the proofs are simulated since the former games. In the end, if the execution succeeds, he asks a Computation query, and he finally sets the bit b to 1 in the Delivery query. Otherwise, if the execution fails, he also asks a Computation query but sets the bit b to 0 for the delivery.

Since the first-round commitment is hiding, the adversary cannot become aware of the transformation of the $\widetilde{\mathsf{pw}}_i$'s into pw_i's: this game is indistinguishable from $\mathbf{G}_4$.

Game $\mathbf{G}_6$: Simulation of honest users with compatible passwords. From this game on, we show how to simulate the users without using their passwords. More precisely, the simulator is still supposed to know the passwords of the users, but little by little we are going to show that he actually never needs them in the simulation. This will ensure in the end that the simulator does not need the knowledge of the passwords in order to perform the simulation honestly.

Note that from this game on, we can suppose that all the c and the pk are identical, since the case of different values has been dealt with in the former game (and the simulator aborts the protocol in this case). The first round of commitments is simulated as in $\mathbf{G}_5$. We now face two cases.

First, if there are attacked users among the group, the simulation continues as in the former game, $\mathcal{S}$ being allowed to use the passwords of all the users. (We show in $\mathbf{G}_8$ and $\mathbf{G}_9$ how to simulate in this case without using the passwords.)

Second, if all users are honest, we show how to continue the simulation without the help of the passwords. Note that if at some point some flows are non-oracle-generated, the protocol will abort: In this case, the simulator will set the bit b to 0 in the Delivery query. This comes from the non-malleability

of the SSNIZK proofs. If the adversary has not generated the first commitments, he will not be able to construct valid proofs, with unknown witnesses.

The simulator first asks a Computation query along with a Delivery query to the functionality, which gives him either complete or error. In the second case, $\mathcal{S}$ continues the simulation as in the former game, and we allow him to use the passwords of the users (we show in $\mathbf{G}_7$ how to get rid of the use of the passwords).

We now consider the first case and sum up briefly the circumstances which led us here: The users are honest, they have the same c and pk, and their passwords are compatible with the public key. Then, $\mathcal{S}$ keeps on using the passwords $\widetilde{\mathsf{pw}}_2, \ldots, \widetilde{\mathsf{pw}}_n$ for the $n-1$ last users, and he is able to set $g^{\widetilde{\mathsf{pw}}'_1}$ (without knowing $\widetilde{\mathsf{pw}}'_1$) such that all the passwords are compatible with the value g^{sk}, using the equation:

$$g^{\widetilde{\mathsf{pw}}'_1} g^{\widetilde{\mathsf{pw}}_2} \ldots g^{\widetilde{\mathsf{pw}}_n} = g^{\mathsf{sk}} = \mathsf{pk}$$

But he still uses $c^{\widetilde{\mathsf{pw}}_1}$ for the last commitment, since he does not know c^{sk}. Notice that he will give once again a proof for a false statement. The simulator then continues honestly the game, by choosing random values α_i and β_i and executing the four rings as described in the protocol. Note that $\mathcal{S}$ sends a random value for $\zeta'_{n+1} = c^{\mathsf{sk}\beta x}$ on behalf of the (honest) group leader. This means that the simulator does not know x, but such an x exists to be consistant with the actual (but unknown) values of β and sk. The following is then perfectly indistinguishable since the blinding factor x will never be publicly removed. Then, no information about x leaks. In the Delivery query, the bit b is chosen as described in $\mathbf{G}_5$. Finally note that the simulator never needed the knowledge of the passwords of the users.

Due to the non-malleability of the commitments, along with their hiding property, this game is indistinguishable from $\mathbf{G}_5$.

Remark. Note that for the distributed key generation protocol, things would have been simpler. In this case, the simulator not only receives complete or error, but the exact value of $\mathsf{pk} = g^{\mathsf{sk}}$. He is thus able to modify $\widetilde{\mathsf{pw}}_1$ such that the passwords are compatible with the public key. This is similar to what the simulator will do in $\mathbf{G}_8$ for c^{sk} when the group leader is corrupted. Indeed, in this case, the simulator will learn the exact value of c^{sk} from the functionality (as he does for g^{sk} here).

Game $\mathbf{G}_7$: Simulation of honest users with incompatible passwords. This game starts exactly as in $\mathbf{G}_6$. If there are attacked users, the simulator is granted the right to use the passwords of all the users, and continues as in $\mathbf{G}_5$.

If all the users are honest, he continues as in $\mathbf{G}_6$, by asking a CompatiblePwd query. We showed in $\mathbf{G}_6$ how to deal with the case where the functionality returns correct (that is, when the passwords are compatible with the public key) without using the passwords of the users. We now consider the other case and show how to treat it. Thus, we suppose that the CompatiblePwd query returns an error, meaning that the passwords are incompatible with the public key.

The simulator then computes commitments of the values $g^{\widetilde{\mathsf{pw}}_i}$ and $c^{\widetilde{\mathsf{pw}}_i}$ for all the users (there is no need that their passwords should be compatible). He sends them along with the corresponding proofs to the other users. The simulator then continues honestly the game, by choosing random values α_i and β_i and executing the two first rings as described in the protocol. Note that the real passwords of the users are not needed anymore. Since the protocol will fail at the verification step, the simulator will set the bit b to 0 in the Delivery query.

Since the commitments are hiding, this game is indistinguishable from $\mathbf{G}_6$.

Game $\mathbf{G}_8$: Simulation in case of compatible passwords in presence of an adversary. This game starts exactly as in $\mathbf{G}_5$. The case where all the users are honest has been dealt with in the games $\mathbf{G}_6$ and $\mathbf{G}_7$. We now consider the case in which the adversary controls a set of users. Recall that we denote by n_h the number of honest users.

Note that choosing g_1, g_2 and g_3 allows $\mathcal{S}$ to know the discrete logarithm of g_1 and g_2 in base g_3. Since the commitments are linear ciphertexts, knowledge of these discrete logarithms will allow the simulator to decrypt the ciphertexts and then extract the passwords used by $\mathcal{A}$ in the first-round commitments. (The actual extraction requires taking discrete logarithms, but this can be done efficiently using generic

methods, because the discrete logarithms to extract are bits. And the small size is enforced by a WIProofBit.)

After this first round, the simulator thus extracts the passwords used by the adversary in the commitments he sent. Note that the honest users are not supposed to have received the same values from the adversary: We only know that these values are non-oracle-generated, but not necessarily equal. Thus, the simulator chooses at random one of the commitments received from an attacked user to extract and recover its password. One could argue that there is a problem here, but note that the signature given with the second commitment will not be accepted if the value H is not the same for all users (recall that we have assumed a collision-resistant hash function for the computation of H).

Once $\mathcal{S}$ has recovered all the passwords of the attacked users, he asks a CompatiblePwd query. If this query returns incorrect, we continue the simulation as in $\mathbf{G}_5$ (we show in $\mathbf{G}_9$ how to deal with this case without using the passwords of the users).

We now suppose that the query returns correct. This provides the following equation:

$$g^{\sum_{j=1}^{n_h} \mathsf{pw}_{i_j}} = \mathsf{pk} \Big/ g^{P_i \sum_{\text{attacked}} \mathsf{pw}_i}$$

Recall that the passwords of the honest users were chosen at random, so there is no chance that they should be compatible with the (common) public key. $\mathcal{S}$ thus keeps its $n_h - 1$ first passwords and computes a replacement value $g^{\widetilde{\mathsf{pw}}'_{i_{n_h}}}$ thanks to the above equation. Remark that he does not know the corresponding password $\widetilde{\mathsf{pw}}'_{i_{n_h}}$.

In the second round, the simulator must produce commitments that are compatible with the public key. To do so, he makes commitments on the same random passwords as before for the $n_h - 1$ first honest users, and for the last one creates a commitment as a linear encryption of $g^{\widetilde{\mathsf{pw}}'_{i_{n_h}}}$. He sends them all out. In the same round, the simulator must also produce another series of password commitments, this time as linear encryptions of c^{pw_i} and not g^{pw_i}.

We now have to consider two cases. First, suppose that the group leader is attacked. The simulator computes the commitments normally for the first $n_h - 1$ honest players using the simulated passwords. For the last player $P_{i_{n_h}}$, he asks a Delivery query in order to recover c^{sk}, and computes the missing commitment as a linear encryption of $c^{\widetilde{\mathsf{pw}}''_{i_{n_h}}}$, which is given by

$$c^{\widetilde{\mathsf{pw}}''_{n_h}} = c^{\mathsf{sk}} \Big/ c^{\sum_{j=1}^{n_h-1} \widetilde{\mathsf{pw}}_{i_j}} c^{P_i \sum_{\text{attacked}} \mathsf{pw}_i}$$

Otherwise, if the group leader is not attacked, the simulator will not recover c^{sk}. We thus proceed as in $\mathbf{G}_6$, using incorrect values $c^{\widetilde{\mathsf{pw}}_{i_j}}$ for the honest players, not compatible with c^{sk}.

$\mathcal{S}$ sends out the commitments along with the proofs of consistency. Note that in the second case he will prove a false statement for the group leader.

The simulator then continues the game honestly, by choosing random values α_i and β_i and executing the four rings as described in the protocol. Everything is simulated perfectly if the group leader is attacked (with incorrect passwords, though, but compatible with c^{sk}). If the group leader tries to cheat in the value ζ'_{n+1}, the unblinding ring will still use the values β_i that it does not know, and $\mathcal{S}$ can simulate this unblinding ring correctly. The simulation of the honest players is done independently of the group leader (whether it plays honestly or not), so that it cannot learn anything more than the simulator already knows, that is c^{sk}.

Otherwise, the last step fails, without any bad consequence on the protocol since the group leader ends the ring (as in $\mathbf{G}_6$). The bit b of the Delivery query is chosen as described in $\mathbf{G}_5$: if something goes wrong and the protocol halts, then $b = 0$, otherwise $b = 1$.

Since the commitments are computationally hiding under the DLin assumption, the adversary cannot become aware that the passwords are not the good ones, or that the password for the user $P_{i_{n_h}}$ changed between the two rounds of commitments. This game is indistinguishable from $\mathbf{G}_7$.

Game $\mathbf{G}_9$: Simulation in case of incompatible passwords in presence of an adversary. This game starts exactly as in $\mathbf{G}_8$. We now suppose that the CompatiblePwd query returns incorrect. Since the verification step will fail, $\mathcal{S}$ can keep all the values $\widetilde{\mathsf{pw}}_{i_1}, \ldots, \widetilde{\mathsf{pw}}_{i_{n_h}}$ for the second round of commitments (in base g as in base c). He then sends these commitments along with the corresponding proofs.

He then continues honestly the protocol (without knowing the real passwords of the users) until it fails, at the verification step. The simulator then asks a CompatiblePwd query, and a Delivery query with bit $b = 0$. For the same reasons than in the former game, $\mathbf{G}_9$ is indistinguishable from $\mathbf{G}_8$.

Game $\mathbf{G}_{10}$: Indistinguishability with the ideal world. We have shown that $\mathcal{S}$ is able in any case to simulate the whole protocol without using the passwords of the users. Thus, we can now suppose that he does not know these passwords.

The only difference between $\mathbf{G}_9$ and $\mathbf{G}_{10}$ is that the CompatiblePwd query is replaced by a Private Computation query to the functionality, and the Delivery by a Leader Computation Delivery query. If a session aborts or terminates, $\mathcal{S}$ reports it to $\mathcal{A}$. If a session terminates with a message m, then $\mathcal{S}$ makes a Delivery call to the functionality, specifying a bit $\mathsf{b} = 1$. If the protocol fails, he gives a bit $\mathsf{b} = 0$.

We now show that this last game $\mathbf{G}_{10}$ is indistinguishable from the ideal-world experiment IWE. More precisely, we have to show that the group leader receives a correct message in $\mathbf{G}_{10}$ if and only if it receives a correct message in the ideal world.

First, if the users share compatible passwords, the same public key, the same ciphertext, and all the flows are oracle-generated until the end of the game, then the group leader will obtain a correct message, both in $\mathbf{G}_{10}$ (from $\mathbf{G}_6$) and the ideal world. More precisely, the protocol will end on a random value in $\mathbf{G}_6$, but nobody will be able to distinguish it from the real value. And the leader will obtain the correct value from the Delivery query. Second, if they share compatible passwords, the same public key, the same ciphertext, and if the group leader is corrupted, then the group leader will also receive a correct message (from $\mathbf{G}_8$). Third, if they do not share compatible passwords or if some received flows differ from one user to an other, then the group leader will get an error (from $\mathbf{G}_7$ and $\mathbf{G}_9$ in the first case, and due to the bit b otherwise).

This establishes that, given any adversary $\mathcal{A}$ that attacks the protocol Π in the real world, we can build a simulator $\mathcal{S}$ that interacts with the functionality $\mathcal{F}$ in the ideal world, in such a way that the environment cannot distinguish which world it is in.