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.

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 [9]. 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. 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 $\mathsf{TestPwd}$ 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 $\mathsf{TestPwd}$ 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 the full version [8]), 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 $\mathcal{F}_{\mathsf{pwDistPublicKeyGen}}$ is to provide a public key to the users, computed according to their passwords with respect to a function $\mathsf{PublicKeyGen}$ 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 $\mathcal{F}_{\mathsf{pwDistPrivateComp}}$, 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 $\mathsf{PublicKeyVer}$, and if it is correct it will then compute the secret key sk from the passwords with the help of a function $\mathsf{SecretKeyGen}$, and from there evaluate $\mathsf{PrivateComp}(\mathsf{sk}, in)$ 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 [10], 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, $\mathsf{sk}_{\mathrm{lin}} = (x_1, x_2)$, and the public key, $\mathsf{pk}_{\mathrm{lin}} = (g_1, g_2, g_3)$, 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 $r_1, r_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, and the ciphertext consists of $C = \mathcal{E}_{\mathsf{pk}_{\mathrm{lin}}}(M; r_1, r_2) = (C_1, C_2, C_3) = ({g_1}^{r_1}, {g_2}^{r_2}, M{g_3}^{r_1+r_2})$. The decryption process consists of $M = \mathcal{D}_{\mathsf{pk}_{\mathrm{lin}}}(C) = C_3/({C_1}^{x_1}{C_2}^{x_2})$.

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:

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

The password/public-key verification function is then

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

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 the full version), 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: $C'_i = (C'^{(1)}_i, C'^{(2)}_i, C'^{(3)}_i) = ({g_1}^{t_i}, {g_2}^{u_i}, g^{\mathsf{pw}_i} {g_3}^{t_i+u_i})$ (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 [10]. 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

Commitment First Step

$$\left\{\begin{array}{l}
(1a)\ r_{i,j}, s_{i,j} \xleftarrow{R} \mathbb{Z}_p^* \\
C_{i,j} = \mathsf{Commit}_g(\mathsf{pw}_{i,j}, r_{i,j}, s_{i,j}) = ({U_1}^{\mathsf{pw}_{i,j}} {g_1}^{r_{i,j}}, {U_2}^{\mathsf{pw}_{i,j}} {g_2}^{s_{i,j}}, g^{\mathsf{pw}_{i,j}} {g_3}^{r_{i,j}+s_{i,j}}) \\
\Pi^0_{i,j} = \mathsf{WIProofBit}(C_{i,j}) \\
(\mathsf{SK}_i, \mathsf{VK}_i) \leftarrow \mathsf{SKG} \qquad \xrightarrow{\mathbf{C_i} = \{C_{i,j}\}_j, \{\Pi^0_{i,j}\}_j, \mathsf{VK}_i}
\end{array}\right.$$

Commitment Second Step

$$\left\{\begin{array}{l}
(1b)\ H = \mathcal{H}(\mathbf{C_1}, \ldots, \mathbf{C_n}, \mathsf{VK}_1, \ldots, \mathsf{VK}_n) \qquad t_i, u_i \xleftarrow{R} \mathbb{Z}_p^* \\
C'_i = \mathsf{Encrypt}_g(\mathsf{pw}_i, t_i, u_i) = ({g_1}^{t_i}, {g_2}^{u_i}, g^{\mathsf{pw}_i} {g_3}^{t_i+u_i}) \\
C_i = \left(\prod \left({C_{i,j}}^{(1)} \right)^{2^j}, \prod \left({C_{i,j}}^{(2)} \right)^{2^j}, \prod \left({C_{i,j}}^{(3)} \right)^{2^j} \right) \\
\Pi^1_i = \mathsf{SSNIZKEq}_{g,g}(C_i, C'_i) \qquad \sigma_i = \mathsf{Sign}(H; \mathsf{SK}_i) \qquad \xrightarrow{C'_i, \Pi^1_i, \sigma_i}
\end{array}\right.$$

Blinding Ring

$$\left\{\begin{array}{l}
(1c)\ \text{abort if one of the signatures } \sigma_i \text{ is invalid} \\
\gamma_0^{(0)} = \prod_i {C'_i}^{(3)} = g^{\sum_i \mathsf{pw}_i} {g_3}^{\sum_i t_i + \sum_i u_i} \quad \gamma_0^{(1)} = g_1 \quad \gamma_0^{(2)} = g_2 \quad \gamma_0^{(3)} = g_3 \\
\text{This round is done sequentially, for i=1,...,n.} \\
\text{Upon receiving } (\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}, \gamma_j^{(3)}) \text{ for } j = 1, \ldots, i-1, \\
\text{check } \mathsf{CDH}(\gamma_{j-1}^{(0)}, \gamma_j^{(0)}, \gamma_{j-1}^{(1)}, \gamma_j^{(1)}), \mathsf{CDH}(\gamma_{j-1}^{(0)}, \gamma_j^{(0)}, \gamma_{j-1}^{(2)}, \gamma_j^{(2)}) \\
\qquad \text{and } \mathsf{CDH}(\gamma_{j-1}^{(0)}, \gamma_j^{(0)}, \gamma_{j-1}^{(3)}, \gamma_j^{(3)}); \text{ abort if one of them is invalid} \\
\alpha_i \xleftarrow{R} \mathbb{Z}_p^* \quad \gamma_i^{(0)} = (\gamma_{i-1}^{(0)})^{\alpha_i} \quad \gamma_i^{(1)} = (\gamma_{i-1}^{(1)})^{\alpha_i} \quad \gamma_i^{(2)} = (\gamma_{i-1}^{(2)})^{\alpha_i} \\
\qquad \gamma_i^{(3)} = (\gamma_{i-1}^{(3)})^{\alpha_i} \qquad \xrightarrow{\gamma_i^{(0)}, \gamma_i^{(1)}, \gamma_i^{(2)}, \gamma_i^{(3)}} \\
(1d)\ \text{given } \gamma_n^{(0)} = g^{\alpha \sum_i \mathsf{pw}_i} {g_3}^{\alpha(\sum_i t_i + \sum_i u_i)} \quad \gamma_n^{(1)} = {g_1}^{\alpha} \quad \gamma_n^{(2)} = {g_2}^{\alpha} \quad \gamma_n^{(3)} = {g_3}^{\alpha} \\
\qquad \text{check } \mathsf{CDH}(\gamma_{n-1}^{(0)}, \gamma_n^{(0)}, \gamma_{n-1}^{(1)}, \gamma_n^{(1)}), \mathsf{CDH}(\gamma_{n-1}^{(0)}, \gamma_n^{(0)}, \gamma_{n-1}^{(2)}, \gamma_n^{(2)}) \\
\qquad \text{and } \mathsf{CDH}(\gamma_{n-1}^{(0)}, \gamma_n^{(0)}, \gamma_{n-1}^{(3)}, \gamma_n^{(3)}) \\
\text{for all } i, P_i \text{ computes } G_{1,i} = (\gamma_n^{(1)})^{t_i}, G_{2,i} = (\gamma_n^{(2)})^{u_i}, \\
\qquad G_{3,i} = (\gamma_n^{(3)})^{t_i}, G_{4,i} = (\gamma_n^{(3)})^{u_i} \qquad \xrightarrow{G_{1,i}, G_{2,i}, G_{3,i}, G_{4,i}}
\end{array}\right.$$

Unblinding Ring

$$\left\{\begin{array}{l}
(1e)\ \text{given, for } j = 1, \ldots, n \qquad G_{1,j}, G_{2,j}, G_{3,j}, G_{4,j} \\
\qquad \text{check } \mathsf{CDH}(g_1, {C'_j}^{(1)}, \gamma_n^{(1)}, G_{1,j}), \mathsf{CDH}(g_2, {C'_j}^{(2)}, \gamma_n^{(2)}, G_{2,j}), \\
\qquad \mathsf{CDH}(\gamma_n^{(1)}, G_{1,j}, \gamma_n^{(3)}, G_{3,j}) \text{ and } \mathsf{CDH}(\gamma_n^{(2)}, G_{2,j}, \gamma_n^{(3)}, G_{4,j}) \\
\zeta_{n+1} = \gamma_n^{(0)} / \left(\prod_j G_{3,j} G_{4,j} \right) = g^{\alpha \sum_j \mathsf{pw}_j} \\
\text{This round is done sequentially, for i from n down to 1.} \\
\text{given, for } j \text{ from } n \text{ down to } i+1, \zeta_j, \text{check } \mathsf{CDH}(\gamma_{j-1}^{(1)}, \gamma_j^{(1)}, \zeta_j, \zeta_{j+1}) \\
\zeta_i = (\zeta_{i+1})^{1/\alpha_i} \qquad \xrightarrow{\zeta_i} \\
(1f)\ \text{given, for } j \text{ from } i-1 \text{ down to } 1, \zeta_j, \text{check } \mathsf{CDH}(\gamma_{j-1}^{(1)}, \gamma_j^{(1)}, \zeta_j, \zeta_{j+1}) \\
\mathsf{pk} = \zeta_1
\end{array}\right.$$

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

(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: $\Pi^2_i = \mathsf{SSNIZKEq}_{g,c}(C'_i, A_i)$.

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 $x \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$ 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 the full version [8] 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.

Commitment Steps

$(2a) = (1a)$ $\quad \xrightarrow{\{C_{i,j}, \Pi^0_{i,j}\}_j}$

$(2b) = (1b) \quad \text{except } v_i, w_i \xleftarrow{R} \mathbb{Z}_p^*$

$A_i = \mathsf{Encrypt}_c(\mathsf{pw}_i, v_i, w_i) = ({g_1}^{v_i}, {g_2}^{w_i}, c^{\mathsf{pw}_i} {g_3}^{v_i + w_i})$

$\Pi_i^2 = \mathsf{SSNIZKEq}_{g,c}(C'_i, A_i)$ $\quad \xrightarrow{C'_i, A_i, \Pi_i^2}$

Blind. Ring

$(2c) = (1c)$ $\quad \xrightarrow{\gamma_i^{(1)}, \gamma_i^{(2)}, \Pi_i^2}$

$(2d) = (1d)$ $\quad \xrightarrow{(G_{1,i}, G_{2,i}, G_{3,i}, G_{4,i})}$

Unblind. Ring

$(2e) = (1e)$ $\quad \xrightarrow{\zeta_i}$

$(2f) = (1f) \qquad \mathsf{pk} \stackrel{?}{=} \zeta_1$

Blinding Ring

$(3a)$ abort if one of the signatures σ_i is invalid

$\delta_0^{(0)} = \prod_i {A_i}^{(3)} = c^{\sum_i \mathsf{pw}_i} {g_3}^{\sum_i v_i + \sum_i w_i} \quad \delta_0^{(1)} = g_1 \quad \delta_0^{(2)} = g_2 \quad \delta_0^{(3)} = g_3$

P_1 chooses at random $\beta_1 \xleftarrow{R} \mathbb{Z}_p^*$ and computes

$\quad \delta_1^{(0)} = (\delta_0^{(0)})^{\beta_1} \quad \delta_1^{(1)} = (\delta_0^{(1)})^{\beta_1} \quad \delta_1^{(2)} = (\delta_0^{(2)})^{\beta_1} \quad \delta_1^{(3)} = (\delta_0^{(3)})^{\beta_1}$

This round is done sequentially, for i=2,...,n.

Upon receiving $(\delta_j^{(0)}, \delta_j^{(1)}, \delta_j^{(2)}, \delta_j^{(3)})$ for $j = 1, \ldots, i-1$,

check $\mathsf{CDH}(\delta_{j-1}^{(0)}, \delta_j^{(0)}, \delta_{j-1}^{(1)}, \delta_j^{(1)}), \mathsf{CDH}(\delta_{j-1}^{(0)}, \delta_j^{(0)}, \delta_{j-1}^{(2)}, \delta_j^{(2)}),$

$\quad \mathsf{CDH}(\delta_{j-1}^{(0)}, \delta_j^{(0)}, \delta_{j-1}^{(3)}, \delta_j^{(3)})$; abort if one of them is invalid

$\beta_i \xleftarrow{R} \mathbb{Z}_p^*$

$\delta_i^{(0)} = (\delta_{i-1}^{(0)})^{\beta_i} \quad \delta_i^{(1)} = (\delta_{i-1}^{(1)})^{\beta_i} \quad \delta_i^{(2)} = (\delta_{i-1}^{(2)})^{\beta_i} \quad \delta_i^{(3)} = (\delta_{i-1}^{(3)})^{\beta_i}$ $\quad \xrightarrow{\delta_i^{(1)}, \delta_i^{(2)}}$

$(3b)$ given $\delta_n^{(0)} = c^{\beta \sum_i \mathsf{pw}_i} {g_3}^{\beta(\sum_i v_i + \sum_i w_i)} \quad \delta_n^{(1)} = {g_1}^{\beta} \quad \delta_n^{(2)} = {g_2}^{\beta} \quad \delta_n^{(3)} = {g_3}^{\beta}$

$\quad$ check $\mathsf{CDH}(\delta_{n-1}^{(0)}, \delta_n^{(0)}, \delta_{n-1}^{(1)}, \delta_n^{(1)}), \mathsf{CDH}(\delta_{n-1}^{(0)}, \delta_n^{(0)}, \delta_{n-1}^{(2)}, \delta_n^{(2)})$

$\quad \mathsf{CDH}(\delta_{n-1}^{(0)}, \delta_n^{(0)}, \delta_{n-1}^{(3)}, \delta_n^{(3)})$

for $i \neq 1, P_i$ computes $G'_{1,i} = (\delta_n^{(1)})^{v_i}, G'_{2,i} = (\delta_n^{(2)})^{w_i},$

$\quad G'_{3,i} = (\delta_n^{(3)})^{v_i}, G'_{4,i} = (\delta_n^{(3)})^{w_i}$ $\quad \xrightarrow{G'_{1,i}, G'_{2,i}, G'_{3,i}, G'_{4,i}}$

Unblinding Ring

$(3c)$ given, for $j = 1, \ldots, n \qquad G'_{1,j}, G'_{2,j}, G'_{3,j}, G'_{4,j}$

$\quad$ check $\mathsf{CDH}(g_1, {A_j}^{(1)}, \delta_n^{(1)}, G'_{1,j}), \mathsf{CDH}(g_2, {A_j}^{(2)}, \delta_n^{(2)}, G'_{2,j}),$

$\quad \mathsf{CDH}(\delta_n^{(1)}, G'_{1,j}, \delta_n^{(3)}, G'_{3,j})$ and $\mathsf{CDH}(\delta_n^{(2)}, G'_{2,j}, \delta_n^{(3)}, G'_{4,j})$

P_1 computes $G'_{1,1} = (\delta_n^{(1)})^{v_1}, G'_{2,1} = (\delta_n^{(2)})^{w_1}, G'_{3,1} = (\delta_n^{(3)})^{v_1}, G'_{4,1} = (\delta_n^{(3)})^{w_1}$

P_1 chooses at random $x \xleftarrow{R} \mathbb{Z}_p^*$

$\quad$ and computes $\zeta'_{n+1} = \left(\delta_n^{(0)} / \prod_j (G'_{3,j} G'_{4,j})\right)^x = c^{\beta x \sum_j \mathsf{pw}_j}$ $\quad \xrightarrow{\zeta'_{n+1}}$

This round is done sequentially, for i from n down to 2.

Upon receiving, for j from n down to $i+1 \qquad \zeta'_j$

$\quad$ check $\mathsf{CDH}(\delta_{j-1}^{(1)}, \delta_j^{(1)}, \zeta'_j, \zeta'_{j+1})$

$\zeta'_i = (\zeta'_{i+1})^{1/\beta_i}$ $\quad \xrightarrow{\zeta'_i}$

$(3d)$ given, for j from $i-1$ down to $2 \quad \zeta'_j$

$\quad$ check $\mathsf{CDH}(\gamma_{j-1}^{(1)}, \gamma_j^{(1)}, \zeta'_j, \zeta'_{j+1})$

P_1 gets $\zeta'_1 = (\zeta'_2)^{1/\beta_1} = c^{x \sum \mathsf{pw}_i} = c^{x\mathsf{sk}}$ and finally c^{sk}

Fig. 2. Individual steps of the distributed decryption protocol

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 the full version [8].

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.

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 $(C_1 U_1^{-1}, C_2 U_2^{-1}, C_3 g^{-1})$ yields a proof that $x \in \{0, 1\}$, thus implements $\mathsf{WIProofBit}$, in an efficient way and without random oracles.

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[4] 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 $(\mathsf{vk}_{\mathrm{ot}}, \mathsf{sk}_{\mathrm{ot}})$ for Σ_{ot}, proves a statement and, at the end, adds a signature under $\mathsf{vk}_{\mathrm{ot}}$ 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 $\mathsf{vk}_{\mathrm{ot}}$ valid under vk. Groth [12] shows how to construct a new set of equations which is satisfiable iff $\{E_k\}$ or $\{V_k(\mathsf{vk}, \mathsf{vk}_{\mathrm{ot}}, \cdot)\}$ 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 $(\mathsf{vk}_{\mathrm{ot}}, \mathsf{sk}_{\mathrm{ot}})$, one uses the trapdoor to produce a signature σ on $\mathsf{vk}_{\mathrm{ot}}$ valid under vk and uses σ as a witness

[4] 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 $(m^*, \sigma^*) \neq (m, \sigma)$.

for $\{V_k(\mathsf{vk}, \mathsf{vk}_{\mathrm{ot}}, \cdot)\}$. (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 $(\mathsf{vk}^*_{\mathrm{ot}}, \mathsf{sk}^*_{\mathrm{ot}})$ (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 $\mathsf{vk}_{\mathrm{ot}}$. 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 $\boldsymbol{h} := (h_0, h_1, \dots, h_\ell) \leftarrow \mathbb{G}^{\ell+1}$. 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

$$e(\sigma_1, g)\, e(\mathcal{W}(M), \sigma_2) = e(f, X) \tag{1}$$

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 CRS is a (binding) key for linear commitments to group elements. The proof consists of commitments to each X_i and 9 elements 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 $m_2 = d^\mu$.

$$e(C_1, g_3) = e(g_1, a_1) \quad e(C_2, g_3) = e(g_2, a_2) \tag{2}$$

$$e(D_1, g_3) = e(g_1, a_3) \quad e(D_2, g_3) = e(g_2, a_4) \quad e(C_3 a_1^{-1} a_2^{-1}, d) = e(c, D_3 a_3^{-1} a_4^{-1})$$

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 (2) or (1), i.e., if we either know a satisfying (2) or σ satisfying (1). 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 (2) and (1), therefore termed "$(2 \vee 1)$".

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

From (1): $e(\chi_2, \psi_1^{-1}\sigma_1) = 1 \quad e(\chi_2, \psi_2^{-1}\mathcal{W}(M)) = 1 \quad e(\chi_2, \psi_3^{-1} f) = 1$

$$e(\psi_1, g)\, e(\psi_2, \sigma_2)\, e(\psi_3, X)^{-1} = 1$$

From (2):

$$e(\chi_1, \phi_1^{-1} g_1) = 1 \qquad e(\chi_1, \phi_2^{-1} g_2) = 1$$

$$e(\chi_1, \phi_3^{-1} g_3) = 1 \quad e(\chi_1, \phi_4^{-1} c) = 1 \quad e(\chi_1, \phi_5^{-1} d) = 1$$

$$e(C_1, \phi_3)\, e(\phi_1, a_1)^{-1} = 1 \qquad e(C_2, \phi_3)\, e(\phi_2, a_2) = 1$$

$$e(D_1, \phi_3)\, e(\phi_1, a_3)^{-1} = 1 \qquad e(D_2, \phi_3)\, e(\phi_2, a_4) = 1$$

$$e(C_3 a_1^{-1} a_2^{-1}, \phi_5)\, e(\phi_4, D_3 a_3^{-1} a_4^{-1}) = 1$$

Completeness. To produce a proof we proceed as follows: If we have an assignment a for (2), 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 (2) are satisfied, because a is a witness for (2). Since $\chi_2 = 1$, we can set $\psi_i := 1$ (for all i) as well, which satisfies the block for (1), no matter what value we set σ.

On the other hand, if we know a signature σ satisfying (1), 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 $(2 \vee 1)$ is satisfied then either a satisfies (2) or σ satisfies (1): 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 (2) (by the last five equations of the block for (2)) or σ satisfies (1) (by the last equation of the block for (1)).

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

$$\begin{array}{lll} e(a_1, U_2) = e(U_1, a_2) & e(C_1 a_1^{-1}, g_3) = e(g_1, a_4) & e(D_1, g_3) = e(g_1, a_5) \\ e(a_1, g) = e(U_1, a_3) & e(C_2 a_2^{-1}, g_3) = e(g_2, C_3 a_3^{-1} a_4^{-1}) & e(D_2, g_3) = e(g_2, D_3 a_3^{-1} a_5^{-1}) \end{array} \quad (3)$$

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 (3) and (1) to a set $(3 \vee 1)$ analogously to the construction of $(2 \vee 1)$, 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 $(2 \vee 1)$ by $(3 \vee 1)$.

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 (2). 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 $(2 \vee 1)$ 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 $(2 \vee 1)$.

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-2007-216646 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. X. Boyen, C. Chevalier, G. Fuchsbauer, and D. Pointcheval. Strong cryptography from weak secrets: Building efficient PKE and IBE from distributed passwords. In *Africacrypt 2010*, 2010. Full version available from the web page of the authors.
9. 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.
10. 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.
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.