This is the Full Version of the Extended Abstract that appears
in the Proceedings of the 16th International Conference on Practice and Theory in Public-Key Cryptography (PKC '13)
(26 February – 1 March 2013, Nara, Japan)
Kaoru Kurosawa Ed., Springer-Verlag, LNCS ????, pages ???–???.

Efficient UC-Secure Authenticated Key-Exchange for Algebraic Languages

Fabrice Ben Hamouda[1], Olivier Blazy[2], Céline Chevalier[3], David Pointcheval[1], and Damien Vergnaud[1]

[1] ENS, Paris, France †
[2] Ruhr-Universität Bochum, Germany
[3] Université Panthéon-Assas, Paris, France

Abstract *Authenticated Key Exchange* (AKE) protocols enable two parties to establish a shared, cryptographically strong key over an insecure network using various authentication means, such as cryptographic keys, short (*i.e.*, low-entropy) secret keys or *credentials*. In this paper, we provide a general framework, that encompasses several previous AKE primitives such as *(Verifier-based) Password-Authenticated Key Exchange* or *Secret Handshakes*, we call *LAKE* for *Language-Authenticated Key Exchange*.
We first model this general primitive in the *Universal Composability* (UC) setting. Thereafter, we show that the Gennaro-Lindell approach can efficiently address this goal. But we need *smooth projective hash functions* on new languages, whose efficient implementations are of independent interest. We indeed provide such hash functions for languages defined by combinations of linear pairing product equations.
Combined with an efficient commitment scheme, that is derived from the highly-efficient UC-secure Lindell's commitment, we obtain a very practical realization of Secret Handshakes, but also *Credential-Authenticated Key Exchange protocols*. All the protocols are UC-secure, in the standard model with a common reference string, under the classical Decisional Linear assumption.

1 Introduction

The main goal of an *Authenticated Key Exchange* (AKE) protocol is to enable two parties to establish a shared cryptographically strong key over an insecure network under the complete control of an adversary. AKE is one of the most widely used and fundamental cryptographic primitives. In order for AKE to be possible, the parties must have authentication means, *e.g.* (public or secret) cryptographic keys, short (*i.e.*, low-entropy) secret keys or *credentials* that satisfy a (public or secret) policy.

Motivation. PAKE, for *Password-Authenticated Key Exchange*, was formalized by Bellovin and Merritt [BM92] and followed by many proposals based on different cryptographic assumptions (see [ACP09, CCGS10] and references therein). It allows users to generate a strong cryptographic key based on a shared "human-memorable" (*i.e.* low-entropy) password without requiring a public-key infrastructure. In this setting, an adversary controlling all communication in the network should not be able to mount an off-line dictionary attack.

The concept of *Secret Handshakes* has been introduced in 2003 by Balfanz, Durfee, Shankar, Smetters, Staddon and Wong [BDS+03] (see also [JL09, AKB07]). It allows two members of the same group to identify each other secretly, in the sense that each party reveals his affiliation to the other only if they are members of the same group. At the end of the protocol, the parties can set up an ephemeral session key for securing further communication between them and an outsider is unable to determine if the handshake succeeded. In case of failure, the players do not learn any information about the other party's affiliation.

More recently, *Credential-Authenticated Key Exchange* (CAKE) was presented by Camenisch, Casati, Groß and Shoup [CCGS10]. In this primitive, a common key is established if and only if a specific relation is satisfied between credentials hold by the two players. This primitive includes variants of PAKE and Secret Handshakes, and namely Verifier-based PAKE, where the client owns a password pw and the server knows a one-way transformation v of the password only. It prevents massive password recovering in case of server corruption. The two players eventually agree on a common high entropy secret if and only if pw and v match together, and off-line dictionary attacks are prevented for third-party players.

Our Results. We propose a new primitive that encompasses most of the previous notions of authenticated key exchange. It is closely related to CAKE and we call it LAKE, for *Language-Authenticated Key-Exchange*, since parties establish a common key if and only if they hold credentials that belong to specific (and possibly independent) languages. The definition of the primitive is more practice-oriented than the definition of CAKE from [CCGS10] but the two notions are very similar. In particular, the new primitive enables privacy-preserving

† ENS, CNRS & INRIA – UMR 8548

authentication and key exchange protocols by allowing two members of the same group to secretly and privately authenticate to each other without revealing this group beforehand.

In order to define the security of this primitive, we use the UC framework and an appropriate definition for languages that permits to dissociate the public part of the policy, the private common information the users want to check and the (possibly independent) secret values each user owns that assess the membership to the languages. We provide an ideal functionality for LAKE and give efficient realizations of the new primitive (for a large family of languages) secure under classical mild assumptions, in the standard model (with a common reference string – CRS), with static corruptions.

We significantly improve the efficiency of several CAKE protocols [CCGS10] for specific languages and we enlarge the set of languages for which we can construct practical schemes. Notably, we obtain a very practical realization of Secret Handshakes and a Verifier-based Password-Authenticated Key Exchange.

Our Techniques. A general framework to design PAKE in the CRS model was proposed by Gennaro and Lindell [GL03] in 2003. This approach was applied to the UC framework by Canetti, Halevi, Katz, Lindell, and MacKenzie [CHK$^+$05], and improved by Abdalla, Chevalier and Pointcheval [ACP09]. It makes use of the *smooth projective hash functions* (SPHF), introduced by Cramer and Shoup [CS02]. Such a hashing family is a family of hash functions that can be evaluated in two ways: using the (secret) hashing key, one can compute the function on every point in its domain, whereas using the (public) *projection* key one can only compute the function on a special subset of its domain. Our first contribution is the description of smooth projective hash functions for new interesting languages: Abdalla, Chevalier and Pointcheval [ACP09] explained how to make disjunctions and conjunctions of languages, we study here languages defined by linear pairing product equations on committed values.

In 2011, Lindell [Lin11] proposed a highly-efficient commitment scheme, with a non-interactive opening algorithm, in the UC framework. We will not use it in black-box, but instead we will patch it to make the initial Gennaro and Lindell's approach to work, without zero-knowledge proofs [CHK$^+$05], using the equivocability of the commitment.

Language Definition. In [ACP09], Abdalla *et al.* already formalized languages to be considered for SPHF. But, in the following, we will use a more simple formalism, which is nevertheless more general: we consider any efficiently computable binary relation $\mathcal{R} : \{0,1\}^* \times \mathcal{P} \times \mathcal{S} \to \{0,1\}$, where the additional parameters $\mathsf{pub} \in \{0,1\}^*$ and $\mathsf{priv} \in \mathcal{P}$ define a language $L_{\mathcal{R}}(\mathsf{pub}, \mathsf{priv}) \subseteq \mathcal{S}$ of the words W such that $\mathcal{R}(\mathsf{pub}, \mathsf{priv}, W) = 1$:

- pub are public parameters;
- priv are private parameters the two players have in mind, and they should think to the same values: they will be committed to, but never revealed;
- W is the word the sender claims to know in the language: it will be committed to, but never revealed.

Our LAKE primitive, specific to two relations $\mathcal{R}_a$ and $\mathcal{R}_b$, will allow two users, Alice and Bob, owning a word $W_a \in L_{\mathcal{R}_a}(\mathsf{pub}, \mathsf{priv}_a)$ and $W_b \in L_{\mathcal{R}_b}(\mathsf{pub}, \mathsf{priv}_b)$ respectively, to agree on a session key under some specific conditions: they first both agree on the public parameter pub, Bob will think about priv'_a for his expected value of priv_a, Alice will do the same with priv'_b for priv_b; eventually, if $\mathsf{priv}'_a = \mathsf{priv}_a$ and $\mathsf{priv}'_b = \mathsf{priv}_b$, and if they both know words in the languages, then the key agreement will succeed. In case of failure, no information should leak about the reason of failure, except the inputs did not satisfy the relations $\mathcal{R}_a$ or $\mathcal{R}_b$, or the languages were not consistent.

We stress that each LAKE protocol will be specific to a pair of relations $(\mathcal{R}_a, \mathcal{R}_b)$ describing the way Alice and Bob will authenticate to each other. This pair of relations $(\mathcal{R}_a, \mathcal{R}_b)$ specifies the sets $\mathcal{P}_a$, $\mathcal{P}_b$ and $\mathcal{S}_a$, $\mathcal{S}_b$ (to which the private parameters and the words should respectively belong). Therefore, the formats of priv_a, priv_b and W_a and W_b are known in advance, but not their values. When $\mathcal{R}_a$ and $\mathcal{R}_b$ are clearly defined from the context (e.g., PAKE), we omit them in the notations. For example, these relations can formalize:

- Password authentication: The language is defined by $\mathcal{R}(\mathsf{pub}, \mathsf{priv}, W) = 1 \Leftrightarrow W = \mathsf{priv}$, and thus $\mathsf{pub} = \emptyset$. The classical setting of PAKE requires the players A and B to use the same password W, and thus we should have $\mathsf{priv}_a = \mathsf{priv}'_b = \mathsf{priv}_b = \mathsf{priv}'_a = W_a = W_b$;
- Signature authentication: $\mathcal{R}(\mathsf{pub}, \mathsf{priv}, W) = 1 \Leftrightarrow \mathsf{Verif}(\mathsf{pub}_1, \mathsf{pub}_2, W) = 1$, where $\mathsf{pub} = (\mathsf{pub}_1 = \mathsf{vk}, \mathsf{pub}_2 = M)$ and $\mathsf{priv} = \emptyset$. The word W is thus a signature of M valid under vk, both specified in pub;

– Credential authentication: we can consider any mix for vk and M in pub or priv, and even in W, for which the relation $\mathcal{R}$ verifies the validity of the signature. When M and vk are in priv or W, we achieve *affiliation-hiding* property.

In the two last cases, the parameter pub can thus consist of a message on which the user is expected to know a signature valid under vk: either the user knows the signing key and can generate the signature on the fly to run the protocol, or the user has been given signatures on some messages (credentials). As a consequence, we just assume that, after having publicly agreed on a common pub, the two players have valid words in the appropriate languages. The way they have obtained these words does not matter.

Following our generic construction, private elements will be committed using encryption schemes, derived from Cramer-Shoup's scheme, and will thus have to be first encoded as n-tuples of elements in a group $\mathbb{G}$. In the case of PAKE, authentication will check that a player knows an appropriate password. The relation is a simple equality test, and accepts for one word only. A random commitment (and thus of a random group element) will succeed with negligible probability. For signature-based authentication, the verification key can be kept secret, but the signature should be unforgeable and thus a random word W should quite unlikely satisfy the relation. We will often make this assumption on useful relations $\mathcal{R}$: for any pub, $\{(\mathsf{priv}, W) \in \mathcal{P} \times \mathcal{S}, \mathcal{R}(\mathsf{pub}, \mathsf{priv}, W) = 1\}$ is sparse (negligible) in $\mathcal{P} \times \mathcal{S}$, and *a fortiori* in the set $\mathbb{G}^n$ in which elements are first embedded.

2 Definitions

In this section, we first briefly recall the notations and the security notions of the basic primitives we will use in the rest of the paper, and namely public key encryption and signature. More formal definitions, together with the classical computational assumptions (CDH, DDH, and DLin) are provided in the Appendix A.1: A public-key encryption scheme is defined by four algorithms: $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$, $c \leftarrow \mathsf{Encrypt}(\mathsf{ek}, m; r)$, and $m \leftarrow \mathsf{Decrypt}(\mathsf{dk}, c)$. We will need the classical notion of IND-CCA security. A signature scheme is defined by four algorithms: $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$, $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m; s)$, and $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$. We will need the classical notion of EUF-CMA security. In both cases, the global parameters param will be ignored, included in the CRS. We will furthermore make use of collision-resistant hash function families.

2.1 Universal Composability

Our main goal will be to provide protocols with security in the universal composability framework. The interested reader is referred to [Can01, CHK+05] for details. More precisely, we will work in the UC framework with joint state proposed by Canetti and Rabin [CR03] (with the CRS as the joint state). Since players are not individually authenticated, but just afterward if the credentials are mutually consistent with the two players' languages, the adversary will be allowed to interact on behalf of any player from the beginning of the protocol, either with the credentials provided by the environment (static corruption) or without (impersonation attempt). As with the Split Functionality [BCL+05], according to whom sends the first flow for a player, either the player itself or the adversary, we know whether this is an honest player or a dishonest player (corrupted or impersonation attempt, but anyway controlled by the adversary). Then, our goal will be to prove that the best an adversary can do is to try to play against one of the other players, as an honest player would do, with a credential it guessed or obtained in any possible way. This is exactly the so-called one-line dictionary attack when one considers PAKE protocols. In the adaptive corruption setting, the adversary could get complete access to the private credentials and the internal memory of an honest player, and then get control of it, at any time. But we will restrict to the static corruption setting in this paper. It is enough to deal with most of the concrete requirements: related credentials, arbitrary compositions, and forward-secrecy. To achieve our goal, for a UC-secure LAKE, we will use some other primitives which are secure in the classical setting only.

2.2 Commitment

Commitments allow a user to commit to a value, without revealing it, but without the possibility to later change his mind. It is composed of three algorithms: $\mathsf{Setup}(1^k)$ generates the system parameters, according to a security parameter k; $\mathsf{Commit}(\ell, m; r)$ produces a commitment c on the input message $m \in \mathcal{M}$ using the random coins

$r \xleftarrow{\$} \mathcal{R}$, under the label ℓ, and the opening information d; while $\mathsf{Decommit}(\ell, c, m, d)$ opens the commitment c with the message m and the opening information d that proves the correct opening under the label ℓ.

Such a commitment scheme should be both *hiding*, which says that the commit phase does not leak any information about m, and *binding*, which says that the decommit phase should not be able to open to two different messages. Additional features will be required in the following, such as non-malleability, extractability, and equivocability. We also included a label ℓ, which can be empty or an additional public information that has to be the same in both the commit and the decommit phases. A labeled commitment that is both non-malleable and extractable can be instantiated by an IND-CCA labeled encryption scheme (see the Appendix A.1). We will use the Linear Cramer-Shoup encryption scheme [Sha07, CKP07]. We will then patch it, using a technique inspired from [Lin11], to make it additionally equivocable (see Section 3). It will have an interactive commit phase, in two rounds: $\mathsf{Commit}(\ell, m; r)$ and a challenge ε from the receiver, which will define an implicit full commitment to be open latter.

2.3 Smooth Projective Hash Functions

Smooth projective hash function (SPHF) systems have been defined by Cramer and Shoup [CS02] in order to build a chosen-ciphertext secure encryption scheme. They have thereafter been extended [GL03, ACP09, BPV12] and applied to several other primitives. Such a system is defined on a language L, with five algorithms:

- $\mathsf{Setup}(1^k)$ generates the system parameters, according to a security parameter k;
- $\mathsf{HashKG}(L)$ generates a hashing key hk for the language L;
- $\mathsf{ProjKG}(\mathsf{hk}, L, W)$ derives the projection key hp, possibly depending on a word W;
- $\mathsf{Hash}(\mathsf{hk}, L, W)$ outputs the hash value from the hashing key;
- $\mathsf{ProjHash}(\mathsf{hp}, L, W, w)$ outputs the hash value from the projection key and the witness w that $W \in L$.

The correctness of the scheme assures that if W is in L with w as a witness, then the two ways to compute the hash values give the same result: $\mathsf{Hash}(\mathsf{hk}, L, W) = \mathsf{ProjHash}(\mathsf{hp}, L, W, w)$. In our setting, these hash values will belong to a group $\mathbb{G}$. The security is defined through two different notions: the *smoothness* property guarantees that if $W \notin L$, the hash value is *statistically* indistinguishable from a random element, even knowing hp; the *pseudo-randomness* property guarantees that even for a word $W \in L$, but without the knowledge of a witness w, the hash value is *computationally* indistinguishable from a random element, even knowing hp.

3 Double Linear Cramer-Shoup Encryption (DLCS)

As explained earlier, any IND-CCA labeled encryption scheme can be used as a non-malleable and extractable labeled commitment scheme: one could use the Cramer-Shoup encryption scheme (see the Appendix A.4), but we will focus on the DLin-based primitives, and thus the Linear Cramer-Shoup scheme (see the Appendix A.3), we call LCS. Committed/encrypted elements will either directly be group elements, or bit-strings on which we apply a reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$. In order to add the equivocability, one can use a technique inspired from [Lin11]. See the Appendix B for more details, but we briefly present the commitment scheme we will use in the rest of this paper in conjunction with SPHF.

Linear Cramer-Shoup Commitment Scheme. The parameters, in the CRS, are a group $\mathbb{G}$ of prime order p, with three independent generators $(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3$, a collision-resistant hash function $\mathfrak{H}_K$, and possibly an additional reversible mapping $\mathcal{G}$ from $\{0,1\}^n$ to $\mathbb{G}$ to commit bit-strings. From 9 scalars $(x_1, x_2, x_3,\ y_1, y_2, y_3,\ z_1, z_2, z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, one also sets, for $i = 1, 2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. The public parameters consist of the encryption key $\mathsf{ek} = (\mathbb{G},\ g_1, g_2, g_3,\ c_1, c_2,\ d_1, d_2,\ h_1, h_2,\ \mathfrak{H}_K)$, while the trapdoor for extraction is $\mathsf{dk} = (x_1, x_2, x_3,\ y_1, y_2, y_3,\ z_1, z_2, z_3)$. One can define the encryption process:

$$\mathsf{LCS}(\ell, \mathsf{ek}, M; r, s) \stackrel{\mathsf{def}}{=} (\mathbf{u} = (g_1^r, g_2^s, g_3^{r+s}), e = M \cdot h_1^r h_2^s, v = (c_1 d_1^{\xi})^r (c_2 d_2^{\xi})^s)$$

where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$. When ξ is specified from outside, one additionally denotes it $\mathsf{LCS}^*(\ell, \mathsf{ek}, M, \xi; r, s)$. The commitment to a message $M \in \mathbb{G}$, or $M = \mathcal{G}(m)$ for $m \in \{0,1\}^n$, encrypts M under ek: $\mathsf{LCSCom}(\ell, M; r, s) \stackrel{\mathsf{def}}{=} \mathsf{LCS}(\ell, \mathsf{ek}, M; r, s)$. The decommit process consists of M and (r, s) to check the correctness of the encryption. It is possible to do implicit verification, without any decommit information, but just an SPHF on the language of the ciphertexts of M that is privately shared by the two players. Since the underlying encryption scheme is IND-CCA, this commitment scheme is non-malleable and extractable.

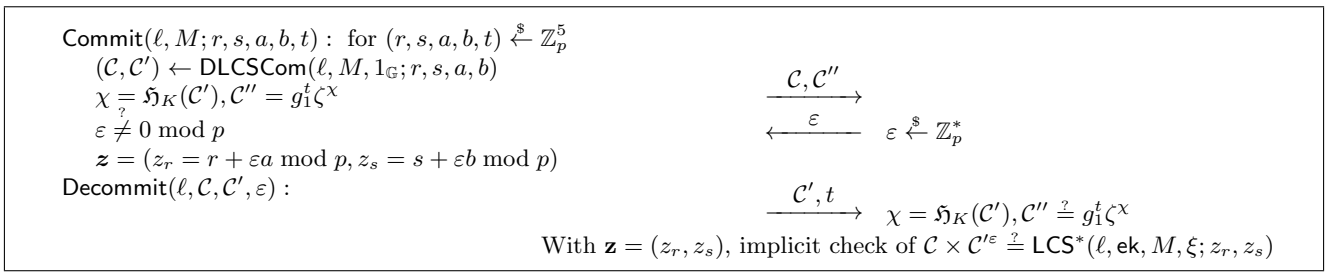

Figure 1. DLCSCom′ Commitment Scheme for SPHF

Double Linear Cramer-Shoup Commitment Schemes. To make it equivocable, we double the commitment process, in two steps. The CRS additionally contains a scalar $\aleph \xleftarrow{\$} \mathbb{Z}_p$, one also sets, $\zeta = g_1^\aleph$. The trapdoor for equivocability is $\aleph$. The Double Linear Cramer-Shoup encryption scheme, denoted DLCS and detailed in the Appendix B is

$$\mathsf{DLCS}(\ell, \mathsf{ek}, M, N; r, s, a, b) \stackrel{\text{def}}{=} (\mathcal{C} \leftarrow \mathsf{LCS}(\ell, \mathsf{ek}, M; r, s), \mathcal{C}' \leftarrow \mathsf{LCS}^*(\ell, \mathsf{ek}, N, \xi; a, b))$$

where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ is computed during the generation of $\mathcal{C}$ and transfered for the generation of $\mathcal{C}'$. As above, we denote DLCSCom denotes the use of DLCS with the encryption key ek. The usual commit/decommit processes are described on Figure 6 in the Appendix B. On Figure 1, one can find the DLCSCom′ scheme where one can implicitly check the opening with an SPHF. These two constructions essentially differ with $\chi = \mathfrak{H}_K(\mathcal{C}')$ (for the SPHF implicit check) instead of $\chi = \mathfrak{H}_K(M, \mathcal{C}')$ (for the explicit check). We stress that with this alteration, the DLCSCom′ scheme is not a real commitment scheme (not formally extractable/binding): in DLCSCom′, the sender can indeed encrypt M in $\mathcal{C}$ and $N \neq 1_{\mathbb{G}}$ in $\mathcal{C}'$, and then, the global ciphertext $\mathcal{C} \times \mathcal{C}'^\varepsilon$ contains $M' = MN^\varepsilon \neq M$, whereas one would have extracted M from $\mathcal{C}$. But M' is unknown before ε is sent, and thus, if one checks the membership of M' to a sparse language, it will unlikely be true.

Multi-Message Schemes. One can extend these encryption and commitment schemes to vectors of n messages (see the Appendix B). We will denote them n-DLCSCom′ or n-DLCSCom for the commitment schemes. They consist in encrypting each message with independent random coins in $\mathcal{C}_i = (\mathbf{u}_i, e_i, v_i)$ but the same $\xi = \mathfrak{H}_K(\ell, (\mathbf{u}_i), (e_i))$, together with independent companion ciphertexts $\mathcal{C}'_i$ of $1_{\mathbb{G}}$, still with the same ξ for the doubled version. In the latter case, n independent challenges $\varepsilon_i \xleftarrow{\$} \mathbb{Z}_p^*$ are then sent to lead to the full commitment $(\mathcal{C}_i \times \mathcal{C}_i'^{\varepsilon_i})$ with random coins $z_{r_i} = r_i + \varepsilon_i a_i$ and $z_{s_i} = s_i + \varepsilon_i b_i$. Again, if one of the companion ciphertext $\mathcal{C}'_i$ does not encrypt $1_{\mathbb{G}}$, the full commitment encrypts a vector with at least one unpredictable component M'_i. Several non-unity components in the companion ciphertexts would lead to independent components in the full commitment. For languages sparse enough, this definitely turns out not to be in the language.

4 SPHF for Implicit Proofs of Membership

In [ACP09], Abdalla *et al.* presented a way to compute a conjunction or a disjunction of languages by some simple operations on their projection keys. Therefore all languages presented afterward can easily be combined together. However as the original set of manageable languages was not really developed, we are going to present several steps to extend it, and namely in order to cover some languages useful in various AKE instantiations.

We will show that almost all the vast family of languages covered by the Groth-Sahai methodology [GS08] can be addressed by our approach too. More precisely, we can handle all the linear pairing product equations, when witnesses are committed using our above (multi-message) DLCSCom′ commitment scheme, or even the non-equivocable LCSCom version. This will be strong enough for our applications. For using them in black-box to build our LAKE protocol, one should note that the projection key is computed from the ciphertext $\mathcal{C}$ when using the simple LCSCom commitment, but also when using the DLCSCom′ version. The full commitment $\mathcal{C} \times \mathcal{C}'^\varepsilon$ is not required, but ξ only, which is known as soon as $\mathcal{C}$ is given (or the vector $(\mathcal{C}_i)_i$ for the multi-message version). Of course, the hash value will then depend on the full commitment (either $\mathcal{C}$ for the LCSCom commitment, or $\mathcal{C} \cdot \mathcal{C}'^\varepsilon$ for the DLCSCom′ commitment).

This will be relevant to our AKE problem: equality of two passwords, in PAKE protocols; corresponding signing/verification keys associated with a valid signature on a pseudonym or a hidden identity, in secret

handshakes; valid credentials, in CAKE protocols. All those tests are quite similar: one has to show that the ciphertexts are valid and that the plaintexts satisfy the expected relations in a group. We first illustrate that with commitments of Waters signatures of a public message under a committed verification key. We then explain the general method. The formal proofs are provided in the Appendix C.

4.1 Commitments of Signatures

Let us consider the Waters signature [Wat05] in a symmetric bilinear group, as reviewed in the Appendix A.3, and then we just need to recall that, in a pairing-friendly setting $(p, \mathbb{G}, \mathbb{G}_T, e)$, with public parameters $(\mathcal{F}, g, h)$, and a verification key vk, a signature $\sigma = (\sigma_1, \sigma_2)$ is valid with respect to the message M under the key vk if it satisfies $e(\sigma_1, g) = e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2)$.

A similar approach has already been followed in [BPV12], however not with a Linear Cramer-Shoup commitment scheme, nor with such general languages. We indeed first consider the language of the signatures $(\sigma_1, \sigma_2) \in \mathbb{G}^2$ of a message $M \in \{0,1\}^k$ under the verification key $\mathsf{vk} \in \mathbb{G}$, where M is public but vk is private: $L(\mathsf{pub}, \mathsf{priv})$, where $\mathsf{priv} = \mathsf{vk}$ and $\mathsf{pub} = M$. One will thus commit the pair $(\mathsf{vk}, \sigma_1) \in \mathbb{G}^2$ with the label $\ell = (M, \sigma_2)$ using a 2-$\mathsf{DLCSCom}'$ commitment and then prove the commitment actually contains (vk, σ_1) such that $e(\sigma_1, g) = e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2)$. We insist on the fact that σ_1 only has to be encrypted, and not σ_2, in order to hide the signature, since the latter σ_2 is a random group element. If one wants unlinkability between signature commitments, one simply needs to re-randomize (σ_1, σ_2) before encryption. Hence σ_2 can be sent in clear, but bounded to the commitment in the label, together with the pub part of the language. In order to prove the above property on the committed values, we will use conjunctions of SPHF: first, to show that each commitment is well-formed (valid ciphertexts), and then that the associated plaintexts verify the linear pairing equation, where the committed values are underlined: $e(\underline{\sigma_1}, g) = e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2)$ Note that vk is not used as a committed value for this verification of the membership of σ to the language since this is the verification key expected by the verifier, specified in the private part priv, which has to be independently checked with respect to the committed verification key. This is enough for the affiliation-hiding property. We could consider the similar language where $M \in \{0,1\}^k$ is in the word too: $e(\underline{\sigma_1}, g) = e(h, \mathsf{vk}) \cdot e(\underline{\mathcal{F}(M)}, \sigma_2)$, and then one should commit M, bit-by-bit, and then use a $(k+2)$-$\mathsf{DLCSCom}'$ commitment.

4.2 Linear Pairing Product Equations

Instead of describing in details the SPHF for the above examples, let us show it for a more general framework: we considered

$$e(\underline{\sigma_1}, g) = e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2) \text{ or } e(\underline{\sigma_1}, g) = e(h, \mathsf{vk}) \cdot e(\underline{\mathcal{F}(M)}, \sigma_2),$$

where the unknowns are underlined. These are particular instantiations of t simultaneous equations

$$\left(\prod_{i \in A_k} e(\underline{\mathcal{Y}_i}, \mathcal{A}_{k,i}) \right) \cdot \left(\prod_{i \in B_k} \underline{\mathcal{Z}_i}^{\mathfrak{z}_{k,i}} \right) = \mathcal{B}_k, \text{ for } k = 1, \ldots, t,$$

where $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, and $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, as well as $A_k \subseteq \{1, \ldots, m\}$ and $B_k \subseteq \{m+1, \ldots, n\}$ are public, but the $\mathcal{Y}_i \in \mathbb{G}$ and $\mathcal{Z}_i \in \mathbb{G}_T$ are simultaneously committed using the multi-message $\mathsf{DLCSCom}'$ or LCSCom commitments scheme, in $\mathbb{G}$ or $\mathbb{G}_T$ respectively. This is more general than the relations covered by [CCGS10], since one can also commit scalars bit-by-bit. In the Appendix C.4, we detail how to build the corresponding SPHF, and prove the soundness of our approach. For the sake of clarity, we focus here to a single equation only, since multiple equations are just conjunctions. We can even consider the simpler equation $\prod_{i=1}^{i=m} \underline{\mathcal{Z}_i}^{\mathfrak{z}_i} = \mathcal{B}$, since one can lift any ciphertext from $\mathbb{G}$ to a ciphertext in $\mathbb{G}_T$, setting $\mathcal{Z}_i = e(\mathcal{Y}_i, \mathcal{A}_i)$, as well as, for $j = 1, 2, 3$, $G_{i,j} = e(g_j, \mathcal{A}_i)$ and for $j = 1, 2$, $H_{i,j} = e(h_j, \mathcal{A}_i)$, $C_{i,j} = e(c_j, \mathcal{A}_i)$, $D_{i,j} = e(d_j, \mathcal{A}_i)$, to lift all the group basis elements. Then, one transforms $\mathcal{C}_i = \mathsf{LCS}^*(\ell, \mathsf{ek}, \mathcal{Y}_i, \xi; \mathbf{z}_i) = (\boldsymbol{u_i} = (g_1^{z_{r_i}}, g_2^{z_{s_i}}, g_3^{z_{r_i}+z_{s_i}}), e_i = h_1^{z_{r_i}} h_2^{z_{s_i}} \cdot \mathcal{Y}_i, v_i = (c_1 d_1^{\xi})^{z_{r_i}} \cdot (c_2 d_2^{\xi})^{z_{s_i}})$ into $(\boldsymbol{U_i} = (G_{i,1}^{z_{r_i}}, G_{i,2}^{z_{s_i}}, G_{i,3}^{z_{r_i}+z_{s_i}}), E_i = H_{i,1}^{z_{r_i}} H_{i,2}^{z_{s_i}} \cdot \mathcal{Z}_i, V_i = (C_{i,1} D_{i,1}^{\xi})^{z_{r_i}} \cdot (C_{i,2} D_{i,2}^{\xi})^{z_{s_i}})$. Encryptions of $\mathcal{Z}_i$ originally in $\mathbb{G}_T$ use constant basis elements for $j = 1, 2, 3$, $G_{i,j} = G_j = e(g_j, g)$ and for $j = 1, 2$, $H_{i,j} = H_j = e(h_j, g)$, $C_{i,j} = C_j = e(c_j, g)$, $D_{i,j} = D_j = e(d_j, g)$.

The commitments have been generated in $\mathbb{G}$ and $\mathbb{G}_T$ simultaneously using the m-$\mathsf{DLCSCom}'$ version, with a common ξ, where the possible combination with the companion ciphertext to the power ε leads to the above $\mathcal{C}_i$,

thereafter lifted to $\mathbb{G}_T$. For the hashing keys, one picks random scalars $(\lambda, (\eta_i, \theta_i, \kappa_i, \mu_i)_{i=1,\ldots,m}) \xleftarrow{\$} \mathbb{Z}_p^{4m+1}$, and sets $\mathsf{hk}_i = (\eta_i, \theta_i, \kappa_i, \lambda, \mu_i)$. One then computes the projection keys as $\mathsf{hp}_i = (g_1^{\eta_i} g_3^{\kappa_i} h_1^{\lambda} (c_1 d_1^{\xi})^{\mu_i}, g_2^{\theta_i} g_3^{\kappa_i} h_2^{\lambda} (c_2 d_2^{\xi})^{\mu_i}) \in \mathbb{G}^2$. The hash value is

$$\prod_i e(u_{i,1}^{\eta_i} \cdot u_{i,2}^{\theta_i} \cdot u_{i,3}^{\kappa_i} \cdot e_i^{\lambda} \cdot v_i^{\mu_i}, \mathcal{A}_i) \times \mathcal{B}^{-\lambda} = \prod_i e(\mathsf{hp}_{i,1}^{z_{r_i}} \mathsf{hp}_{i,2}^{z_{s_i}}, \mathcal{A}_i),$$

where $\mathcal{A}_i$ is the constant used to compute $\mathcal{Z}_i = e(\mathcal{Y}_i, \mathcal{A}_i)$ and to lift ciphertexts from $\mathbb{G}$ to $\mathbb{G}_T$, or $\mathcal{A}_i = g^{\mathfrak{d}_i}$ if the ciphertext was already in $\mathbb{G}_T$. These evaluations can be computed either from the commitments and the hashing keys, or from the projection keys and the witnesses. We insist on the fact that, whereas the hash values are in $\mathbb{G}_T$, the projection keys are in $\mathbb{G}$ even if the ciphertexts are initially in $\mathbb{G}_T$. We stress again that the projection keys require the knowledge of ξ only: known from the LCSCom commitment or the first part $\mathcal{C}$ of the $\mathsf{DLCSCom}'$ commitment.

5 Language-Authenticated Key Exchange

5.1 The Ideal Functionality

We generalize the Password-Authenticated Key Exchange functionality $\mathcal{F}_{\text{PAKE}}$ (first provided in [CHK+05]) to more complex languages: the players agree on a common secret key if and only if they own words that lie in the languages the partners have in mind. More precisely, after an agreement on pub between P_i and P_j (modeled here by the use of the split functionality, see below), player P_i uses a word W_i belonging to $L_i = L_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and it expects its partner P_j to use a word W_j belonging to the language $L'_j = L_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}'_j)$, and vice-versa for P_j and P_i. We assume relations $\mathcal{R}_i$ and $\mathcal{R}_j$ to be specified by the kind of protocol we study (PAKE, Verifier-based PAKE, secret handshakes, ...) and so the languages are defined by the additional parameters pub, priv_i and priv_j only: they both agree on the public part pub, to be possibly parsed in a different way by each player for each language according to the relations. Note however that the respective languages do not need to be the same or to use similar relations: authentication means could be totally different for the 2 players. The key exchange should succeed if and only if the two following pairs of equations hold: ($L'_i = L_i$ and $W_i \in L_i$) and ($L'_j = L_j$ and $W_j \in L_j$).

Description. In the initial $\mathcal{F}_{\text{PAKE}}$ functionality [CHK+05], the adversary was given access to a $\mathsf{TestPwd}$-query, which modeled the on-line dictionary attack. But it is known since [BCL+05] that it is equivalent to use the split functionality model [BCL+05], generate the $\mathsf{NewSession}$-queries corresponding to the corrupted players and tell the adversary (on behalf of the corrupted player) whether the protocol should succeed or not. Both methods enable the adversary to try a credential for a player (on-line dictionary attack). The second method (that we use here) implies allowing $\mathcal{S}$ to ask $\mathsf{NewSession}$-queries on behalf of the corrupted player, and letting it to be aware of the success or failure of the protocol in this case: the adversary learns this information only when it plays on behalf of a player (corruption or impersonation attempt). This is any way an information it would learn at the end of the protocol. We insist that third parties will not learn whether the protocol succeeded or not, as required for secret handshakes. To this aim, the NewKey-query informs in this case the adversary whether the credentials are consistent with the languages or not. In addition, the split functionality model guarantees from the beginning which player is honest and which one is controlled by the adversary. This finally allows us to get rid of the $\mathsf{TestPwd}$-query. The $\mathcal{F}_{\text{LAKE}}$ functionality is presented in Figure 2 and the corresponding split functionality $s\mathcal{F}_{\text{LAKE}}$ in Figure 3, where the languages are formally described and compared using the pub and priv parts.

The security goal is to show that the best attack for the adversary is a basic trial execution with a credential of its guess or choice: the proof will thus consist in emulating any real-life attack by either a trial execution by the adversary, playing as an honest player would do, but with a credential chosen by the adversary or obtained in any way; or a denial of service, where the adversary is clearly aware that its behavior will make the execution fail.

5.2 A Generic UC-Secure LAKE Construction

Intuition. Using smooth projective hash functions on commitments, one can generically define a LAKE protocol as done in [ACP09]. The basic idea is to make the player commit to their private information (for the expected

The functionality $\mathcal{F}_{\text{LAKE}}$ is parametrized by a security parameter k and a public parameter pub for the languages. It interacts with an adversary $\mathcal{S}$ and a set of parties $P_1,\ldots,P_n$ via the following queries:

- New Session: Upon receiving a query $(\mathsf{NewSession} : \mathsf{sid}, P_i, P_j, W_i, L_i = L(\mathsf{pub}, \mathsf{priv}_i), L'_j = L(\mathsf{pub}, \mathsf{priv}'_j))$ from P_i,
  - If this is the first $\mathsf{NewSession}$-query with identifier sid, record the tuple $(P_i, P_j, W_i, L_i, L'_j, \mathsf{initiator})$. Send $(\mathsf{NewSession}; \mathsf{sid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ to $\mathcal{S}$ and P_j.
  - If this is the second $\mathsf{NewSession}$-query with identifier sid and there is a record $(P_j, P_i, W_j, L_j, L'_i, \mathsf{initiator})$, record the tuple $(P_j, P_i, W_j, L_j, L'_i, \mathsf{initiator}, W_i, L_i, L'_j, \mathsf{receiver})$. Send $(\mathsf{NewSession}; \mathsf{sid}, P_i, P_j, \mathsf{pub}, \mathsf{receiver})$ to $\mathcal{S}$ and P_j.
- Key Computation: Upon receiving a query $(\mathsf{NewKey} : \mathsf{sid})$ from $\mathcal{S}$, if there is a record of the form $(P_i, P_j, W_i, L_i, L'_j, \mathsf{initiator}, W_j, L_j, L'_i, \mathsf{receiver})$ and this is the first NewKey-query for session sid, then
  - If $(L'_i = L_i$ and $W_i \in L_i)$ and $(L'_j = L_j$ and $W_j \in L_j)$, then pick a random key sk of length k and store $(\mathsf{sid}, \mathsf{sk})$. If one player is corrupted, send $(\mathsf{sid}, \mathsf{success})$ to the adversary.
  - Else, store $(\mathsf{sid}, \perp)$, and send $(\mathsf{sid}, \mathsf{fail})$ to the adversary if one player is corrupted.
- Key Delivery: Upon receiving a query $(\mathsf{SendKey} : \mathsf{sid}, P_i, \mathsf{sk})$ from $\mathcal{S}$, then
  - if there is a record of the form $(\mathsf{sid}, \mathsf{sk}')$, then, if both players are uncorrupted, output $(\mathsf{sid}, \mathsf{sk}')$ to P_i. Otherwise, output $(\mathsf{sid}, \mathsf{sk})$ to P_i.
  - if there is a record of the form $(\mathsf{sid}, \perp)$, then pick a random key sk' of length k and output $(\mathsf{sid}, \mathsf{sk}')$ to P_i.

Figure 2. Ideal Functionality $\mathcal{F}_{\text{LAKE}}$

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

- Initialization:
  - Upon receiving $(\mathsf{Init}, \mathsf{sid}, \mathsf{pub}_i)$ from party P_i, send $(\mathsf{Init}, \mathsf{sid}, P_i, \mathsf{pub}_i)$ to the adversary.
  - Upon receiving a message $(\mathsf{Init}, \mathsf{sid}, P_i, H, \mathsf{pub}, \mathsf{sid}_H)$ from $\mathcal{S}$, where $H = \{P_i, P_j\}$ is a set of party identities, check that P_i has already sent $(\mathsf{Init}, \mathsf{sid}, \mathsf{pub}_i)$ and that for all recorded $(H', \mathsf{pub}', \mathsf{sid}_{H'})$, either $H = H'$, $\mathsf{pub} = \mathsf{pub}'$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$ or H and H' are disjoint and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$. If so, record the pair $(H, \mathsf{pub}, \mathsf{sid}_H)$, send $(\mathsf{Init}, \mathsf{sid}, \mathsf{sid}_H, \mathsf{pub})$ to P_i, and invoke a new functionality $(\mathcal{F}_{\text{LAKE}}, \mathsf{sid}_H, \mathsf{pub})$ denoted as $\mathcal{F}_{\text{LAKE}}^{(H,\mathsf{pub})}$ and with set of honest parties H.
- Computation:
  - Upon receiving $(\mathsf{Input}, \mathsf{sid}, m)$ from party P_i, find the set H such that $P_i \in H$, the public value pub recorded, and forward m to $\mathcal{F}_{\text{LAKE}}^{(H,\mathsf{pub})}$.
  - Upon receiving $(\mathsf{Input}, \mathsf{sid}, P_j, H, m)$ from $\mathcal{S}$, such that $P_j \notin H$, forward m to $\mathcal{F}_{\text{LAKE}}^{(H,\mathsf{pub})}$ as if coming from P_j.
  - When $\mathcal{F}_{\text{LAKE}}^{(H,\mathsf{pub})}$ 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.

Figure 3. Split Functionality $s\mathcal{F}_{\text{LAKE}}$

languages and the owned words), and eventually the smooth projective hash functions will be used to make implicit validity checks of the global relation.

To this aim, we use the commitments and associated smooth projective hash functions as described in Sections 3 and 4. More precisely, all examples of SPHF in Section 4 can be used on extractable commitments divided into one or two parts (the non-equivocable LCSCom or the equivocable $\mathsf{DLCSCom}'$ commitments, see Figure 1). The relations on the committed values will not be explicitly checked, since the values will never be revealed, but will be implicitly checked using SPHF. It is interesting to note that in both cases (one-part or two-part commitment), the projection key will only depend on the first part of the commitment.

As it is often the case in the UC setting, we need the initiator to use stronger primitives than the receiver. They both have to use non-malleable and extractable commitments, but the initiator will use a commitment that is additionally equivocable, the $\mathsf{DLCSCom}'$ in two parts ($(\mathcal{C}_i, \mathcal{C}'_i)$ and $\mathsf{Com}_i = \mathcal{C}_i \cdot \mathcal{C}'^{\boldsymbol{\epsilon}}_i$), while the receiver will only need the basic LCSCom commitment in one part ($\mathsf{Com}_j = \mathcal{C}_j$).

As already explained, SPHF will be used to implicitly check whether $(L'_i = L_i$ and $W_i \in L_i)$ and $(L'_j = L_j$ and $W_j \in L_j)$. But since in our instantiations private parameters priv and words W will have to be committed, the structure of these commitments will thus be publicly known in advance: commitments of $\mathcal{P}$-elements and $\mathcal{S}$-elements. Section 6 discusses on the languages captured by our definition, and illustrates with some AKE protocols. However, while these $\mathcal{P}$ and $\mathcal{S}$ sets are embedded in $\mathbb{G}^n$ from some n, it might be important to prove that the committed values are actually in $\mathcal{P}$ and $\mathcal{S}$ (e.g., one can have to prove it commits bits, whereas messages are first embedded as group elements in $\mathbb{G}$ of large order p). This will be an additional language-membership to prove on the commitments.

This leads to a very simple protocol described on Figure 4. Note that if a player wants to make external adversaries think he owns an appropriate word, as it is required for Secret Handshakes, he can still play, but

Execution between P_i and P_j, with session identifier sid.

– Preliminary Round: each user generates a pair of signing/verification keys $(\mathsf{SK}, \mathsf{VK})$ and sends VK together with its contribution to the public part of the language.

We denote by $\ell_i = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_i, \mathsf{VK}_j)$ and by $\ell_j = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_j, \mathsf{VK}_i)$, where pub is the combination of the contributions of the two players. The initiator now uses a word W_i in the language $L(\mathsf{pub}, \mathsf{priv}_i)$, and the receiver uses a word W_j in the language $L(\mathsf{pub}, \mathsf{priv}_j)$, possibly re-randomized from their long-term secrets (*). We assume commitments and associated smooth projective hash functions exist for these languages.

– First Round: user P_i (with random tape ω_i) generates a multi-$\mathsf{DLCSCom}'$ commitment on $(\mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ in $(\mathcal{C}_i, \mathcal{C}'_i)$, where W_i has been randomized in the language, under the label ℓ_i. It also computes a Pedersen commitment on $\mathcal{C}'_i$ in $\mathcal{C}''_i$ (with random exponent t). It then sends $(\mathcal{C}_i, \mathcal{C}''_i)$ to P_j;
– Second Round: user P_j (with random tape ω_j) computes a multi-LCS commitment on $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$ in $\mathsf{Com}_j = \mathcal{C}_j$, with witness $\boldsymbol{r}$, where W_j has been randomized in the language, under the label ℓ_j. It then generates a challenge $\boldsymbol{\varepsilon}$ on $\mathcal{C}_i$ and hashing/projection keys (**) hk_i and hp_i associated to $\mathcal{C}_i$ (which will be associated to the future Com_i). It finally signs all the flows using SK_j in σ_j, and sends $(\mathcal{C}_j, \boldsymbol{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to P_i;
– Third Round: user P_i first checks the signature σ_j, computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\boldsymbol{\varepsilon}}$ and witness $\mathbf{z}$ (from $\boldsymbol{\varepsilon}$ and ω_i), it generates hashing/projection keys hk_j and hp_j associated to Com_j. It finally signs all the flows using SK_i in σ_i, and sends $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to P_j;
– Hashing: P_j first checks the signature σ_i and the correct opening of $\mathcal{C}''_i$ into $\mathcal{C}'_i$, it computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\boldsymbol{\varepsilon}}$.
P_i computes K_i and P_j computes K_j as follows:

$$K_i = \mathsf{Hash}(\mathsf{hk}_j, \{(\mathsf{priv}'_j, \mathsf{priv}_i)\} \times L(\mathsf{pub}, \mathsf{priv}'_j), \ell_j, \mathsf{Com}_j) \cdot \mathsf{ProjHash}(\mathsf{hp}_i, \{(\mathsf{priv}_i, \mathsf{priv}'_j)\} \times L(\mathsf{pub}, \mathsf{priv}_i), \ell_i, \mathsf{Com}_i; \mathbf{z})$$
$$K_j = \mathsf{ProjHash}(\mathsf{hp}_j, \{(\mathsf{priv}_j, \mathsf{priv}'_i)\} \times L(\mathsf{pub}, \mathsf{priv}_j), \ell_j, \mathsf{Com}_j; \boldsymbol{r}) \cdot \mathsf{Hash}(\mathsf{hk}_i, \{(\mathsf{priv}'_i, \mathsf{priv}_j)\} \times L(\mathsf{pub}, \mathsf{priv}'_i), \ell_i, \mathsf{Com}_i)$$

(*) As explained in Section 1, recall that the languages considered depend on two possibly different relations, namely $L_i = L_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and $L_j = L_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}_j)$, but we omit them for the sake of clarity. We assume they are both self-randomizable.
(**) Recall that the SPHF is constructed in such a way that this projection key does not depend on $\mathcal{C}'_i$ and is indeed associated to the future whole Com_i.

Figure 4. Language-based Authenticated Key Exchange from a Smooth Projective Hash Function on Commitments

will compute everything with dummy words, and will replace the $\mathsf{ProjHash}$ evaluation by a random value, which will lead to a random key at the end.

Security Analysis. Since we have to assume common pub, we make a first round (with flows in each direction) where the players send their contribution, to come up with pub. These flows will also be used to know if there is a player controlled by the adversary (as with the Split Functionality [BCL+05]). In case the languages have empty pub, these additional flows are not required, since the Split Functionality can be applied on the committed values. The signing key for the receiver is not required anymore since there is one flow only from its side. This LAKE protocol is secure against static corruptions. The proof is provided in the Appendix D, and is in the same vein as the one in [CHK+05, ACP09]. However, it is a bit more intricate:

– in PAKE, when one is simulating a player, and knows the adversary used the correct password, one simply uses this password for the simulated player. In LAKE, when one knows the language expected by the adversary for the simulated player and has to simulate a successful execution (because of success announced by the NewKey-query), one has to actually include a correct word in the commitment: smooth projective hash functions do not allow the simulator to cheat, equivocability of the commitment is the unique trapdoor, but with a valid word. The languages must allow the simulator to produce a valid word W in $L(\mathsf{pub}, \mathsf{priv})$, for any pub and $\mathsf{priv} \in \mathcal{P}$ provided by the adversary or the environment. This will be the case in all the interesting applications of our protocol (see Section 6): if priv defines a Waters' verification key $\mathsf{vk} = g^x$, with the master key s such that $h = g^s$, the signing key is $\mathsf{sk} = h^x = \mathsf{vk}^s$, and thus the simulator can sign any message; if such a master key does not exist, one can restrict $\mathcal{P}$, and implicitly check it with the SPHF (the additional language-membership check, as said above). But since a random word is generated by the simulator, we need the real player to derive a random word from his own word, and the language to be *self-randomizable*.
– In addition, as already noted, our commitment $\mathsf{DLCSCom}'$ is not formally binding (contrarily to the much less efficient one used in [ACP09]). The adversary can indeed make the extraction give $\boldsymbol{M}$ from $\mathcal{C}_i$, whereas

Com_i will eventually contain $\boldsymbol{M}'$ if $\mathcal{C}'_i$ does not encrypt $(1_{\mathbb{G}})^n$. However, since the actual value $\boldsymbol{M}'$ depends on the random challenge $\boldsymbol{\varepsilon}$, and the language is assumed sparse (otherwise authentication is easy), the protocol will fail: this can be seen as a denial of service from the adversary.

Theorem 1. *Our LAKE scheme from Figure 4 realizes the $s\mathcal{F}_{\text{LAKE}}$ functionality in the $\mathcal{F}_{\text{CRS}}$-hybrid model, in the presence of static adversaries, under the* DLin *assumption and the security of the One-Time Signature.*

Actually, from a closer look at the full proof, one can notice that $\mathsf{Com}_j = \mathcal{C}_j$ needs to be extractable, but $\mathsf{IND}-\mathsf{CPA}$ security is enough, which leads to a shorter ciphertext (2 group elements less if one uses a Linear ciphertext instead of LCS). Similarly, one will not have to extract W_i from $\mathcal{C}_i$ when simulating sessions where P_i is corrupted. As a consequence, only the private parts of the languages have to be committed to in Com_i in the first and third rounds, whereas W_i can be encrypted independently with an $\mathsf{IND}-\mathsf{CPA}$ encryption scheme in the third round only (5 group elements less in the first round, and 2 group elements less in the third round if one uses a Linear ciphertext instead of LCS).

6 Concrete Instantiations and Comparisons

In this section, we first give some concrete instantiations of several AKE protocols, using our generic protocol of LAKE, and compare the efficiencies of those instantiations.

6.1 Possible Languages

As explained above, our LAKE protocol is provably secure for *self-randomizable* languages only. While this notion may seem quite strong, most of the usual languages fall into it. For example, in a PAKE or a Verifier-based PAKE scheme, the languages consist of a single word and so trivially given a word, each user is able to deduce all the words in the language. One may be a little more worried about Waters Signature in our Secret Handshake, and/or Linear pairing equations. However the *self-randomizability* of the languages is easy to show:

- Given a Waters signature $\sigma = (\sigma_1, \sigma_2)$ over a message m valid under a verification key vk, one is able to randomize the signature into any signature over the same message m valid under the same verification key vk simply by picking a random s and computing $\sigma' = (\sigma_1 \cdot \mathcal{F}(m)^s, \sigma_2 \cdot g^s)$.
- For linear pairing equations, with public parameters $\mathcal{A}_i$ for $i = 1, \ldots, m$ and γ_i for $i = m+1, \ldots, n$, and $\mathcal{B}$, given $(\mathcal{X}_1, \ldots, \mathcal{X}_m, \mathcal{Z}_{m+1}, \ldots, \mathcal{Z}_n)$ verifying $\prod_{i=1}^{m} e(\mathcal{X}_i, \mathcal{A}_i) \cdot \prod_{i=m+1}^{n} \mathcal{Z}_i^{\gamma_i} = \mathcal{B}$, one can randomize the word in the following way:
  - If $m < n$, one simply picks random $(\mathcal{X}'_1, \ldots, \mathcal{X}'_m)$, $(\mathcal{Z}'_{m+1}, \ldots, \mathcal{Z}'_{n-1})$ and sets $\mathcal{Z}'_n = (\mathcal{B}/(\prod_{i=1}^{m} e(\mathcal{X}'_i, \mathcal{A}_i) \cdot \prod_{i=m+1}^{n-1} \mathcal{Z}_i'^{\gamma_i}))^{1/\gamma_n}$,
  - Else, if $m = n > 1$, one picks random $r_1, \ldots, r_{n-1}$ and set $\mathcal{X}'_i = \mathcal{X}_i \cdot \mathcal{A}_n^{r_i}$, for $i = 1, \ldots, m-1$ and $\mathcal{X}'_m = \mathcal{X}_m \cdot \prod_{i=1}^{m-1} \mathcal{A}_i^{-r_i}$,
  - Else $m = n = 1$, this means only one word satisfies the equation. So we already have this word.

As we can see most of the common languages manageable with a SPHF are already *self-randomizable*. We now show how to use them in concrete instantiations.

6.2 Concrete Instantiations

Password-Authenticated Key Exchange. Using our generic construction, we can easily obtain a PAKE protocol, as described on Figure 5, where we optimize from the generic construction, since $\mathsf{pub} = \emptyset$, removing the agreement on pub, but still keeping the one-time signature keys $(\mathsf{SK}_i, \mathsf{VK}_i)$ to avoid man-in-the-middle attacks since it has another later flow: P_i uses a password W_i and expects P_j to own the same word, and thus in the language $L'_j = L_i = \{W_i\}$; P_j uses a password W_j and expects P_i to own the same word, and thus in the language $L'_i = L_j = \{W_j\}$; The relation is the equality test between priv_i and priv_j, which both have no restriction in $\mathbb{G}$ (hence $\mathcal{P} = \mathbb{G}$). As the word W_i, the language private parameters priv_i of a user and priv'_j of the expected language for the other user are the same, each user can commit in the protocol to only one value: its password.

We kept the general description and notations in Figure 5, but $\mathcal{C}_j$ can be a simply $\mathsf{IND}-\mathsf{CPA}$ encryption scheme. It is quite efficient and relies on the DLin assumption, with DLCS for $(\mathcal{C}_i, \mathcal{C}'_i)$ and thus 10 group elements,

P_i uses a password W_i and P_j uses a password W_j. We denote $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$.

- First Round: P_i (with random tape ω_i) first generates a pair of signing/verification keys $(\mathsf{SK}_i, \mathsf{VK}_i)$ and a $\mathsf{DLCSCom}'$ commitment on W_i in $(\mathcal{C}_i, \mathcal{C}'_i)$, under $\ell_i = (\ell, \mathsf{VK}_i)$. It also computes a Pedersen commitment on $\mathcal{C}'_i$ in $\mathcal{C}''_i$ (with random exponent t). It then sends $(\mathsf{VK}_i, \mathcal{C}_i, \mathcal{C}''_i)$ to P_j;
- Second Round: P_j (with random tape ω_j) computes a LCSCom commitment on W_j in $\mathsf{Com}_j = \mathcal{C}_j$, with witness $\boldsymbol{r}$, under the label ℓ. It then generates a challenge ε on $\mathcal{C}_i$ and hashing/projection keys hk_i and the corresponding hp_i for the equality test on Com_i ("Com_i is a valid commitment of W_j", this only requires the value ξ_i computable thanks to $\mathcal{C}_i$). It then sends $(\mathcal{C}_j, \varepsilon, \mathsf{hp}_i)$ to P_i;
- Third Round: user P_i can compute $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}'^{\varepsilon}_i$ and witness $\mathbf{z}$ (from ε and ω_i), it generates hashing/projection keys hk_j and hp_j for the equality test on Com_j. It finally signs all the flows using SK_i in σ_i and send $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to P_j;
- Hashing: P_j first checks the signature and the validity of the Pedersen commitment (thanks to t), it computes $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}'^{\varepsilon}_i$. P_i computes K_i and P_j computes K_j as follows:

$$K_i = \mathsf{Hash}(\mathsf{hk}_j, L'_j, \ell, \mathsf{Com}_j) \cdot \mathsf{ProjHash}(\mathsf{hp}_i, L_i, \ell_i, \mathsf{Com}_i; \mathbf{z})$$
$$K_j = \mathsf{ProjHash}(\mathsf{hp}_j, L_j, \ell, \mathsf{Com}_j; \boldsymbol{r}) \cdot \mathsf{Hash}(\mathsf{hk}_i, L'_i, \ell_i, \mathsf{Com}_i)$$

Figure 5. Password-based Authenticated Key Exchange

but a Linear encryption for $\mathcal{C}_j$ and thus 3 group elements. Projection keys are both 2 group elements. Globally, P_i sends 13 groups elements plus 1 scalar, a verification key and a one-time signature, while P_j sends 5 group elements and 1 scalar: 18 group elements and 2 scalars in total. We can of course instantiate it with the Cramer-Shoup and ElGamal variants, under the DDH assumption: P_i sends 8 groups elements plus 1 scalar, a verification key and a one-time signature, while P_j sends 3 group elements and 1 scalar (all group elements can be in the smallest group): 11 group elements and 2 scalars in total.

Verifier-based PAKE. The above scheme can be modified into an efficient PAKE protocol that is additionally secure against *server compromise*: the so-called verifier-based PAKE, where the client owns a password pw, while the server knows a verifier only, such as g^{pw}, so that in case of break-in to the server, the adversary will not immediately get all the passwords.

To this aim, as usually done, one first does a PAKE with g^{pw} as common password, then asks the client to additionally prove it can compute the Diffie-Hellman value h^{pw} for a basis h chosen by the server. Ideally, we could implement this trick, where the client P_j just considers the equality test between the g^{pw} and the value committed by the server for the language $L'_i = L_j$, while the server P_i considers the equality test with $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, where h is sent as its contribution to the public part of the language by the server $L_i = L'_j$. Since the server chooses h itself, it chooses it as $h = g^{\alpha}$, for an ephemeral random α, and can thus compute $h^{\mathsf{pw}} = (g^{\mathsf{pw}})^{\alpha}$. On its side, the client can compute this value since it knows pw. The client could thus commit to $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, in order to prove its knowledge of pw, whereas the server could just commit to g^{pw}. Unfortunately, from the extractability of the server commitment, one would just get g^{pw}, which is not enough to simulate the client.

To make it in a provable way, the server chooses an ephemeral h as above, and they both run the previous PAKE protocol with $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$ as common password, and mutually checked: h is seen as the pub part, hence the preliminary flows are required.

Credential-Authenticated Key Exchange. In [CCGS10], the authors proposed instantiations of the CAKE primitive for conjunctions of atomic policies that are defined algebraically by relations of the form $\prod_{j=1}^{k} g_j^{F_j} = 1$ where the g_j's are elements of an abelian group and F_j's are integer polynomials in the variables committed by the users.

The core of their constructions relies on their practical UC zero-knowledge proof. There is no precise instantiation of such proof, but it is very likely to be inefficient. Their proof technique indeed requires to transform the underlying Σ-protocols into corresponding Ω-protocols [GMY06] by verifiably encrypting the witness. An Ω-protocol is a Σ-protocol with the additional property that it admits a polynomial-time straight-line extractor. Since the witnesses are scalars in their algebraic relations, their approach requires either inefficient bit-per-bit encryption of these witnesses or Paillier encryption in which case the problem of using group with different orders in the representation and in the encryption requires additional overhead.

Even when used with Σ-protocols, their PAKE scheme without UC-security, requires at least two proofs of knowledge of representations that involve at least 30 group elements (if we assume the encryption to be linear

Cramer Shoup), and some extra for the last proof of existence (*cf.* [CKS11]), where our PAKE requires less than 20 group elements. Anyway they say, their PAKE scheme is less efficient than [CHK$^+$05], which needed 6 rounds and around 30 modular exponentiations per user, while our efficient PAKE requires less than 40 exponentiations, in total, in only 3 rounds. Our scheme is therefore more efficient than the scheme from [CHK$^+$05] for the same security level (*i.e.* UC-security with static corruptions).

Secret-Handshakes. We can also instantiate a (linkable) Secret Handshakes protocol, using our scheme with two different languages: P_i will commit to a valid signature σ_i on a message m_i (his identity for example), under a private verification key vk_i, and expects P_j to commit to a valid signature on a message m'_j under a private verification key vk'_j; but P_j will do analogously with a signature σ_j on m_j under vk_j, while expecting a signature on m'_i under vk'_i. The public parts of the signature (the second component) are sent in clear with the commitments.

In a regular Secret Handshakes both users should use the same languages. But here, we have a more general situation (called *dynamic matching* in [AKB07]): the two participants will have the same final value if and only if they both belong to the organization the other expects. If one lies, our protocol guarantees no information leakage. Furthermore, the semantic security of the session is even guaranteed with respect to the authorities, in a forward-secure way (this property is also achieved in [JL09] but in a weaker security model). Finally, our scheme supports revocation and can handle roles as in [AKB07].

Standard secret handshakes, like [AKB07], usually work with credentials delivered by a unique authority, this would remove our need for a hidden verification key, and private part of the language. Both users would only need to commit to signatures on their identity/credential, and show that they are valid. This would require a dozen of group elements with our approach. Their construction requires only 4 elements under BDH, however it relies on the asymmetric Waters IBE with only two elements, whereas the only security proof known for such IBE [Duc10] requires an extra term in $\mathbb{G}_2$ which would render their technique far less efficient, as several extra terms would be needed to expect a provably secure scheme. While sometimes less effective, our LAKE approach can manage Secret Handshakes, and provide additional functionalities, like more granular control on the credential as part of them can be expressly hidden by both the users. More precisely, we provide affiliation-hiding property and let third parties unaware of the success/failure of the protocol.

Unlinkable Secret-Handshakes. Moving the users' identity from the public pub part to individual private priv part, and combining our technique with [BPV12], it is also possible to design an *unlinkable* Secret Handshakes protocol [JL09] with practical efficiency. It illustrates the case where committed values have to be proven in a strict subset of $\mathbb{G}$, as one has to commit to bits: the signed message M is now committed and not in clear, it thus has to be done bit-by-bit since the encoding $\mathcal{G}$ does not allow algebraic operations with the content to apply the Waters function on the message. It is thus possible to prove the knowledge of a Waters signature on a private message (identity) valid under a private verification key. Additional relations can be required on the latter to make authentication even stronger.

Acknowledgments

This work was supported in part by the European Commission through the FP7-ICT-2011-EU-Brazil Program under Contract 288349 SecFuNet and the ICT Program under Contract ICT-2007-216676 ECRYPT II.

References

[ACGP11] Michel Abdalla, Céline Chevalier, Louis Granboulan, and David Pointcheval. Contributory password-authenticated group key exchange with join capability. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 142–160. Springer, February 2011.

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 671–689. Springer, August 2009.

[AKB07] Giuseppe Ateniese, Jonathan Kirsch, and Marina Blanton. Secret handshakes with dynamic and fuzzy matching. In *ISOC Network and Distributed System Security Symposium – NDSS 2007*. The Internet Society, February / March 2007.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, August 2004.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 361–377. Springer, August 2005.

[BDS+03] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, pages 180–196. IEEE Computer Society, 2003.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 403–422. Springer, March 2011.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

[BPV12] Olivier Blazy, David Pointcheval, and Damien Vergnaud. Round-optimal privacy-preserving protocols with smooth projective hash functions. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *Lecture Notes in Computer Science*, pages 94–111, Springer, March 2012.

[Can01] Ran 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, October 2001.

[CCGS10] Jan Camenisch, Nathalie Casati, Thomas Groß, and Victor Shoup. Credential authenticated identification and key exchange. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 255–276. Springer, August 2010.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421. Springer, May 2005.

[CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 613–630. Springer, August 2007.

[CKS11] Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In *Advances in Cryptology – ASIACRYPT 2011*, Lecture Notes in Computer Science, pages 449–467. Springer, December 2011.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, August 2003.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, August 1998.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64. Springer, April / May 2002.

[Duc10] Léo Ducas. Anonymity from asymmetry: New constructions for anonymous HIBE. In Josef Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 148–164. Springer, March 2010.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 524–543. Springer, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMY06] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, April 2008.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Private mutual authentication and conditional oblivious transfer. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 90–107. Springer, August 2009.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 446–466. Springer, May 2011.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, August 1992.

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/2007/074.pdf`.

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, May 2005.

A Preliminaries

A.1 Formal Definitions of the Primitives

We first recall the definitions of the basic tools, with the security notions with success/advantage that all depend on a security parameter (which is omitted here for simplicity of notation).

Hash Function Family. A hash function family $\mathcal{H}$ is a family of functions $\mathfrak{H}_K$ from $\{0,1\}^*$ to a fixed-length output, either $\{0,1\}^k$ or $\mathbb{Z}_p$. Such a family is said *collision-resistant* if for any adversary $\mathcal{A}$ on a random function $\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}$, it is hard to find a collision. More precisely, we denote

$$\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A}) = \Pr[\mathfrak{H}_K \xleftarrow{\$} \mathcal{H}, (m_0, m_1) \leftarrow \mathcal{A}(\mathfrak{H}_K) : \mathfrak{H}_K(m_0) = \mathfrak{H}_K(m_1)], \qquad \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) = \max_{\mathcal{A} \le t}\{\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(\mathcal{A})\}.$$

Labeled encryption scheme. A labeled public-key encryption scheme is defined by four algorithms:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ generates a pair of keys, the encryption key ek and the decryption key dk;
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, m; r)$ produces a ciphertext c on the input message $m \in \mathcal{M}$ under the label ℓ and encryption key ek, using the random coins r;
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, c)$ outputs the plaintext m encrypted in c under the label ℓ, or $\perp$.

An encryption scheme $\mathcal{E}$ should satisfy the following properties

- *Correctness*: for all key pair $(\mathsf{ek}, \mathsf{dk})$, any label ℓ, all random coins r and all messages m,

$$\mathsf{Decrypt}(\ell, \mathsf{dk}, \mathsf{Encrypt}(\ell, \mathsf{ek}, m; r)) = m.$$

- *Indistinguishability under chosen-ciphertext attacks*: this security notion can be formalized by the following security game, where the adversary $\mathcal{A}$ keeps some internal state between the various calls `FIND` and `GUESS`, and makes use of the oracle $\mathsf{ODecrypt}$:
  - $\mathsf{ODecrypt}(\ell, c)$: This oracle outputs the decryption of c under the label ℓ and the challenge decryption key dk. The input queries (ℓ, c) are added to the list $\mathcal{CT}$.

  The advantages are

$$\mathsf{Exp}^{\mathsf{ind-cca}-b}_{\mathcal{E},\mathcal{A}}(k)$$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$
2. $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(\ell^*, m_0, m_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{ek}, \mathsf{ODecrypt}(\cdot,\cdot))$
4. $c^* \leftarrow \mathsf{Encrypt}(\ell, \mathsf{ek}, m_b)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*, \mathsf{ODecrypt}(\cdot,\cdot))$
6. `IF` $(\ell^*, c^*) \in \mathcal{CT}$ `RETURN` 0
7. `ELSE RETURN` b'

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\mathcal{E}}(\mathcal{A}) = \Pr[\mathsf{Exp}^{\mathsf{ind-cca}-1}_{\mathcal{E},\mathcal{A}}(k) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind-cca}-0}_{\mathcal{E},\mathcal{A}}(k) = 1] \quad \mathsf{Adv}^{\mathsf{ind-cca}}_{\mathcal{E}}(t) = \max_{\mathcal{A} \le t}\{\mathsf{Adv}^{\mathsf{ind-cca}}_{\mathcal{E}}(\mathcal{A})\}.$$

Labeled commitment scheme. A labeled commitment scheme is defined by three algorithms:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{Commit}(\ell, m; r)$ produces a commitment c and the opening information d on the input message $m \in \mathcal{M}$ under the label ℓ, using the random coins r;
- $\mathsf{Decommit}(\ell, c, m, d)$ checks the validity of the opening information d on the commitment c for the message m under the label ℓ. It answers 1 for true, and 0 for false.

A commitment scheme $\mathcal{C}$ should satisfy the following properties

- *Correctness*: for any label ℓ, and all messages m, if $(c, d) \leftarrow \mathsf{Commit}(\ell, m; r)$, then $\mathsf{Decommit}(\ell, c, m, d) = 1$.
- *Hiding*: this security notion is similar to the indistinguishability under chosen-plaintext attacks for encryption, which means that c does not help to distinguish between two candidates m_0 and m_1 as committed values.
- *Binding*: this security notion is more an unforgeability notion, which means that for any commitment c, it should be hard to open it in two different ways, which means to exhibit (m_0, d_0) and (m_1, d_1), such that $m_0 \neq m_1$ and $\mathsf{Decommit}(\ell, c, m_0, d_0) = \mathsf{Decommit}(\ell, c, m_1, d_1) = 1$.

The commitment algorithm can be interactive between the sender and the received, but the hiding and the binding properties should still hold. Several additional properties are sometimes required:

- *Extractability*: an indistinguishable Setup procedure also outputs a trapdoor that allows a extractor to get the committed value m from any commitment c. More precisely, if c can be open in a valid way, the extractor can get this value from the commitment.
- *Equivocability*: an indistinguishable Setup procedure also outputs a trapdoor that allows a simulator to generate commitments that can thereafter be open in any way.
- *Non-Malleability*: it should be hard, from a commitment c to generate a new commitment $c' \neq c$ whose committed values are in relation.

It is well-known that any IND-CCA encryption scheme leads to a non-malleable and extractable commitment scheme [GL03].

Signature scheme. A signature scheme is defined by four algorithms:

- $\mathsf{Setup}(1^k)$, where k is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ generates a pair of keys, the verification key vk and the signing key sk;
- $\mathsf{Sign}(\mathsf{sk}, m; s)$ produces a signature σ on the input message m, under the signing key sk, and using the random coins s;
- $\mathsf{Verif}(\mathsf{vk}, m, \sigma)$ checks whether σ is a valid signature on m, w.r.t. the public key vk; it outputs 1 if the signature is valid, and 0 otherwise.

A signature scheme $\mathcal{S}$ should satisfy the following properties

- *Correctness*: for all key pair $(\mathsf{vk}, \mathsf{sk})$, all random coins s and all messages m, $\mathsf{Verif}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m; s)) = 1$.
- *Existential unforgeability under (adaptive) chosen-message attacks*: this security notion can be formalized by the following security game, where it makes use of the oracle OSign:

  - $\mathsf{OSign}(m)$: This oracle outputs a valid signature on m under the signing key sk. The input queries m are added to the list $\mathcal{SM}$.

  The success probabilities are

$$\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(\mathcal{A}) = \Pr[\mathsf{Exp}^{\mathsf{euf}}_{\mathcal{S},\mathcal{A}}(k) = 1] \quad \mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(k,t) = \max_{\mathcal{A} \leq t}\{\mathsf{Succ}^{\mathsf{euf-cma}}_{\mathcal{S}}(\mathcal{A})\}.$$

$\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathcal{S},\mathcal{A}}(k)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$
2. $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk}, \mathsf{OSign}(\cdot))$
4. $b \leftarrow \mathsf{Verif}(\mathsf{vk}, m^*, \sigma^*)$
5. `IF` $M \in \mathcal{SM}$ `RETURN` 0
6. `ELSE RETURN` b

Smooth Projective Hash Functions A smooth projective hash function system is defined on a language L, with five algorithms:

- $\mathsf{Setup}(1^k)$ generates the system parameters, according to a security parameter k;
- $\mathsf{HashKG}(L)$ generates a hashing key hk for the language L;
- $\mathsf{ProjKG}(\mathsf{hk}, L, W)$ derives the projection key hp, possibly depending on a word W;
- $\mathsf{Hash}(\mathsf{hk}, L, W)$ outputs the hash value from the hashing key;
- $\mathsf{ProjHash}(\mathsf{hp}, L, W, w)$ outputs the hash value from the projection key and the witness w that $W \in L$.

The correctness of the scheme assures that if W is in L with w as a witness, then the two ways to compute the hash values give the same result: $\mathsf{Hash}(\mathsf{hk}, L, W) = \mathsf{ProjHash}(\mathsf{hp}, L, W, w)$. In our setting, these hash values will belong to a group $\mathbb{G}$. The security is defined through two different notions, the smoothness and the pseudo-randomness properties, where we use the distribution $\Delta(L, W) = \{(\mathsf{hk}, \mathsf{hp}), \mathsf{hk} \leftarrow \mathsf{HashKG}(L), \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk}, L, W)\}$:

- the *smoothness* property guarantees that if $W \notin L$, the hash value is *statistically* indistinguishable from a random element, even knowing hp:

$$\{(\mathsf{hp}, G), (\mathsf{hk}, \mathsf{hp}) \leftarrow \Delta(L, W), G \leftarrow \mathsf{Hash}(\mathsf{hk}, L, W)\} \approx_s \{(\mathsf{hp}, G), (\mathsf{hk}, \mathsf{hp}) \leftarrow \Delta(L, W), G \xleftarrow{\$} \mathbb{G}\}.$$

  We define by $\mathsf{Adv}^{\mathsf{smooth}}$ the statistical distance between the two distributions.
- the *pseudo-randomness* property guarantees that even for a word $W \in L$, but without the knowledge of a witness w, the hash value is *computationally* indistinguishable from a random element, even knowing hp:

$$\{(\mathsf{hp}, G), (\mathsf{hk}, \mathsf{hp}) \leftarrow \Delta(L, W), G \leftarrow \mathsf{Hash}(\mathsf{hk}, L, W)\} \approx_c \{(\mathsf{hp}, G), (\mathsf{hk}, \mathsf{hp}) \leftarrow \Delta(L, W), G \xleftarrow{\$} \mathbb{G}\}.$$

  We define by $\mathsf{Adv}^{\mathsf{pr}}(t)$ the computational distance between the two distributions for t-time distinguishers.

A.2 Computational Assumptions

The three classical assumptions we use along this paper are: the computational Diffie-Hellman (CDH), the decisional Diffie-Hellman (DDH) and the decisional Linear (DLin) assumptions. Our constructions essentially rely on the DLin assumption, that implies the CDH. It is the most general since it (presumably) holds in many groups, with or without pairing. Some more efficient instantiations will rely on the DDH assumption but in more specific groups.

Definition 2 (Computational Diffie-Hellman (CDH)). The Computational Diffie-Hellman assumption says that, in a group $(p, \mathbb{G}, g)$, when we are given (g^a, g^b) for unknown random $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to compute g^{ab}. We define by $\mathsf{Succ}^{\mathsf{cdh}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in finding g^{ab} within time t.

Definition 3 (Decisional Diffie-Hellman (DDH)). The Decisional Diffie-Hellman assumption says that, in a group $(p, \mathbb{G}, g)$, when we are given (g^a, g^b, g^c) for unknown random $a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = ab \bmod p$ (a DH tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\mathsf{ddh}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a DH tuple from a random tuple within time t.

Definition 4 (Decisional Linear Problem (DLin)). The Decisional Linear Problem [BBS04] says that, in a group $(p, \mathbb{G}, g)$, when we are given $(g^x, g^y, g^{xa}, g^{yb}, g^c)$ for unknown random $x, y, a, b \xleftarrow{\$} \mathbb{Z}_p$, it is hard to decide whether $c = a+b \bmod p$ (a linear tuple) or $c \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). We define by $\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t)$ the best advantage an adversary can have in distinguishing a linear tuple from a random tuple within time t.

A.3 Some Primitives in Symmetric Groups – Based on DLin

Linear Cramer-Shoup (LCS) encryption scheme. The Linear Cramer-Shoup encryption scheme [Sha07] can be tuned to a labeled public-key encryption scheme:

- $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3$;
- $\mathsf{KeyGen}(\mathsf{param})$ generates $\mathsf{dk} = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, and sets, for $i = 1, 2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. It also chooses a hash function $\mathfrak{H}_K$ in a collision-resistant hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r, s)$, for a message $M \in \mathbb{G}$ and two random scalars $r, s \xleftarrow{\$} \mathbb{Z}_p$, the ciphertext is $\mathcal{C} = (\mathbf{u} = (g_1^r, g_2^s, g_3^{r+s}), e = M \cdot h_1^r h_2^s, v = (c_1 d_1^{\xi})^r (c_2 d_2^{\xi})^s)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, \mathcal{C} = (\mathbf{u}, e, v))$: one first computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ and checks whether $u_1^{x_1+\xi y_1} \cdot u_2^{x_2+\xi y_2} \cdot u_3^{x_3+\xi y_3} \stackrel{?}{=} v$. If the equality holds, one computes $M = e/(u_1^{z_1} u_2^{z_2} u_3^{z_3})$ and outputs M. Otherwise, one outputs $\perp$.

This scheme is indistinguishable against chosen-ciphertext attacks, under the DLin assumption and if one uses a collision-resistant hash function $\mathcal{H}$.

Waters signature. The Waters signature [Wat05] is defined as follows:

- $\mathsf{Setup}(1^k)$: In a pairing-friendly setting $(p, \mathbb{G}, g, \mathbb{G}_T, e)$, one chooses a random vector $\boldsymbol{f} = (f_0, \ldots, f_k) \xleftarrow{\$} \mathbb{G}^{k+1}$ that defines the Waters hash function $\mathcal{F}(M) = f_0 \prod_{i=1}^{k} f_i^{M_i}$ for $M \in \{0,1\}^k$, and an extra generator $h \xleftarrow{\$} \mathbb{G}$. The global parameters param consist of all these elements $(p, \mathbb{G}, g, \mathbb{G}_T, e, \boldsymbol{f}, h)$.
- $\mathsf{KeyGen}(\mathsf{param})$ chooses a random scalar $x \xleftarrow{\$} \mathbb{Z}_p$, which defines the public verification key as $\mathsf{vk} = g^x$, and the secret signing key as $\mathsf{sk} = h^x$.
- $\mathsf{Sign}(\mathsf{sk}, M; s)$ outputs, for some random $s \xleftarrow{\$} \mathbb{Z}_p$, $\sigma = \big(\sigma_1 = \mathsf{sk} \cdot \mathcal{F}(M)^s, \sigma_2 = g^s\big)$.
- $\mathsf{Verif}(\mathsf{vk}, M, \sigma)$ checks whether $e(\sigma_1, g) \stackrel{?}{=} e(h, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_2)$.

This scheme is existentially unforgeable against (adaptive) chosen-message attacks [GMR88] under the CDH assumption.

A.4 Some Primitives in Asymmetric Groups – Based on DDH

Cramer-Shoup encryption scheme. The Cramer-Shoup encryption scheme [CS98] can be tuned into a labeled public-key encryption scheme:

- $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with a generator g
- $\mathsf{KeyGen}(\mathsf{param})$ generates $(g_1, g_2) \xleftarrow{\$} \mathbb{G}^2$, $\mathsf{dk} = (x_1, x_2, y_1, y_2, z) \xleftarrow{\$} \mathbb{Z}_p^5$, and sets, $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$. It also chooses a collision-resistant hash function $\mathfrak{H}_K$ in a hash family $\mathcal{H}$ (or simply a Universal One-Way Hash Function). The encryption key is $\mathsf{ek} = (g_1, g_2, c, d, h, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, M; r)$, for a message $M \in \mathbb{G}$ and a random scalar $r \in \mathbb{Z}_p$, the ciphertext is $C = (\ell, \mathbf{u} = (g_1^r, g_2^r), e = M \cdot h^r, v = (cd^\xi)^r)$, where v is computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.
- $\mathsf{Decrypt}(\ell, \mathsf{dk}, C)$: one first computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$ and checks whether $u_1^{x_1+\xi y_1} \cdot u_2^{x_2+\xi y_2} \stackrel{?}{=} v$. If the equality holds, one computes $M = e/(u_1^z)$ and outputs M. Otherwise, one outputs $\perp$.

This scheme is indistinguishable against chosen-ciphertext attacks, under the DDH assumption and if one uses a collision-resistant hash function $\mathcal{H}$.

Waters signature (asymmetric). This variant of the Waters signature has been proposed and proved in [BFPV11]:

- $\mathsf{Setup}(1^k)$: In a bilinear group $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, \mathfrak{g}_1, \mathbb{G}_T, e)$, one chooses a random vector $\boldsymbol{f} = (f_0, \ldots, f_k) \xleftarrow{\$} \mathbb{G}_1^{k+1}$, an extra generator $h_1 \xleftarrow{\$} \mathbb{G}_1$. The global parameters param consist of $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, \mathfrak{g}_1, \mathbb{G}_T, e, \boldsymbol{f}, h_1)$.
- $\mathsf{KeyGen}(\mathsf{param})$ chooses a random scalar $x \xleftarrow{\$} \mathbb{Z}_p$, which defines the public $\mathsf{vk} = \mathfrak{g}_1^x$, and the secret $\mathsf{sk} = h_1^x$.
- $\mathsf{Sign}(\mathsf{sk}, M; s)$ outputs, for some random $s \xleftarrow{\$} \mathbb{Z}_p$, $\sigma = \big(\sigma_1 = \mathsf{sk} \cdot \mathcal{F}(M)^s, \boldsymbol{\sigma_2} = (g_1^s, \mathfrak{g}_1^s)\big)$.
- $\mathsf{Verif}(\mathsf{vk}, M, \sigma)$ checks whether $e(\sigma_1, \mathfrak{g}_1) = e(h_1, \mathsf{vk}) \cdot e(\mathcal{F}(M), \sigma_{2,2})$, and $e(\sigma_{2,1}, \mathfrak{g}_1) = e(g_1, \sigma_{2,2})$.

This scheme is unforgeable under the following variant of the CDH assumption:

Definition 5 (The Advanced Computational Diffie-Hellman problem (CDH^+)). In a pairing-friendly environment $(p, \mathbb{G}_1, g_1, \mathbb{G}_2, \mathfrak{g}_1, \mathbb{G}_T, e)$. The CDH^+ assumption states that given $(g_1, \mathfrak{g}_1, g_1^a, \mathfrak{g}_1^a, g_1^b)$, for random $a, b \in \mathbb{Z}_p$, it is hard to compute g_1^{ab}.

B Multi Double Linear Cramer-Shoup Commitment

B.1 Multi Double Linear Cramer-Shoup ($n - \mathsf{DLCS}$) Encryption

We can extend the encryption scheme implicitly presented in Section 3 to vectors $(M_i, N_i)_{i=1,\ldots,n}$, partially IND-CCA protected, with a common ξ. It of course also includes the $n - \mathsf{LCS}$ scheme on vectors $(M_i)_i$, when ignoring the $\mathcal{C}'$ part, which is already anyway the case for the decryption oracle:

- $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with three independent generators $(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3$;
- $\mathsf{KeyGen}(\mathsf{param})$ generates $\mathsf{dk} = (x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3) \xleftarrow{\$} \mathbb{Z}_p^9$, and sets, for $i = 1, 2$, $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. It also chooses a collision-resistant hash function $\mathfrak{H}_K$. The encryption key is $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathfrak{H}_K)$.
- $\mathsf{Encrypt}(\ell, \mathsf{ek}, \boldsymbol{M}; \boldsymbol{r}, \boldsymbol{s})$, for a vector $\boldsymbol{M} \in \mathbb{G}^n$ and two vectors $\boldsymbol{r}, \boldsymbol{s} \in \mathbb{Z}_p^n$, computes

$$\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_n), \text{ where } \mathcal{C}_i = (\mathbf{u}_i = (g_1^{r_i}, g_2^{s_i}, g_3^{r_i+s_i}), e_i = M_i \cdot h_1^{r_i} h_2^{s_i}, v_i = (c_1 d_1^\xi)^{r_i} (c_2 d_2^\xi)^{s_i})$$

  with the v_i computed afterwards with $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$.
- $\mathsf{Encrypt}'(\ell, \mathsf{ek}, \boldsymbol{N}, \xi; \boldsymbol{a}, \boldsymbol{b})$, for a vector $\boldsymbol{N} \in \mathbb{G}^n$ and two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^n$, computes

$$\mathcal{C}' = (\mathcal{C}'_1, \ldots, \mathcal{C}'_n), \text{ where } \mathcal{C}'_i = (\boldsymbol{\alpha}_i = (g_1^{a_i}, g_2^{b_i}, g_3^{a_i+b_i}), \beta_i = N_i \cdot h_1^{a_i} h_2^{b_i}, \gamma_i = (c_1 d_1^\xi)^{a_i} (c_2 d_2^\xi)^{b_i})$$

  where the γ_i's are computed with the above $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$, hence the additional input. One can use both simultaneously: on input $(\ell, \mathsf{ek}, \boldsymbol{M}, \boldsymbol{N}; \boldsymbol{r}, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{b})$, the global encryption algorithm first calls $\mathsf{Encrypt}(\ell, \mathsf{ek}, \boldsymbol{M}; \boldsymbol{r}, \boldsymbol{s})$ and to get $\mathcal{C}$ and ξ, and then calls $\mathsf{Encrypt}'(\ell, \mathsf{ek}, \boldsymbol{N}, \xi; \boldsymbol{a}, \boldsymbol{b})$ to get $\mathcal{C}'$.

- $\mathsf{Decrypt}(\ell, \mathsf{dk}, \mathcal{C}, \mathcal{C}')$: one first parses $\mathcal{C} = (C_1, \ldots, C_n)$ and $\mathcal{C}' = (\mathcal{C}'_1, \ldots, \mathcal{C}'_n)$, where $C_i = (\mathbf{u}_i, e_i, v_i)$ and $\mathcal{C}'_i = (\boldsymbol{\alpha}_i, \beta_i, \gamma_i)$, for $i = 1, \ldots, n$, computes $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$ and checks whether, for $i = 1, \ldots, n$, $u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3} \stackrel{?}{=} v_i$ (but not for the γ_i's). If the equality holds, one computes $M_i = e_i/(u_{i,1}^{z_1} u_{i,2}^{z_2} u_{i,3}^{z_3})$ and $N_i = \beta_i/(\alpha_{i,1}^{z_1} \alpha_{i,2}^{z_2} \alpha_{i,3}^{z_3})$, and outputs $(\boldsymbol{M} = (M_1, \ldots, M_n), \boldsymbol{N} = (N_1, \ldots, N_n))$. Otherwise, one outputs $\perp$.
- $\mathsf{PDecrypt}(\ell, \mathsf{dk}, \mathcal{C})$: is a partial decryption algorithm that does as above but working on the $\mathcal{C}$ part only to get $\boldsymbol{M} = (M_1, \ldots, M_n)$ or $\perp$.

DLCS denotes the particular case where $n = 1$: $\mathsf{DLCS}(\ell, \mathsf{ek}, M, N; r, s, a, b) = (\mathcal{C}, \mathcal{C}')$, with

$$\mathcal{C} = (\mathbf{u} = (g_1^r, g_2^s, g_3^{r+s}), e = M \cdot h_1^r h_2^s, v = (c_1 d_1^\xi)^r (c_2 d_2^\xi)^s) = \mathsf{LCS}(\ell, \mathsf{ek}, M; r, s),$$
$$\mathcal{C}' = (\boldsymbol{\alpha}_i = (g_1^a, g_2^b, g_3^{a+b}), \beta = N \cdot h_1^a h_2^b, \gamma = (c_1 d_1^\xi)^a (c_2 d_2^\xi)^b) = \mathsf{LCS}^*(\ell, \mathsf{ek}, N, \xi; a, b)$$

where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$.

B.2 Security of the Multi Double Linear Cramer Shoup Encryption

Security model. This scheme is indistinguishable against *partial-decryption chosen-ciphertext* attacks, where a partial-decryption oracle only is available, but even when we allow the adversary to choose $\boldsymbol{M}$ and $\boldsymbol{N}$ in two different steps (see the security game below), under the DLin assumption and if one uses a collision-resistant hash function $\mathcal{H}$.

Indistinguishability against partial-decryption chosen-ciphertext attacks for vectors, in two steps: this security notion can be formalized by the following security game, where the adversary $\mathcal{A}$ keeps some internal state between the various calls $\texttt{FIND}_M$, $\texttt{FIND}_N$ and $\texttt{GUESS}$. In the first stage $\texttt{FIND}_M$, it receives the encryption key ek; in the second stage $\texttt{FIND}_N$, it receives the encryption of $\boldsymbol{M}_b$: $\mathcal{C}^* = \mathsf{Encrypt}(\ell, \mathsf{ek}, \boldsymbol{M}_b)$; in the last stage $\texttt{GUESS}$ it receives the encryption of $\boldsymbol{N}_b$: $\mathcal{C}'^* = \mathsf{Encrypt}'(\ell, \mathsf{ek}, \xi^*, \boldsymbol{N}_b)$, where ξ^* is the value involved in $\mathcal{C}$. During all these stages, it can make use of the oracle $\mathsf{ODecrypt}(\ell, \mathcal{C})$, that outputs the decryption of $\mathcal{C}$ under the label ℓ and the challenge decryption key dk, using $\mathsf{PDecrypt}(\ell, \mathsf{dk}, \mathcal{C})$. The input queries $(\ell, \mathcal{C})$ are added to the list $\mathcal{CT}$.

$\mathsf{Exp}_{\mathcal{E},\mathcal{A}}^{\mathsf{ind-pd-cca}-b}(k, n)$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^k)$; $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
2. $(\ell^*, \boldsymbol{M}_0, \boldsymbol{M}_1) \leftarrow \mathcal{A}(\texttt{FIND}_M : \mathsf{ek}, \mathsf{ODecrypt}(\cdot, \cdot))$
3. $\mathcal{C}^* \leftarrow \mathsf{Encrypt}(\ell^*, \mathsf{ek}, \boldsymbol{M}_b)$
4. $(\boldsymbol{N}_0, \boldsymbol{N}_1) \leftarrow \mathcal{A}(\texttt{FIND}_N : \mathcal{C}^*, \mathsf{ODecrypt}(\cdot, \cdot))$
5. $\mathcal{C}'^* \leftarrow \mathsf{Encrypt}'(\ell^*, \mathsf{ek}, \xi^*, \boldsymbol{N}_b)$
6. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : \mathcal{C}'^*, \mathsf{ODecrypt}(\cdot, \cdot))$
7. $\texttt{IF}\ \ (\ell^*, \mathcal{C}^*) \in \mathcal{CT}\ \texttt{RETURN}\ 0$
8. $\texttt{ELSE RETURN}\ b'$

The advantages are, where q_d is the number of decryption queries:

$$\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind-pd-cca}}(\mathcal{A}) = \Pr[\mathsf{Exp}_{\mathcal{E},\mathcal{A}}^{\mathsf{ind-pd-cca}-1}(k, n) = 1] - \Pr[\mathsf{Exp}_{\mathcal{E},\mathcal{A}}^{\mathsf{ind-pd-cca}-0}(k, n) = 1]$$
$$\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind-pd-cca}}(n, q_d, t) = \max_{\mathcal{A} \le t} \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind-pd-cca}}(\mathcal{A}).$$

Theorem 6. *The Multiple $n-\mathsf{DLCS}$ encryption scheme is IND-PD-CCA if $\mathcal{H}$ is a collision-resistant hash function family, under the* DLin *assumption in $\mathbb{G}$:*

$$\mathsf{Adv}_{n-\mathsf{DLCS}}^{\mathsf{ind-pd-cca}}(n, q_d, t) \le 4n \times \left(\mathsf{Adv}_{p,\mathbb{G},g}^{\mathsf{dlin}}(t) + \mathsf{Succ}_{\mathcal{H}}^{\mathsf{coll}}(t) + \frac{q_d}{p} \right).$$

Corollary 7. *The Multiple $n-\mathsf{LCS}$ encryption scheme is IND-CCA if $\mathcal{H}$ is a collision-resistant hash function family, under the* DLin *assumption in $\mathbb{G}$.*

Security proof. Let us be given a DLin challenge $(g_1, g_2, g_3, u_1 = g_1^r, u_2 = g_2^s, u_3 = g_3^t)$, for which we have to decide whether (u_1, u_2, u_3) is a linear tuple in basis (g_1, g_2, g_3), and thus $t = r + s \bmod p$, or a random one. From an IND-PD-CCA adversary $\mathcal{A}$ against the encryption scheme, we built a DLin distinguisher $\mathcal{B}$. The latter first uses (g_1, g_2, g_3) as the global parameters. It also picks $x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3 \xleftarrow{\$} \mathbb{Z}_p^9$ and sets $c_i = g_i^{x_i} g_3^{x_3}, d_i = g_i^{y_i} g_3^{y_3}, h_i = g_i^{z_i} g_3^{z_3}$, for $i = 1, 2$. It chooses a collision-resistant hash function $\mathfrak{H}_K$ and provides $\mathcal{A}$ with the encryption key $\mathsf{ek} = (c_1, c_2, d_1, d_2, h_1, h_2, \mathfrak{H}_K)$.

– In the initial game $\mathcal{G}_0$,
  - A's decryption queries are answered by $\mathcal{B}$, simply using the decryption key dk.
  - When $\mathcal{A}$ submits the first challenge vectors $\boldsymbol{M}_0 = (M_{0,1}, \ldots, M_{0,n})$ and $\boldsymbol{M}_1 = (M_{1,1}, \ldots, M_{1,n})$, with a label ℓ^*, $\mathcal{B}$ chooses a random bit $b \xleftarrow{\$} \{0, 1\}$ and encrypts $\boldsymbol{M}_b$:
    * it chooses two random vectors $\boldsymbol{r}^*, \boldsymbol{s}^* \xleftarrow{\$} \mathbb{Z}_p^n$
    * it defines $\mathcal{C}_i^* = (\mathbf{u}_i^* = (g_1^{r_i^*}, g_2^{s_i^*}, g_3^{r_i^*+s_i^*}), e_i^* = M_{b,i} \cdot h_1^{r_i^*} h_2^{s_i^*}, v_i^* = (c_1 d_1^{\xi^*})^{r_i^*} (c_2 d_2^{\xi^*})^{s_i^*})$, for $i = 1, \ldots, n$, where the v_i^*'s are computed with $\xi^* = \mathfrak{H}_K(\ell^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_n^*, e_1^*, \ldots, e_n^*)$, and $\mathcal{C}^* = (\mathcal{C}_1^*, \ldots, \mathcal{C}_n^*)$.
  - When $\mathcal{A}$ submits the second challenge vectors $\boldsymbol{N}_0 = (N_{0,1}, \ldots, N_{0,n})$ and $\boldsymbol{N}_1 = (N_{1,1}, \ldots, N_{1,n})$,
    * $\mathcal{B}$ chooses two random vectors $\boldsymbol{a}^*, \boldsymbol{b}^* \xleftarrow{\$} \mathbb{Z}_p^n$
    * it defines $\mathcal{C}_i'^* = (\boldsymbol{\alpha}_i^* = (g_1^{a_i^*}, g_2^{b_i^*}, g_3^{a_i^*+b_i^*}), \beta_i^* = N_{b,i} \cdot h_1^{a_i^*} h_2^{b_i^*}, \gamma_i^* = (c_1 d_1^{\xi^*})^{a_i^*} (c_2 d_2^{\xi^*})^{b_i^*})$, for $i = 1, \ldots, n$, where the γ_i^*'s are computed with the above $\xi^* = \mathfrak{H}_K(\ell^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_n^*, e_1^*, \ldots, e_n^*)$, and $\mathcal{C}'^* = (\mathcal{C}_1'^*, \ldots, \mathcal{C}_n'^*)$.
  - When $\mathcal{A}$ returns b', $\mathcal{B}$ outputs $b' \stackrel{?}{=} b$.

$$\Pr_0[1 \leftarrow \mathcal{B}] = \Pr_0[b' = b] = (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2.$$

– In game $\mathcal{G}_1$, where we assume $t = r + s \bmod p$, to encrypt the challenge vectors $\boldsymbol{M}_b$ and $\boldsymbol{N}_b$, $\mathcal{B}$ does as above, except for $\mathcal{C}_1^*$: $\mathcal{C}_1^* = (\mathbf{u}_1^* = (u_1, u_2, u_3), e_i^* = M_{b,1} \cdot u_1^{z_1} u_2^{z_2} u_3^{z_3}, v_1^* = u_1^{x_1+\xi^* y_1} u_2^{x_2+\xi^* y_2} u_3^{x_3+\xi^* y_3})$, which actually defines $r_1^* = r$ and $s_1^* = s$.

$$\mathbf{u}_1^* = (g_1^{r_1^*}, g_2^{s_1^*}, g_3^{r_1^*+s_1^*}) \qquad e_1^* = M_{b,1} \cdot (g_1^{r_1^*})^{z_1} (g_2^{s_1^*})^{z_2} (g_3^{r_1^*+s_1^*})^{z_3} = M_{b,1} \cdot h_1^{r_1^*} h_2^{s_1^*}$$
$$v_1^* = (g_1^{r_1^*})^{x_1+\xi^* y_1} (g_2^{s_1^*})^{x_2+\xi^* y_2} (g_3^{r_1^*+s_1^*})^{x_3+\xi^* y_3} = (c_1 d_1^{\xi^*})^{r_1^*} (c_2 d_2^{\xi^*})^{s_1^*}$$

The challenge ciphertexts are identical to the encryptions of $\boldsymbol{M}_b$ and $\boldsymbol{N}_b$ in $\mathcal{G}_0$. Decryption queries are still answered the same way. Hence the gap between this game and the previous game is 0.

$$\Pr_1[1 \leftarrow \mathcal{B}] = \Pr_0[1 \leftarrow \mathcal{B}] = (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2.$$

– In game $\mathcal{G}_2$, we now assume that $t \xleftarrow{\$} \mathbb{Z}_p$ (a random tuple). First, we have to check that the *incorrect* computation of v_1^* does not impact the probability to reject invalid ciphertexts, then we prove that e_1^* is totally independent of $M_{b,1}$.

1. About the validity checks, $u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3} \stackrel{?}{=} v_i$, where $\xi = \mathfrak{H}_K(\ell, \mathbf{u}_1, \ldots, \mathbf{u}_n, e_1, \ldots, e_n)$, three cases can appear with respect to the challenge ciphertext $\mathcal{C}^* = ((\mathbf{u}_1^*, e_1^*, v_1^*), \ldots, (\mathbf{u}_n^*, e_n^*, v_n^*))$:
   (a) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) = (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, then necessarily, for some i, $v_i \neq v_i^*$, then the check on index i will fail since one value only is acceptable;
   (b) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) \neq (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, but $\xi = \xi^*$, then the adversary has generated a collision for the hash function $\mathfrak{H}_K$.
   (c) $(\ell, \mathbf{u}_1, e_1, \ldots, \mathbf{u}_n, e_n) \neq (\ell^*, \mathbf{u}_1^*, e_1^*, \ldots, \mathbf{u}_n^*, e_n^*)$, and $\xi \neq \xi^*$: the ciphertext should be accepted iff $v_i = u_{i,1}^{x_1+\xi y_1} \cdot u_{i,2}^{x_2+\xi y_2} \cdot u_{i,3}^{x_3+\xi y_3}$, for $i = 1, \ldots, n$. To make it acceptable, if we denote $g_2 = g_1^{\beta_2}$ and $g_3 = g_1^{\beta_3}$, we indeed have

$$\begin{array}{lllll}
\log_{g_1} c_1 = x_1 & & +\beta_3 x_3 & & \\
\log_{g_1} d_1 = & & & y_1 & +\beta_3 y_3 \\
\log_{g_1} c_2 = & \beta_2 x_2 & +\beta_3 x_3 & & \\
\log_{g_1} d_2 = & & & \beta_3 y_2 & +\beta_3 y_3
\end{array}$$

with in addition,

$$\begin{aligned}
\log_{g_1} v_1^* &= rx_1 + s\beta_2 x_2 + t\beta_3 x_3 + r\xi^* y_1 + s\xi^*\beta_2 y_2 + t\xi^*\beta_3 y_3 \\
\log_{g_1} v_i^* &= r_i^* x_1 + s_i^*\beta_2 x_2 + (r_i^* + s_i^*)\beta_3 x_3 + r_i^*\xi^* y_1 + s_i^*\xi^*\beta_2 y_2 + (r_i^* + s_i^*)\xi^*\beta_3 y_3 \\
&= r_i^* \log_{g_1} c_1 + s_i^* \log_{g_1} c_2 + \xi^* r_i^* \log_{g_1} d_1 + \xi^* s_i^* \log_{g_1} c_2 \qquad \text{for } i = 2, \ldots, n \\
\log_{g_1} \gamma_i^* &= a_i^* x_1 + b_i^*\beta_2 x_2 + (a_i^* + b_i^*)\beta_3 x_3 + a_i^*\xi^* y_1 + b_i^*\xi^*\beta_2 y_2 + (a_i^* + b_i^*)\xi^*\beta_3 y_3 \\
&= a_i^* \log_{g_1} c_1 + b_i^* \log_{g_1} c_2 + \xi^* a_i^* \log_{g_1} d_1 + \xi^* b_i^* \log_{g_1} c_2 \qquad \text{for } i = 1, \ldots, n
\end{aligned}$$

The $2n-1$ last relations are thus linearly dependent with the 4 above relations, hence remains the useful relations

$$\begin{array}{lccccccc}
\log_{g_1} c_1 = & x_1 & & +\beta_3 x_3 & & & & (1) \\
\log_{g_1} d_1 = & & & & y_1 & & +\beta_3 y_3 & (2) \\
\log_{g_1} c_2 = & & \beta_2 x_2 & +\beta_3 x_3 & & & & (3) \\
\log_{g_1} d_2 = & & & & & \beta_2 y_2 & +\beta_3 y_3 & (4) \\
\log_{g_1} v_1^* = & rx_1 & +s\beta_2 x_2 & +t\beta_3 x_3 & +r\xi^* y_1 & +s\xi^*\beta_2 y_2 & +t\xi^*\beta_3 y_3 & (5)
\end{array}$$

One can note that for v_1^* to be predictable, because of the x_1, x_2 and y_1, y_2 components, we need to have $(5) = r\ (1) + s\ (3) + r\xi^*\ (2) + s\xi^*\ (4)$, and then $t = r + s$, which is not the case, hence v_1^* looks random: in this game, v_1^* is perfectly uniformly distributed in $\mathbb{G}$.
Furthermore, for any v_i in the decryption query, if $\mathbf{u}_i = (g_1^{r'}, g_2^{s'}, g_3^{t'})$ is not a linear triple, then it should be such that

$$\log_{g_1} v_i = r'x_1 + s'\beta_2 x_2 + t'\beta_3 x_3 + r'\xi y_1 + s'\xi\beta_2 y_2 + t'\xi\beta_3 y_3.$$

Since the matrix

$$\begin{pmatrix}
1 & 0 & \beta_3 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & \beta_3 \\
0 & \beta_2 & \beta_3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \beta_2 & \beta_3 \\
a & b\beta_2 & c\beta_3 & a\xi^* & b\xi^*\beta_2 & c\xi^*\beta_3 \\
r' & s'\beta_2 & t'\beta_3 & r'\xi & s'\xi\beta_2 & t'\xi\beta_3
\end{pmatrix} \quad \text{has determinant } \beta_2^2\beta_3^2(\xi^* - \xi)(t - r - s)(t' - r' - s') \neq 0,$$

then the correct value for v_i is unpredictable: an invalid ciphertext will be accepted with probability $1/p$.

2. Let us now consider the mask $u_1^{z_1} u_2^{z_2} u_3^{z_3}$: its discrete logarithm in basis g_1 is $rz_1 + s\beta_2 z_2 + t\beta_3 z_3$, whereas the informations about (z_1, z_2, z_3) are $h_1 = g_1^{z_1} g_3^{z_3}$ and $h_2 = g_2^{z_2} g_3^{z_3}$. The matrix

$$\begin{pmatrix}
1 & 0 & \beta_3 \\
0 & \beta_2 & \beta_3 \\
r & s\beta_2 & t\beta_3
\end{pmatrix} \quad \text{has determinant } \beta_2\beta_3(t - r - s)(t' - r' - s') \neq 0,$$

then the value of the mask is unpredictable: in this game, e_1^* is perfectly uniformly distributed in $\mathbb{G}$.

Since the unique difference between the two games is the linear/random tuple, unless a collision is found for $\mathfrak{H}_K$ (probability bounded by $\mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t)$) and or an invalid ciphertext is accepted (probability bounded by q_d/p), then

$$\Pr_2[1 \leftarrow \mathcal{B}] \geq \Pr_1[1 \leftarrow \mathcal{B}] - \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}.$$

– In game $\mathcal{G}_3$, to encrypt the challenge vectors $\boldsymbol{M}_b$ and $\boldsymbol{N}_b$, $\mathcal{B}$ does as above, except for $\mathcal{C}_1^*$: for a random $t_1^* \xleftarrow{\$} \mathbb{Z}_p$, $\mathbf{u}_1^* = (g_1^{r_1^*}, g_2^{s_1^*}, g_3^{t_1^*})$, $e_1^* \xleftarrow{\$} \mathbb{G}$, and $v_1^* \xleftarrow{\$} \mathbb{G}$. As just explained, this is perfectly indistinguishable with the previous game:

$$\Pr_3[1 \leftarrow \mathcal{B}] = \Pr_2[1 \leftarrow \mathcal{B}] \geq (\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1)/2 - \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}.$$

– In game $\mathcal{G}_4$, to encrypt the challenge vectors $\boldsymbol{M}_b$ and $\boldsymbol{N}_b$, $\mathcal{B}$ does as above, except for $\mathcal{C}^*$: for a random vector $\boldsymbol{t}^* \xleftarrow{\$} \mathbb{Z}_p^n$, for $i = 2, \ldots, n$: $\mathbf{u}_i^* = (g_1^{r_i^*}, g_2^{s_i^*}, g_3^{t_i^*})$, $e_i^* \xleftarrow{\$} \mathbb{G}$, and $v_i^* \xleftarrow{\$} \mathbb{G}$. Thus replacing sequentially the $\mathcal{C}_i^*$'s by random ones, as we've just done, we obtain

$$\Pr_4[1 \leftarrow \mathcal{B}] \leq \Pr_3[1 \leftarrow \mathcal{B}] - (n-1)\left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right).$$

– In game $\mathcal{G}_5$, to encrypt the challenge vectors $\boldsymbol{M}_b$ and $\boldsymbol{N}_b$, $\mathcal{B}$ does as above, except for $\mathcal{C}'^*$: for a random vector $\boldsymbol{c}^* \xleftarrow{\$} \mathbb{Z}_p^n$, for $i = 1, \ldots, n$: $\boldsymbol{\alpha}_1^* = (g_1^{a_i^*}, g_2^{b_i^*}, g_3^{c_i^*})$, $\beta_i^* \xleftarrow{\$} \mathbb{G}$, and $\gamma_i^* \xleftarrow{\$} \mathbb{G}$. Thus replacing sequentially the $\mathcal{C}_i'^*$'s by random ones, as we've just done, we obtain

$$\Pr_5[1 \leftarrow \mathcal{B}] \leq \Pr_4[1 \leftarrow \mathcal{B}] - n\left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right).$$

In this last game, it is clear that $\Pr_5[1 \leftarrow \mathcal{B}] = 1/2$, since $(\boldsymbol{M}_b, \boldsymbol{N}_b)$ is not used anymore:

$$\frac{\mathsf{Adv}^{\mathsf{ind-pd-cca}}_{n-\mathsf{DLCS}}(\mathcal{A}) - 1}{2} - 2n \times \left(\mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) - \mathsf{Succ}^{\mathsf{coll}}_{\mathcal{H}}(t) - \frac{q_d}{p}\right) \leq \frac{1}{2},$$

which concludes the proof. □

B.3 Double Linear Cramer-Shoup (DLCS) Commitment

Recently, Lindell [Lin11] proposed a highly efficient UC commitment. Our commitment strongly relies on it, but does not need to be UC secure. We will then show that the decommitment check can be done in an implicit way with an appropriate smooth projective hash function. Basically, the technique consists in encrypting M in $\mathcal{C} = (\mathbf{u}, e, v) = \mathsf{LCS}(\ell, M; r, s)$, also getting $\xi = \mathfrak{H}_K(\ell, \mathbf{u}, e)$, and then encrypting $1_{\mathbb{G}}$ in $\mathcal{C}' = \mathsf{LCS}^*(\ell, 1_{\mathbb{G}}, \xi; a, b)$, with the same ξ. For a given challenge ε, we can see $\mathcal{C} \times \mathcal{C}'^{\varepsilon} = \mathsf{LCS}^*(\ell, M, \xi; r + \varepsilon a, s + \varepsilon b)$, where the computations are done component-wise, as an encryption of M, still using the same above ξ. Note that Lindell [Lin11] used $\mathcal{C}^{\varepsilon} \times \mathcal{C}'$, but our choice seems more natural, since we essentially re-randomize the initial encryption $\mathcal{C}$, but we have to take care of choosing $\varepsilon \neq 0$. It makes use of an equivocable commitment: the Pedersen commitment [Ped92].

– $\mathsf{Setup}(1^k)$ generates a group $\mathbb{G}$ of order p, with two independent generators g and ζ;
– $\mathsf{Commit}(m; r)$, for a message $m \xleftarrow{\$} \mathbb{Z}_p$ and random coins $r \xleftarrow{\$} \mathbb{Z}_p$, produces a commitment $c = g^m \zeta^r$;
– $\mathsf{Decommit}(c, m; r)$ outputs m and r, which opens c into m, with checking ability: $c \stackrel{?}{=} g^m \zeta^r$.

This commitment is computationally binding under the discrete logarithm assumption: two different openings (m, r) and (m', r') for a commitment c, leads to the discrete logarithm of ζ in basis g, that is equal to $(m' - m) \cdot (r - r')^{-1} \bmod p$. Granted this logarithm as additional information from the setup, one can equivocate any dummy commitment.

Description. Our n-message vector commitment, which includes labels, is depicted on Figure 6, where the computation between vectors are component-wise. We assume we commit vectors of group elements, but they can come from the reversible transformation $\mathcal{G}$. Note that for this commitment scheme, we can use $\boldsymbol{\varepsilon} = (\varepsilon, \ldots, \varepsilon)$. For the version with SPHF implicit verification, according to the language, one can have to use independent components $\boldsymbol{\varepsilon} \xleftarrow{\$} (\mathbb{Z}_p^*)^n$.

Analysis. Let us briefly show the properties of this commitment:

– Hiding property: $\boldsymbol{M}$ is committed in the Pedersen commitment $\mathcal{C}''$, that does not leak any information, and in the $n - \mathsf{LCS}$ encryption $\mathcal{C}$, that is indistinguishable, even with access to the decryption oracle (extractability). This also implies non-malleability.
– Binding property: $\boldsymbol{M}$, after having been hashed, is committed in the Pedersen commitment $\mathcal{C}''$, that is computationally binding.

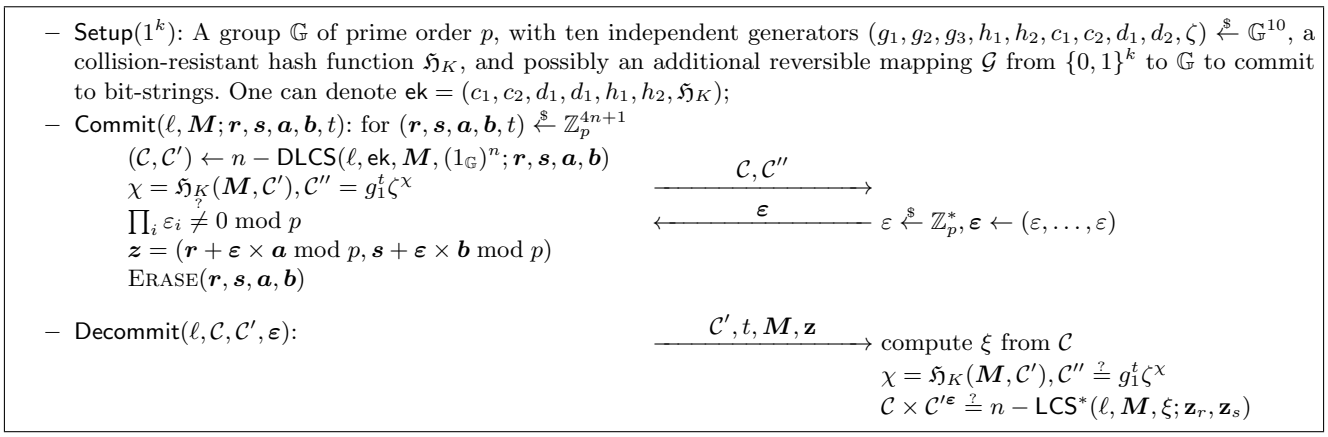

Figure 6. $n - \mathsf{DLCS}$ Commitment Scheme

– Extractability: using the decryption key of the LCS encryption scheme, one can extract $\boldsymbol{M}$ from $\mathcal{C}$. Later, one has to open the ciphertext $\mathcal{C}\mathcal{C}'^{\boldsymbol{\varepsilon}}$ with $\boldsymbol{M}'$, which can be different from $\boldsymbol{M}$ in the case that $\mathcal{C}'$ contains $\boldsymbol{N} \neq (1_\mathbb{G})^n$. But then $\boldsymbol{M}' = \boldsymbol{M} \times \boldsymbol{N}^{\boldsymbol{\varepsilon}}$, that is unpredictable at the commit time of $\mathcal{C}''$. With probability at most $1/p$, one can open the commitment with a value $\boldsymbol{M}'$ different from $\boldsymbol{M}$, if this value $\boldsymbol{M}'$ has been correctly anticipated in $\mathcal{C}''$.
– Equivocability: if one wants to open with $\boldsymbol{M}'$, one can compute $\boldsymbol{N} = (\boldsymbol{M}'/\boldsymbol{M})^{1/\boldsymbol{\varepsilon}}$, encrypt $\boldsymbol{N}$ in $\mathcal{C}' = n - \mathsf{LCS}^*(\ell, \boldsymbol{N}, \xi; \boldsymbol{a}, \boldsymbol{b})$, and update χ and t, using the Pedersen trapdoor for equivocability.

To allow an implicit verification with SPHF, one omits to send $\boldsymbol{M}$ and $\mathbf{z}$, but make an implicit proof of their existence. Therefore, $\boldsymbol{M}$ cannot be committed/verified in $\mathcal{C}''$, which has an impact on the binding property: $\mathcal{C}$ and $\mathcal{C}''$ are not binded to a specific $\boldsymbol{M}$, even in a computational way. However, as said above, if $\mathcal{C}''$ contains a ciphertext $\mathcal{C}'$ of $\boldsymbol{N} \neq (1_\mathbb{G})^n$, the actual committed value will depend on $\boldsymbol{\varepsilon}$: $\boldsymbol{M}' = \boldsymbol{M}\boldsymbol{N}^{\boldsymbol{\varepsilon}}$ has its i-component, where $N_i \neq 1_\mathbb{G}$, uniformly distributed in $\mathbb{G}$ when ε is uniformly distributed in $\mathbb{Z}_p^*$. In addition, if $\boldsymbol{\varepsilon} \xleftarrow{\$} (\mathbb{Z}_p^*)^n$, all these i-component where $N_i \neq 1_\mathbb{G}$ are randomly and independently distributed in $\mathbb{G}$. Then, if the committed value has to satisfy a specific relation, with very few solutions, $\boldsymbol{M}'$ will unlikely satisfy it.

C Smooth Projective Hash Functions on More Complex Languages

C.1 Basic Relations

We first consider Diffie-Hellman pairs and linear tuples and show we can make proof of membership without using any pairing.

DDH pairs. Let us assume a user is given two elements g, h and then wants to send $G = g^a, H = h^a$ for a chosen a and prove that the pair (G, H) is well-formed with respect to (g, h). We thus consider the language of Diffie Hellman tuples $(g, h, G = g^a, H = h^a)$, with a as a witness.

As done in [CS98], we define a projection key $\mathsf{hp} = g^{x_1} h^{x_2}$ by picking two random scalars $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$, which define the secret hashing key $\mathsf{hk} = (x_1, x_2)$. One can then compute the hash value in two different ways: $\mathsf{ProjHash}(\mathsf{hp}, (g, h, G, H), a) \stackrel{\mathsf{def}}{=} \mathsf{hp}^a = (g^{ax_1} h^{ax_2}) = G^{x_1} H^{x_2} \stackrel{\mathsf{def}}{=} \mathsf{Hash}(\mathsf{hk}, (g, h, G, H))$.

Such SPHF is smooth: this can be seen by proceeding like in the Cramer-Shoup proof. Given $\mathsf{hp} = g^\alpha$, $h = g^\beta$, $G = g^a$ and $H = h^{a'}$, the hash value is g^γ that satisfies:

$$\begin{pmatrix} \alpha \\ \gamma \end{pmatrix} = \begin{pmatrix} 1 & \beta \\ a & \beta a' \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The determinant of this matrix is $\Delta = \beta(a' - a)$, that is zero if and only if we do have a valid Diffie-Hellman tuple. Otherwise, from hp, γ is perfectly hidden, from an information theoretical point of view, and so is $\mathsf{Hash}(\mathsf{hk}, (g, h, G, H))$ too.

DLin tuples. Let us consider three generators u, v, w, and a tuple $U = u^r, V = v^s, W = w^t$ one wants to prove be linear (*i.e.* $t = r + s$). We first define two projection keys $\mathsf{hp}_1 = u^{x_1}w^{x_3}, \mathsf{hp}_2 = v^{x_2}w^{x_3}$, for random scalars that define the secret hashing key $\mathsf{hk} = (x_1, x_2, x_3)$. One can then compute the hash value in two different ways: $\mathsf{ProjHash}(\mathsf{hp}_1, \mathsf{hp}_2, (u, v, w, U, V, W), r, s) \stackrel{\mathsf{def}}{=} \mathsf{hp}_1^r\mathsf{hp}_2^s = (u^{rx_1}v^{sx_2}w^{x_3(r+s)}) = U^{x_1}V^{x_2}W^{x_3} \stackrel{\mathsf{def}}{=} \mathsf{Hash}(\mathsf{hk}, (u, v, w, U, V, W))$.

Once again this SPHF can be shown to be smooth: given $\mathsf{hp}_1 = u^\alpha$, $\mathsf{hp}_2 = u^\beta$, $v = u^\gamma$, $w = u^\delta$, the hash value is u^λ that satisfies:

$$\begin{pmatrix}\alpha\\ \beta\\ \lambda\end{pmatrix} = \begin{pmatrix}1 & 0 & \delta\\ 0 & \gamma & \delta\\ r & \gamma s & \delta t\end{pmatrix} \cdot \begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}$$

The determinant of this matrix is $\Delta = \gamma\delta(t - s - r)$, that is zero if and only if we do have a valid linear tuple.

C.2 Smooth Projective Hashing on Commitments

We now show that our commitments LCS or DLCS' are well-suited for a use together with smooth projective hash functions: instead of publishing $\boldsymbol{z}$ at the decommit phase, in order to check whether $\mathcal{C}\times\mathcal{C}'^{\varepsilon} \stackrel{?}{=} \mathsf{LCS}^*(\ell, M, \xi; z_r, z_s)$ (with $\varepsilon = 0$ in the LCS non-equivocable case, or with $\varepsilon \neq 0$ in the DLCS' case), one uses a smooth projective hash function to "implicitly" prove the existence of a witness that the commitment actually contains the claimed (or assumed) value M. We will thereafter be able to use this primitive in Language-Authenticated Key Exchange, for complex languages.

Smooth projective hash functions. We thus have a commitment, either $\mathcal{C}$ or $\mathcal{C}\cdot\mathcal{C}'^{\varepsilon}$, but we use in both cases the notation $\mathcal{C}$, and want to check whether there exists $\mathbf{z} = (z_r, z_s)$ such that

$$\mathcal{C} = \mathsf{LCS}^*(\ell, M, \xi; z_r, z_s) = (\mathbf{u} = (g_1^{z_r}, g_2^{z_s}, g_3^{z_r+z_s}), e = M\cdot h_1^{z_r}h_2^{z_s}, v = v_1^{z_r}v_2^{z_s}),$$

where we denote $v_1 = c_1d_1^{\xi}$ and $v_2 = c_2d_2^{\xi}$. We note here that all the bases g_1, g_2, g_3, h_1, h_2 but also v_1, v_2 are known as soon as ξ is known (the $\mathcal{C}$ part of the DLCS' commitment). One then generates $\mathsf{hk} = (\eta, \theta, \kappa, \lambda, \mu) \stackrel{\$}{\leftarrow} \mathbb{Z}_p^5$, and derives the projection key that depends on ξ only: $\mathsf{hp} = (\mathsf{hp}_1 = g_1^\eta g_3^\kappa h_1^\lambda v_1^\mu, \mathsf{hp}_2 = g_2^\theta g_3^\kappa h_2^\lambda v_2^\mu)$. Then, one can compute the hash value:

$$H = \mathsf{Hash}(\mathsf{hk}, M, \mathcal{C}) \stackrel{\mathsf{def}}{=} u_1^\eta u_2^\theta u_3^\kappa (e/M)^\lambda v^\mu = \mathsf{hp}_1^{z_r}\mathsf{hp}_2^{z_s} \stackrel{\mathsf{def}}{=} \mathsf{ProjHash}(\mathsf{hp}, M, \mathcal{C}; z_r, z_s) = H'.$$

Security properties. Let us claim and prove the security properties:

Theorem 8. *Under the* DLin *assumption, the above smooth projective hash function is both smooth and pseudo-random:*

- *Smoothness:* $\mathsf{Adv}_{\Pi}^{\mathsf{smooth}} = 0$*;*
- *Pseudo-Randomness:* $\mathsf{Adv}_{\Pi}^{\mathsf{pr}}(t) \leq \mathsf{Adv}_{p,\mathbb{G},g}^{\mathsf{dlin}}(t)$.

Proof. For the correctness, one can easily check that if $\mathcal{C}$ contains $M = M'$, then $H = H'$:

$$\begin{aligned} H &= u_1^\eta u_2^\theta u_3^\kappa (e/M)^\lambda v^\mu = (g_1^{z_r})^\eta (g_2^{z_s})^\theta (g_3^{z_r+z_s})^\kappa (h_1^{z_r}h_2^{z_s}M'/M)^\lambda (v_1^{z_r}v_2^{z_s})^\mu\\ &= (g_1^\eta g_3^\kappa h_1^\lambda v_1^\mu)^{z_r} \times (g_2^\theta g_3^\kappa h_2^\lambda v_2^\mu)^{z_s} \times (M'/M)^\lambda = \mathsf{hp}_1^{z_r}\mathsf{hp}_2^{z_s} \times (M'/M)^\lambda = H' \times (M/M')^\lambda \end{aligned}$$

Smoothness: if $\mathcal{C}$ is not a correct encryption of M, then H is unpredictable: let us denote M' and z_s' such that $\mathcal{C} = (\boldsymbol{u} = (g_1^{z_r}, g_2^{z_s}, g_3^{z_t}), e = M'h_1^{z_r}h_2^{z_s}, v = v_1^{z_r}v_2^{z_s'})$. Then, if we denote $g_2 = g_1^{\beta_2}$ and $g_3 = g_1^{\beta_3}$, and $h_1 = g_1^{\rho_1}$ and $h_2 = g_1^{\rho_2}$, but also $v_1 = g_1^{\delta_1}$ and $v_2 = g_1^{\delta_2}$, and $\Delta = \log_{g_1}(M'/M)$:

$$\begin{aligned} H &= g_1^{\eta z_r} g_1^{\beta_2\theta z_s} g_1^{\beta_3\kappa z_t} (M'/M)^\lambda (g_1^{\rho_1 z_r + \rho_2 z_s})^\lambda (v_1^{z_r}v_2^{z_s'})^\mu\\ \log_{g_1} H &= \eta z_r + \beta_2\theta z_s + \beta_3\kappa z_t + \lambda(\rho_1 z_r + \rho_2 z_s) + \mu(\delta_1 z_+ \delta_2 z_s') + \lambda\Delta \end{aligned}$$

The information leaked by the projected key is $\log_{g_1} \mathsf{hp}_1 = \eta+\beta_3\kappa+\rho_1\lambda+\delta_1\mu$ and $\log_{g_1} \mathsf{hp}_2 = \beta_2\theta+\beta_3\kappa+\rho_2\lambda+\delta_2\mu$, which leads to the matrix

$$\begin{pmatrix} 1 & 0 & \beta_3 & \rho_1 & \delta_1 \\ 0 & \beta_2 & \beta_3 & \rho_2 & \delta_2 \\ z_r & \beta_2 z_s & \beta_3 z_t & \Delta + \rho_1 z_r + \rho_2 z_s & \delta_1 z_r + \delta_2 z'_s \end{pmatrix}$$

One remarks that if $z_t \neq z_r + z_s \bmod p$, then the three rows are not linearly dependent even considering the 3 first components only, and then H is unpredictable. Hence, we can assume that $z_t = z_r + z_s \bmod p$. The third row must thus be the first multiplied by z_r plus the second multiplied by z_s: $\rho_2 z_s = \Delta + \rho_2 z_s \bmod p$ and $z_s = z'_s \bmod p$, which implies $z'_s = s$ and $\Delta = 0$, otherwise, H remains unpredictable.

As a consequence, if $\mathcal{C}$ is not a correct encryption of W, H is perfectly unpredictable in $\mathbb{G}$:

$$\begin{aligned} &\{(\mathsf{hp}, H), \mathsf{hk} = (\eta, \theta, \kappa, \lambda, \mu) \xleftarrow{\$} \mathbb{Z}_p^5, \mathsf{hp} = (\mathsf{hp}_1 = g_1^\eta g_3^\kappa h_1^\lambda v_1^\mu, \mathsf{hp}_2 = g_2^\theta g_3^\kappa h_2^\lambda v_2^\mu), H \leftarrow \mathsf{Hash}(\mathsf{hk}, M, \mathcal{C})\} \\ &\approx_s \{(\mathsf{hp}, H), \mathsf{hk} = (\eta, \theta, \kappa, \lambda, \mu) \xleftarrow{\$} \mathbb{Z}_p^5, \mathsf{hp} = (\mathsf{hp}_1 = g_1^\eta g_3^\kappa h_1^\lambda v_1^\mu, \mathsf{hp}_2 = g_2^\theta g_3^\kappa h_2^\lambda v_2^\mu), H \xleftarrow{\$} \mathbb{G}\}. \end{aligned}$$

Pseudo-Randomness: we've just shown that if $\mathcal{C}$ is not a correct encryption of M, then H is statistically unpredictable. Let us be given a triple (g_1, g_2, g_3) together with another triple $\boldsymbol{u} = (u_1 = g_1^a, u_2 = g_2^b, u_3 = g_3^c)$. We choose random exponents $(x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3)$, and for $i = 1, 2$, we set $c_i = g_i^{x_i} g_3^{x_3}$, $d_i = g_i^{y_i} g_3^{y_3}$, and $h_i = g_i^{z_i} g_3^{z_3}$. We generate $\mathcal{C} = (\boldsymbol{u}, e = M \times u_1^{z_1} u_2^{z_2} u_3^{z_3}, v = u_1^{x_1+\xi y_1} u_2^{x_2+\xi y_2} u_3^{x_3+\xi y_3})$. If $c = a + b \bmod p$ (i.e., $\boldsymbol{u}$ is a linear tuple in basis $\boldsymbol{g}$), then $\mathcal{C}$ is a valid encryption of M, otherwise this is not, and we can apply the smoothness property:

$$\mathsf{Adv}^{\mathsf{pr}}_{\Pi}(t) \leq \mathsf{Adv}^{\mathsf{smooth}}_{\Pi} + \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t) \leq \mathsf{Adv}^{\mathsf{dlin}}_{p,\mathbb{G},g}(t).$$

□

C.3 Single Equation

Let us assume that we have $\mathcal{Y}_i$ committed in $\mathbb{G}$, in $\boldsymbol{c}_i$, for $i = 1, \ldots, m$ and $\mathcal{Z}_i$ committed in $\mathbb{G}_T$, in $\boldsymbol{D}_i$, for $i = m+1, \ldots, n$, and we want to show they simultaneously satisfy

$$\left(\prod_{i=1}^{m} e(\mathcal{Y}_i, \mathcal{A}_i)\right) \cdot \left(\prod_{i=m+1}^{n} \mathcal{Z}_i^{\mathfrak{z}_i}\right) = \mathcal{B}$$

where $\mathcal{A}_i \in \mathbb{G}$, $\mathcal{B} \in \mathbb{G}_T$, and $\mathfrak{z}_i \in \mathbb{Z}_p$ are public. As already said, the commitment can either be the LCS or the DLCS' version, but they both come up to a ciphertext $\mathcal{C}$ with the appropriate random coins $\mathbf{z}$:

$\boldsymbol{c}_i = (\boldsymbol{u_i} = (g_1^{z_{r_i}}, g_2^{z_{s_i}}, g_3^{z_{r_i}+z_{s_i}}), e_i = h_1^{z_{r_i}} h_2^{z_{s_i}} \cdot \mathcal{Y}_i, v_i = (c_1 d_1^\xi)^{z_{r_i}} \cdot (c_2 d_2^\xi)^{z_{s_i}})$ for $i = 1, \ldots, m$
which can be transposed into $\mathbb{G}_T$:
$\boldsymbol{C}_i = (\boldsymbol{U_i} = (G_{i,1}^{z_{r_i}}, G_{i,2}^{z_{s_i}}, G_{i,3}^{z_{r_i}+z_{s_i}}), E_i = H_{i,1}^{z_{r_i}} H_{i,2}^{z_{s_i}} \cdot \mathcal{Z}_i, V_i = (C_{i,1} D_{i,1}^\xi)^{z_{r_i}} \cdot (C_{i,2} D_{i,2}^\xi)^{z_{s_i}})$ for $i = 1, \ldots, m$
where, for $j = 1, 2, 3$, $G_{i,j} = e(g_j, \mathcal{A}_i)$ and for $j = 1, 2$, $H_{i,j} = e(h_j, \mathcal{A}_i)$, $C_{i,j} = e(c_j, \mathcal{A}_i)$, $D_{i,j} = e(d_j, \mathcal{A}_i)$,
but also, $\mathcal{Z}_i = e(\mathcal{Y}_i, \mathcal{A}_i)$, and
$\boldsymbol{D}_i = (\boldsymbol{U_i} = (G_{i,1}^{z_{r_i}}, G_{i,2}^{z_{s_i}}, G_{i,3}^{z_{r_i}+z_{s_i}}), E_i = H_{i,1}^{z_{r_i}} H_{i,2}^{z_{s_i}} \cdot \mathcal{Z}_i, V_i = (C_{i,1} D_{i,1}^\xi)^{z_{r_i}} \cdot (C_{i,2} D_{i,2}^\xi)^{z_{s_i}})$ for $i = m+1, \ldots, n$
where, for $j = 1, 2, 3$, $G_{i,j} = e(g_j, g)$ and for $j = 1, 2$, $H_{i,j} = e(h_j, g)$, $C_{i,j} = e(c_j, g)$, $D_{i,j} = e(d_j, g)$,

where $\xi = \mathfrak{H}_K(\boldsymbol{u_1}, \ldots, \boldsymbol{u_m}, \boldsymbol{U_{m+1}}, \ldots, \boldsymbol{U_n}, e_1, \ldots, e_m, E_{m+1}, \ldots, E_n)$: $\mathbb{G}$-elements are encrypted under $\mathsf{ek} = (\boldsymbol{g} = (g_1, g_2, g_3), \boldsymbol{h} = (h_1, h_2), \boldsymbol{c} = (c_1, d_1), \boldsymbol{d} = (c_2, d_2))$, and $\mathbb{G}_T$-element are encrypted under $\mathsf{EK}_i = (\boldsymbol{G}_i = (G_{i,1}, G_{i,2}, G_{i,3}), \boldsymbol{H}_i = (H_{i,1}, H_{i,2}), \boldsymbol{C}_i = (C_{i,1}, C_{i,2}), \boldsymbol{D}_i = (D_{i,1}, D_{i,2}))$. Note that an additional label ℓ can be included in the computation of ξ.

For the hashing keys, one picks scalars $(\lambda, (\eta_i, \theta_i, \kappa_i, \mu_i)_{i=1,\ldots,n}) \xleftarrow{\$} \mathbb{Z}_p^{4n+1}$, and sets $\mathsf{hk}_i = (\eta_i, \theta_i, \kappa_i, \lambda, \mu_i)$. One then computes the projection keys as $\mathsf{hp}_i = (g_1^{\eta_i} g_3^{\kappa_i} h_1^\lambda (c_1 d_1^\xi)^{\mu_i}, g_2^{\theta_i} g_3^{\kappa_i} h_2^\lambda (c_2 d_2^\xi)^{\mu_i}) \in \mathbb{G}^2$. The associated projection keys in $\mathbb{G}_T$ are $\mathsf{HP}_i = (e(\mathsf{hp}_{i,1}, \mathcal{A}_i), e(\mathsf{hp}_{i,2}, \mathcal{A}_i))$, for $i = 1, \ldots, n$, where $\mathcal{A}_i = g^{\mathfrak{z}_i}$ for $i = m+1, \ldots, n$.

The hash value is

$$H = \left(\prod_i U_{i,1}^{\eta_i} \cdot U_{i,2}^{\theta_i} \cdot U_{i,3}^{\kappa_i} \cdot E_i^{\lambda} \cdot V_i^{\mu_i}\right) \times \mathcal{B}^{-\lambda}$$
$$= \left(\prod_i \mathsf{HP}_{i,1}^{r_i} \mathsf{HP}_{i,2}^{s_i} \mathcal{Z}_i^{\lambda}\right) \times \mathcal{B}^{-\lambda} = \left(\prod_i \mathsf{HP}_{i,1}^{r_i} \mathsf{HP}_{i,2}^{s_i}\right) \left(\left(\prod_{i=1}^{m} e(\mathcal{Y}_i, \mathcal{A}_i)\right) \left(\prod_{i=m+1}^{n} \mathcal{Z}_i\right) / \mathcal{B}\right)^{\lambda}$$
$$= \prod_i \mathsf{HP}_{i,1}^{z_{r_i}} \mathsf{HP}_{i,2}^{z_{s_i}} = \left(\prod_{i=1}^{m} e(\mathsf{hp}_{i,1}^{z_{r_i}}, \mathcal{A}_i) \cdot e(\mathsf{hp}_{i,2}^{z_{s_i}}, \mathcal{A}_i)\right) \times \left(e(\prod_{i=m+1}^{n} \mathsf{hp}_{i,1}^{z_{r_i}}, g^{\mathfrak{z}_i}) \cdot e(\prod_{i=m+1}^{n} \mathsf{hp}_{i,2}^{z_{s_i}}, g^{\mathfrak{z}_i})\right)$$

which can be computed either from the commitments and the hashing keys, or from the projection keys and the witnesses. We prove below the smoothness, but first extend it even more to several equations.

C.4 Multiple Equations

Let us assume that we have $\mathcal{Y}_i$ committed in $\mathbb{G}$, in $\boldsymbol{c}_i$, for $i = 1, \ldots, m$ and $\mathcal{Z}_i$ committed in $\mathbb{G}_T$, in $\boldsymbol{D}_i$, for $i = m+1, \ldots, n$, and we want to show they simultaneously satisfy

$$\left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i})\right) \cdot \left(\prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}}\right) = \mathcal{B}_k, \text{ for } k = 1, \ldots, t.$$

where $\mathcal{A}_{k,i} \in \mathbb{G}$, $\mathcal{B}_k \in \mathbb{G}_T$, and $\mathfrak{z}_{k,i} \in \mathbb{Z}_p$, as well as $A_k \subseteq \{1, \ldots, m\}$ and $B_k \subseteq \{m+1, \ldots, n\}$ are public. As above, from the commitments, one derives the global ξ, which can also involves the label ℓ, and one can also derive the commitments in $\mathbb{G}_T$, $\boldsymbol{C}_{k,i}$ that correspond to the encryption of $\mathcal{Z}_{k,i} = e(\mathcal{Y}_i, \mathcal{A}_{k,i})$ under the keys $\mathsf{EK}_{k,i} = (\boldsymbol{G}_{k,i} = (G_{k,i,1}, G_{k,i,2}, G_{k,i,3}), \boldsymbol{H}_{k,i} = (H_{k,i,1}, H_{k,i,2}), \boldsymbol{C}_{k,i} = (C_{k,i,1}, C_{k,i,2}), \boldsymbol{D}_{k,i} = (D_{k,i,1}, D_{k,i,2}))$, where the capital letters $X_{k,i,j}$ correspond to the lower-case letters x_j paired with $\mathcal{A}_{k,i}$.

For the hashing keys, one picks scalars $(\lambda, \{\eta_i, \theta_i, \kappa_i, \mu_i\}_{i=1,\ldots,n}) \xleftarrow{\$} \mathbb{Z}_p^{4n+1}$, $\{\varepsilon_k\}_{k=1,\ldots,t} \xleftarrow{\$} \mathbb{Z}_p^t$ and sets $\mathsf{hk} = (\{\mathsf{hk}_i = (\eta_i, \theta_i, \kappa_i, \lambda, \mu_i)\}_{i=1,\ldots,n}, \{\varepsilon_k\}_{k=1,\ldots,t})$. We insist on the fact that the ε_k's have to be sent after the commitments have been sent, or at least committed to (such as $\mathcal{C}$ and $\mathcal{C}''$ which prevent from any modification). One then computes the projection keys as $\mathsf{hp}_i = (g_1^{\eta_i} g_3^{\kappa_i} h_1^{\lambda} (c_1 d_1^{\xi})^{\mu_i}, g_2^{\theta_i} g_3^{\kappa_i} h_2^{\lambda} (c_2 d_2^{\xi})^{\mu_i}) \in \mathbb{G}^2$, together with ε_k. The associated projection keys in $\mathbb{G}_T$ are $\mathsf{HP}_{k,i} = (e(\mathsf{hp}_{i,1}, \mathcal{A}_{k,i}), e(\mathsf{hp}_{i,2}, \mathcal{A}_{k,i}))$, for $t = 1, \ldots, t$ and $i = 1, \ldots, n$, where $\mathcal{A}_{k,i} = g^{\mathfrak{z}_{k,i}}$ for $i = m+1, \ldots, n$, together with ε_k. The hash function and the projective hash function are defined as:

$$H = \prod_k \left(\left(\prod_{i \in A_k \cup B_k} U_{k,i,1}^{\eta_i} \cdot U_{k,i,2}^{\theta_i} \cdot U_{k,i,3}^{\kappa_i} \cdot E_{k,i}^{\lambda} \cdot V_{k,i}^{\mu_i}\right) \times \mathcal{B}_k^{-\lambda}\right)^{\varepsilon_k}$$
$$= \prod_k \left(\prod_{i \in A_k \cup B_k} \mathsf{HP}_{k,i,1}^{z_{r_i}} \cdot \mathsf{HP}_{k,i,2}^{z_{s_i}}\right)^{\varepsilon_k} \times \prod_k \left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}} \times \mathcal{B}_k^{-1}\right)^{\lambda \varepsilon_k}$$
$$H' = \prod_k \left(\prod_{i \in A_k \cup B_k} \mathsf{HP}_{k,i,1}^{z_{r_i}} \cdot \mathsf{HP}_{k,i,2}^{z_{s_i}}\right)^{\varepsilon_k}$$

which can be computed either from the commitments and the hashing keys, or from the projection keys and the witnesses. They lead to the same values $H' = H$ if

- for every k, $\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}} = \mathcal{B}_k$, which means that all the equations are simultaneously satisfied;
- $\lambda = 0$, which is quite unlikely;
- $\prod_k \Delta_k^{\varepsilon_k} = 1$, where for every k, $\Delta_k = \prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{z}_{k,i}} / \mathcal{B}_k$, which is also quite unlikely since the Δ_k's are fixed before the ε_k's are known.

C.5 Security Analysis

Smoothness. In this section, first we prove the smoothness of the SPHF built right before. For $k = 1$, this proves the smoothness of the SPHF built to handle variables in one linear pairing equation. The list of commitments $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$, which possibly results from the multiplication by the companion ciphertext when using the equivocable variant, should be considered in the language if and only if:

- the commitments are all valid Linear Cramer-Shoup ciphertexts (in either $\mathbb{G}$ or $\mathbb{G}_T$), with the common and fixed ξ;
- the plaintexts satisfy the linear pairing product equations.

Let us assume that one of the commitments is not a valid ciphertext, this means that for some index $i \in \{1, \ldots, n\}$, the ciphertext $(\boldsymbol{U_i}, E_i, V_i)$ in $\mathbb{G}_T$ satisfies $(\boldsymbol{U_i} = (G_1^{r_i}, G_2^{s_i}, G_3^{t_i}), V_i)$ with either $t_i \neq r_i + s_i$ or $V_i \neq (C_1 D_1^{\xi})^{r_i} \cdot (C_2 D_2^{\xi})^{s_i}$. Then, the contribution of this ciphertext in the hash value is $(U_{i,1}^{\eta_i} \cdot U_{i,2}^{\theta_i} \cdot U_{i,3}^{\kappa_i} \cdot E_i^{\lambda} \cdot V_i^{\mu_i})^{\varepsilon'_i}$, where $\varepsilon'_i = \sum_{k, i \in A_k \cup B_k} \varepsilon_k$, knowing the projection keys that reveal, at most,

$$\log_{g_1} \mathsf{hp}_{i,1} = \eta_i + x_3 \times \kappa_i + x_4 \times \lambda + (y_1 + \xi y_3) \times \mu_i \quad \log_{g_1} \mathsf{hp}_{i,2} = x_2 \times \theta_i + x_3 \times \kappa_i + x_5 \times \lambda + (y_2 + \xi y_4) \times \mu_i,$$

where $g_2 = g_1^{x_2}$ $g_3 = g_1^{x_3}$ $h_1 = g_1^{x_4}$ $h_2 = g_1^{x_5}$ $c_1 = g_1^{y_1}$ $c_2 = g_1^{y_2}$ $d_1 = g_1^{y_3}$ $d_2 = g_1^{y_4}$. This contribution is thus $(G_1^{r_i\eta_i + x_2 s_i \theta_i + x_3 t_i \kappa_i + z_i \mu_i} \times E_i^{\lambda})^{\varepsilon'_i}$, where $V_i = G_1^{z_i}$. But even if all the discrete logarithms were known, and also λ, one has to guess $r_i\eta_i + x_2 s_i \theta_i + x_3 t_i \kappa_i + z_i \mu_i$, given $\eta_i + x_3 \times \kappa_i + (y_1 + \xi y_3) \times \mu_i$ and $x_2 \times \theta_i + x_3 \times \kappa_i + (y_2 + \xi y_4) \times \mu_i$:

$$\begin{pmatrix} 1 & 0 & x_3 & (y_1 + \xi y_3) \\ 0 & x_2 & x_3 & (y_2 + \xi y_4) \\ r_i & x_2 s_i & x_3 t_i & z_i \end{pmatrix}.$$

The first 3-column matrix has determinant is $x_2 x_3 (t_i - (r_i + s_i))$, that is non-zero as soon as $t_i \neq r_i + s_i$. In this case, there is no way to guess the correct value better than by chance: $1/p$. If $t_i = (r_i + s_i)$, the third line is linearly dependent with the 2 first, if and only if $z_i = r_i(y_1 + \xi y_3) + s_i(y_2 + \xi y_4)$. Otherwise, one has no better way to guess the value than by chance either. Hence the smoothness of this hash function when one commitment is not valid.

About the equation validity, the E_i's of the involved ciphertexts contain plaintexts $\mathcal{Y}_i$ or $\mathcal{Z}_i$, and contribute to the hash value: from the projection keys, the k-th equation contributes to

$$H_k = \left(\prod_{i \in A_k} \mathsf{HP}_{k,i,1}^{r_i} \cdot \mathsf{HP}_{k,i,2}^{s_i} \times \prod_{i \in B_k} \left(\mathsf{HP}_{i,1}^{r_i} \cdot \mathsf{HP}_{i,2}^{s_i} \right)^{\mathfrak{d}_{k,i}} \right)^{\varepsilon_k} \times \left(\prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{d}_{k,i}} \times \mathcal{B}_k^{-1} \right)^{\lambda \varepsilon_k}$$

Let us denote $\alpha_k = \prod_{i \in A_k} e(\mathcal{Y}_i, \mathcal{A}_{k,i}) \times \prod_{i \in B_k} \mathcal{Z}_i^{\mathfrak{d}_{k,i}} \times \mathcal{B}_k^{-1}$, then the uncertainty about H is $(\prod_k \alpha_k^{\varepsilon_k})^{\lambda}$. As soon as one of the equations is not satisfied, one of the α_k is different from 1. Since the ε_k's are unknown at the commitment time, one cannot make the α_k to compensate themselves, but by chance: if one equation is not satisfied, the probability that $\prod_k \alpha_k^{\varepsilon_k} = 1$ is $1/p$. Except this negligible case, $(\prod_k \alpha_k^{\varepsilon_k})^{\lambda}$ is totally unpredictable since λ is random.

Pseudo-randomness. The pseudo-randomness can be proven under the DLin assumption: with invalid ciphertexts, the smoothness guarantees unpredictability; without the witnesses, one cannot distinguish a valid ciphertext from an invalid ciphertext.

C.6 Asymmetric Setting

Our approach has been presented in the symmetric setting (at least when pairing are required). We can do the same in asymmetric bilinear groups, with $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and even more efficiently, using the Cramer-Shoup encryption scheme, and the analogous n-message commitment scheme, which security relies on the DDH assumption in either $\mathbb{G}_1$ or $\mathbb{G}_2$. In this setting, our methodology can handle linear pairing product equations:

$$\left(\prod_{i=1}^{m} e(\mathcal{X}_i, \mathcal{B}_i) \right) \cdot \left(\prod_{j=1}^{n} e(\mathcal{A}_j, \mathcal{Y}_j) \right) \cdot \left(\prod_{k=1}^{o} \mathcal{Z}_k^{\mathfrak{d}_k} \right) = g_T,$$

where $\mathcal{A}_j, \mathcal{B}_i, g_T$ are public values, in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively, and $\mathcal{X}_i, \mathcal{Y}_j, \mathcal{Z}_k$ are the unknown values, committed in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

D Security of the LAKE Protocol: Proof of Theorem 1

For the sake of simplicity, we give in Figure 7 an explicit version of the protocol described in Figure 4. We omit the additional verification that all the committed values are in the correct subsets $\mathcal{P}$ and $\mathcal{S}$, since in the proof below we will always easily guarantee this membership. The proof heavily relies on the properties of the commitments and smooth projective hash functions given in Sections 3, 4 and Appendix B.

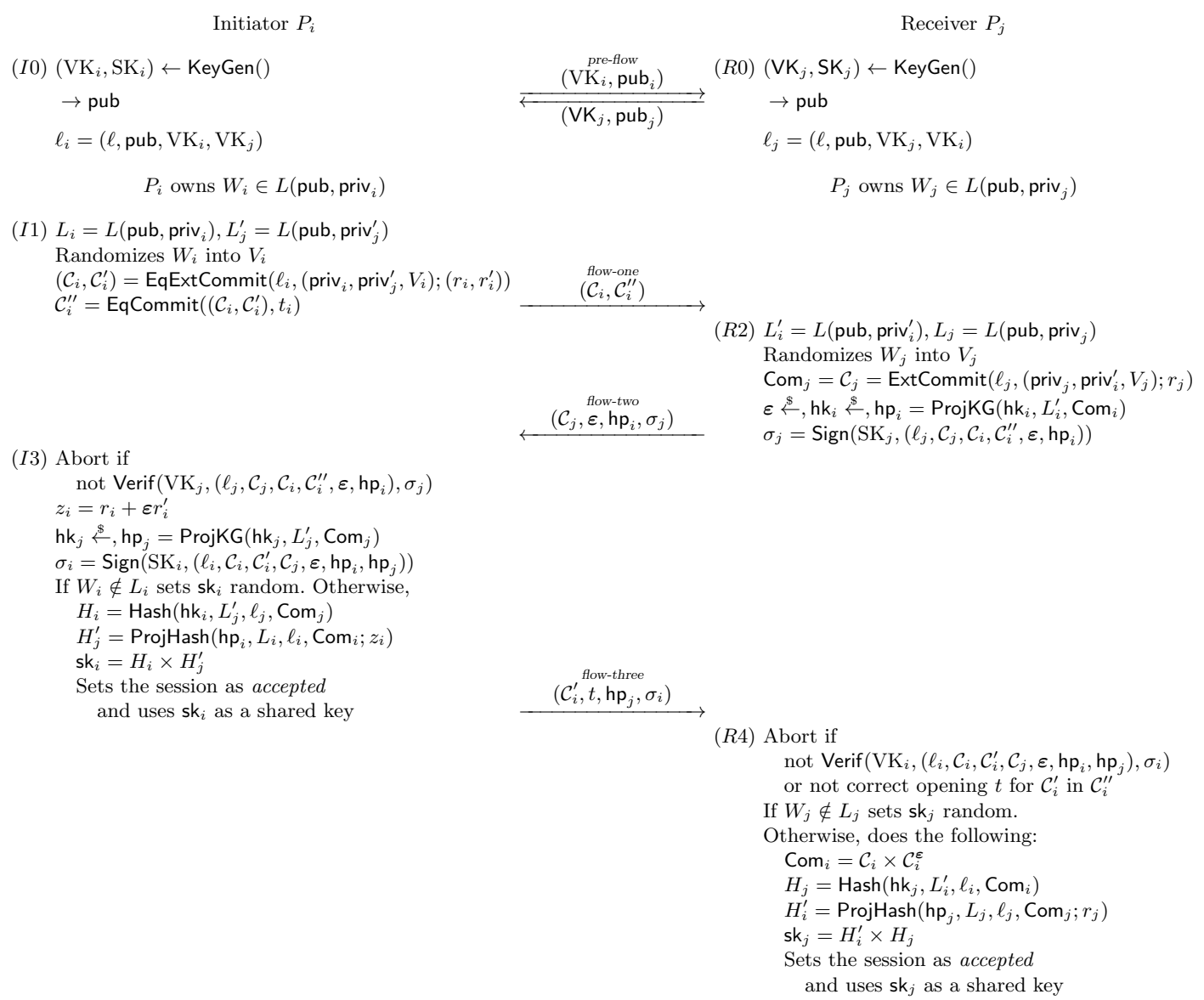

Figure 7. Description of the language authenticated key exchange protocol for players (P_i, ssid), with index i, message $W_i \in L_i = L(\mathsf{pub}, \mathsf{priv}_i)$ and expected language for P_j $L'_j = L(\mathsf{pub}, \mathsf{priv}'_j)$ and (P_j, ssid), with index j, message $W_j \in L_j = L(\mathsf{pub}, \mathsf{priv}_j)$ and expected language for P_i $L'_i = L(\mathsf{pub}, \mathsf{priv}'_i)$. The label is $\ell = (\mathrm{sid}, \mathrm{ssid}, P_i, P_j)$. The random values used in the commitments (witnesses) are all included in r_i and r_j.

D.1 Notations

The protocol is played between an initiator, denoted to as P_i, and a receiver, P_j. Each player P_k owns a public part pub_k of a language. Those two public parts pub_i and pub_j will combine to create the common public part pub of the language used in the protocol. Player P_k also owns a private part priv_k and a word $W_k \in L(\mathsf{pub}, \mathsf{priv}_k)$[1].

[1] Since pub is unknown before the beginning of the protocol, one can imagine that P_k knows several words W_k, corresponding to different possibilities for the public part pub_ℓ its partner can choose. Once pub is set, P_k chooses a word $W_k \in L(\mathsf{pub}, \mathsf{priv}_k)$ among them or aborts the protocol if this public value does not correspond to one it had in mind.

It rerandomizes this word W_k into a word V_k still in $L(\mathsf{pub}, \mathsf{priv}_k)$: we assume the languages used to be self-randomizable, which allows such a rerandomization.

We need three different types of commitments for this protocol:

- $\mathsf{EqCommit}$ is an equivocable commitment, such as Pedersen [Ped92], used to engage P_i on its further committed values $\mathcal{C}_i$ and $\mathcal{C}'_i$ with randomness t_i: $\mathcal{C}''_i = \mathsf{EqCommit}((\mathcal{C}_i, \mathcal{C}'_i); t_i)$;
- $\mathsf{EqExtCommit}$ is a labeled equivocable and extractable commitment, used by P_i to commit to its private values (used in the smooth projective hash function) and asking P_j to send a challenge value $\boldsymbol{\varepsilon}$.
  It is based on a double encryption scheme (Enc_i and Enc'_i) that is partial-decryption chosen-ciphertext secure (the latter one being strongly related to the former), verifying the following properties, if we denote by $+$ and $\times$ two group laws adapted to the schemes:

$$\begin{aligned} \mathcal{C}_i &= \mathsf{Enc}_i(\ell_i, m_i; r_i) \\ \mathcal{C}'_i &= \mathsf{Enc}'_i(\ell_i, n_i; r'_i) \\ \mathsf{Com}_i &= \mathsf{Enc}'_i(\ell_i, m_i \times n_i^{\varepsilon}; r_i + \varepsilon r'_i) = \mathcal{C}_i \mathcal{C}'^{\varepsilon}_i \end{aligned}$$

  In the particular cases of (multi) Double-Cramer-Shoup or Double-Linear-Cramer-Shoup, $\mathcal{C}_i$ is a real ciphertext with the correct ξ value, to guarantee non-malleability, but $\mathcal{C}'_i$ and Com_i use the ξ value of $\mathcal{C}_i$. This is the reason why projection keys can be computed as soon as $\mathcal{C}_i$ is known.
- $\mathsf{ExtCommit}$ is a labeled extractable commitment, used by P_j to commit to its private values (used in the smooth projective hash function). It is based on a chosen-ciphertext secure encryption scheme Enc_j which can be equal to Enc_i or different: $\mathsf{Com}_j = \mathcal{C}_j = \mathsf{ExtCommit}(\ell_j, m_j; r_j) = \mathsf{Enc}_j(\ell_j, m_j; r_j)$

Again, note that the projected keys of the smooth projective hash functions depend on $\mathcal{C}_i$ and $\mathcal{C}_j$ only, and do not need Com_i, justifying it can be computed by P_j in $(R2)$, before having actually received $\mathcal{C}'_i$ and thus being able to compute Com_i.

D.2 Sketch of Proof

The proof follows that of [CHK+05] and [ACP09], but with a different approach since we want to prove that the best attack the adversary can perform is to play as an honest player would do with a chosen credential $(\mathsf{pub}_i, \mathsf{priv}_i, \mathsf{priv}'_j, W_i)$ —when trying to impersonate P_i— or $(\mathsf{pub}_j, \mathsf{priv}_j, \mathsf{priv}'_i, W_j)$ —when trying to impersonate P_j—. In order to prove Theorem 1, we need to construct, for any real-world adversary $\mathcal{A}$ (controlling some dishonest parties), an ideal-world adversary $\mathcal{S}$ (interacting with dummy parties and the split functionality $s\mathcal{F}_{\text{LAKE}}$) such that no environment $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ in the real world and $\mathcal{S}$ in the ideal world with non-negligible probability.

The split functionality $s\mathcal{F}_{\text{LAKE}}$ is defined in Section 5, following [BCL+05]. In particular, we assume that at the beginning of the protocol, $\mathcal{S}$ receives from it the contribution pub_i of P_i to the public language pub as answer to the Init query sent by the environment on behalf of this player. The preflow phase will determine the whole public language pub.

When initialized with security parameter k, the simulator first generates the CRS for the commitment (public parameters but also extraction and equivocation trapdoors), as well as the possibly required trapdoors to be able to generate, for any pub, a word inside or outside the language $L(\mathsf{pub}, \mathsf{priv})$ when priv is known. It then initializes the real-world adversary $\mathcal{A}$, giving it these values. The simulator then starts its interaction with the environment $\mathcal{Z}$, the functionality $s\mathcal{F}_{\text{LAKE}}$ and its subroutine $\mathcal{A}$.

Since we are in the static-corruption model, the adversary can only corrupt players before the execution of the protocol. We assume players to be honest or not at the beginning, and they cannot be corrupted afterwards. However, this does not prevent the adversary from modifying flows coming from the players. Indeed, since we are in a weak authenticated setting, when a player acts dishonestly (even without being aware of it), it is either corrupted, hence the adversary knows its private values and acts on its behalf; or the adversary tries to impersonate it with chosen/guessed inputs. In both cases, we say the player is $\mathcal{A}$-controlled. Following [CHK+05], we say that a flow is *oracle-generated* if it was sent by an honest player and arrives without any alteration to the player it was meant to. We say it is *non-oracle-generated* otherwise, that is if it was sent by a $\mathcal{A}$-controlled player (which means corrupted, or which flows have been modified by the adversary). The one-time signatures are aimed at avoiding changes of players during a session: if *pre-flow* is oracle-generated for P_i, then *flow-one*

and *flow-three* cannot be non-oracle-generated without causing the protocol to fail because of the signature, for which the adversary does not know the signing key. Similarly, for P_j. On the other hand, if *pre-flow* is non-oracle-generated for P_i, then *flow-one* and *flow-three* cannot be oracle-generated without causing the protocol to fail, since the honest player would sign wrong flows (the flows the player sent before the adversary alters them). In both cases, the verifications of the signatures will fail at Steps ($I3$) or ($R4$) and P_i or P_j will abort. One can note that if there is one flow only in the protocol for one player, its signature is not required, which is the case for P_j when there is no pub to agree on at the beginning. But this is just an optimization that can be occasionally applied, as for the PAKE protocol. We do not consider it here.

To deal with both cases of $\mathcal{A}$-controlled players (either corrupted or impersonated by the adversary), we use the Split Functionality model (see Section 2). We thus add a *pre-flow* which will help us know which players are honest and which ones are $\mathcal{A}$-controlled. If one player is honest and the other one corrupted, the adversary will send the *pre-flow* on behalf of the latter, and the simulator will have to send the *pre-flow* on behalf of the former. But in the case where both players are honest at the beginning of the protocol, both *pre-flow* will have to be sent by $\mathcal{S}$ on behalf of these players and the adversary can then decide to modify one of these flows. This models the fact that the adversary can decide to split a session between P_i and P_j by answering itself to P_i, and thus trying to impersonate P_j with respect to P_i, and doing the same with P_j. Then, the Split Functionality model ensures that two independent sessions are created (with sub-session identifiers). We can thus study these sessions independently, which means that we can assume, right after the *pre-flow*, that either a player is honest if its *pre-flow* is oracle-generated, or $\mathcal{A}$-controlled if the *pre-flow* is non-oracle-generated. Since we want to show that the best possible attack for the adversary (by controlling a player) consists in playing honestly with a trial credential, we have to show that the view of the environment is unchanged if we simulate this dishonest player as an honest player with respect to ideal functionality. The simulator then has to transform its flows into queries to the Ideal Functionality $s\mathcal{F}_{\text{LAKE}}$, and namely the $\mathsf{NewSession}$-query. Still, the $\mathcal{A}$-controlled player is not honest, and can have a bad behavior when sending the real-life flows, but then either it has no strong impact, and it is similar to an honest behavior, or it will make the protocol fail: we cannot avoid the adversary to make denial of service attack, and the adversary will learn nothing.

As explained in [BCL$^+$05] and [ACGP11], where the simulator actually had access to a $\mathsf{TestPwd}$ query to the functionality, it is equivalent to grant the adversary the right to test a password (here a credential) for P_i while trying to play on behalf of P_j (i.e., use a $\mathsf{TestPwd}$ query) or to use the split functionality model and generate the $\mathsf{NewSession}$ queries corresponding to the $\mathcal{A}$-controlled players and see how the protocol terminates, since it corresponds to a trial of one credential by the adversary (one-line dictionary attack).

The proof will thus consist in generating ideal queries (and namely the $\mathsf{NewSession}$) when receiving non-oracle-generated flows from $\mathcal{A}$-controlled players, and generating real messages for the honest players (whose $\mathsf{NewSession}$ queries will be received from the environment). This will be done in a indistinguishable way for the environment.

We assume from now on that we know in which case we are (i.e.how many players are $\mathcal{A}$-controlled), and the pub part is fixed. We then describe the simulator for each of these cases, while it has generated the *pre-flow* for the honest players by generating $(\text{VK}, \text{SK}) \leftarrow \mathsf{KeyGen}()$, and thus knows the signing keys. We denote by $L_i = L(\mathsf{pub}, \mathsf{priv}_i)$ the language used by P_i, and by $L'_j = L(\mathsf{pub}, \mathsf{priv}'_j)$ the language that P_i expects P_j to use. We use the same notations in the reverse direction. As explained in Section 1, recall that the languages considered depend on two possibly different relations: $L_i = L_{\mathcal{R}_i}(\mathsf{pub}, \mathsf{priv}_i)$ and $L_j = L_{\mathcal{R}_j}(\mathsf{pub}, \mathsf{priv}_j)$, but we omit them for the sake of clarity. Note that the simulator will use the NewKey query to learn whether the protocol is a success or a failure (in case a player is $\mathcal{A}$-controlled). This will enable it to check whether the LAKE should fulfill, that is, whether the two users play with compatible words and languages, i.e.. $\mathsf{priv}'_i = \mathsf{priv}_i$, $\mathsf{priv}'_j = \mathsf{priv}_j$, $W_i \in L_i$ and $W_j \in L_j$. For the most part, the interaction is implemented by the simulator $\mathcal{S}$ just following the protocol on behalf of all the honest players.

D.3 Description of the Simulators

Initialization and Simulation of *pre-flow*. This is the beginning of the simulation of the protocol, where $\mathcal{S}$ has to send the message *pre-flow* on behalf of each non-corrupted player[2].

[2] Note that $\mathcal{S}$ only has to send one of these flows if one player is corrupted.

Step ($I0$). When receiving the first $(\mathsf{Init} : \text{ssid}, P_i, P_j, \mathsf{pub}_i)$ from $s\mathcal{F}_{\text{LAKE}}$ as answer to the Init query sent by the environment on behalf of P_i, $\mathcal{S}$ starts simulating the new session of the protocol for party P_i, peer P_j, session identifier ssid. $\mathcal{S}$ chooses a key pair $(\text{SK}_i, \text{VK}_i)$ for a one-time signature scheme and generates a *pre-flow* message with the values $(\text{VK}_i, \mathsf{pub}_i)$. It gives this message to $\mathcal{A}$ on behalf of (P_i, ssid).

Step ($R0$). When receiving the second $(\mathsf{Init} : \text{ssid}, P_j, P_i, \mathsf{pub}_j)$ from $s\mathcal{F}_{\text{LAKE}}$ as answer to the Init query sent by the environment on behalf of P_j, $\mathcal{S}$ starts simulating the new session of the protocol for party P_j, peer P_i, session identifier ssid. $\mathcal{S}$ chooses a key pair $(\text{SK}_j, \text{VK}_j)$ for a one-time signature scheme and generates a *pre-flow* message with the values $(\text{VK}_j, \mathsf{pub}_j)$. It gives this message to $\mathcal{A}$ on behalf of (P_j, ssid).

Splitting the Players. As just said, thanks to the Split Functionality model, according to which flows were transmitted or altered by $\mathcal{A}$, we know from the *pre-flow* which player(s) is (are) honest and which player(s) is (are) $\mathcal{A}$-controlled, and the public part pub. We can consider each case independently after the initial split, during which $\mathcal{S}$ generated the signing keys of the honest players. Thanks to the signature in the last flows for each player, if the adversary tries to take control on behalf of a honest user for some part of the execution (without learning the internal states, since we exclude adaptive corruptions), the verification will fail. Then we can assume that the sent flows are the received flows.

One can note that the prior agreement on pub allows to simulate P_i before having received any information from P_j, and also without knowing whether the protocol should be a success or not. Without such an agreement, the simulator would not know which value to use for pub whereas it cannot change its mind later, since it is sent in clear. Everything else is committed: either in an equivocable way on behalf of P_i so that we can change it later when we know the real status of the session; or in a non-equivocable way on behalf of P_j since we can check the status of the session before making this commitment. Of course, both commitments are extractable. In the whole proof, in case the extraction fails, the simulator acts as if the simulation should fail. Indeed, the language of the smooth projective hash function not only verifies the equations, but also that the ciphertext is valid, and this verification will fail.

We come back again to the case of our equivocable commitment with SPHF that is not a really extractable/binding commitment since the player can open it in a different way one would extract it, in case the second ciphertext does not encrypt $1_{\mathbb{G}}$: if extraction leads to an inconsistent tuple, there is little chance that with the random $\boldsymbol{\varepsilon}$ it becomes consistent; if extraction leads to a consistent tuple, there is little chance that with the random $\boldsymbol{\varepsilon}$ it remains consistent, and then the real-life protocol will fail, whereas the ideal-one was successful at the NewKey-time. But then, because of the positive NewKey-answer, the $\mathsf{SendKey}$-query takes the key-input into consideration, that is random on the initiator side because of the SPHF on an invalid word, and thus indistinguishable from the environment point of view from a failed session: this is a denial of service, the adversary should already be aware of.

Hence, the three simulations presented below exploit the properties of our commitments and SPHF to make the view of the environment indistinguishable from a real-life attack, just using the simulator $\mathcal{S}$ that is allowed to interact with the ideal functionality on behalf of players, but in an honest way only, since the functionality is perfect and does not know bad behavior.

During all these simulations, $\mathcal{S}$ knows the equivocability trapdoor of the commitment and the decryption keys of the two encryption schemes.

Case 1: P_i is $\mathcal{A}$-controlled and P_j is honest. In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the honest player P_j, for which it has simulated the *pre-flow* and thus knows the signing key, and has to simulate the queries to the functionality as if the $\mathcal{A}$-controlled player P_i was honest.

Step ($I1$). This step is taken care of by the adversary, who sends its *flow-one*, from which $\mathcal{S}$ extracts $(\mathsf{priv}_i, \mathsf{priv}'_j)$ only. No need to extract W_i, but one generates a random valid $V_i \in L(\mathsf{pub}, \mathsf{priv}_i)$ (we have assumed the existence of a trapdoor in the CRS to generate such valid words). $\mathcal{S}$ then sends the query $(\mathsf{NewSession} : \text{ssid}', P_i, P_j, V_i, L_i = L(\mathsf{pub}, \mathsf{priv}_i), L'_j = L(\mathsf{pub}, \mathsf{priv}'_j), \mathsf{initiator})$ to $\mathcal{F}_{\text{LAKE}}$ on behalf of P_i.

Step ($R2$). The $\mathsf{NewSession}$ query for this player (P_j, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid′). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_j, P_i, \mathsf{pub}, \mathsf{receiver})$ and makes a call NewKey to the functionality to check the success of the protocol. It actually tells whether the languages are consistent, but does not tell anything about the validity of

the word submitted by the adversary for P_i. It indeed receives the answer in the name of P_i. In case of a success, $\mathcal{S}$ generates a word $V_j \in L(\mathsf{pub}, \mathsf{priv}'_j)$ and uses $\mathsf{priv}_j = \mathsf{priv}'_j$ and $\mathsf{priv}'_i = \mathsf{priv}_i$ for this receiver session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and produces a commitment $\mathcal{C}_j$ on the tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, V_j)$. Otherwise, $\mathcal{S}$ produces a commitment $\mathcal{C}_j$ on a dummy tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, V_j)$. It then generates a challenge value $\boldsymbol{\varepsilon}$ and the hashing keys $(\mathsf{hk}_i, \mathsf{hp}_i)$ on $\mathcal{C}_i$. It sends the *flow-two* message $(\mathcal{C}_j, \boldsymbol{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to $\mathcal{A}$ on behalf of P_j, where σ_j is the signature on all the previous information.

Step $(I3)$. This step is taken care of by the adversary, who sends its *flow-three*.

Step $(R4)$. Upon receiving $m = (\textit{flow-three}, \mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$, $\mathcal{S}$ makes the verification checks, and possibly aborts. In case of correct checks, $\mathcal{S}$ already knows whether the protocol should succeed, thanks to the NewKey query. If the protocol is a success, then $\mathcal{S}$ computes receiver session key honestly, and makes a SendKey to P_j. Otherwise, $\mathcal{S}$ makes a SendKey to P_j with a random key that will anyway not be used.

Case 2: P_i is honest and P_j is $\mathcal{A}$-controlled. In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the honest player P_i, for which it has simulated the *pre-flow* and thus knows the signing key, and has to simulate the queries to the functionality as if the $\mathcal{A}$-controlled player P_j was honest.

Step $(I1)$. The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid′). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ and generates a *flow-one* message by committing to a dummy tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$. It gives this commitment $(\mathcal{C}_i, \mathcal{C}''_i)$ to $\mathcal{A}$ on behalf of (P_i, ssid').

Step $(R2)$. This step is taken care of by the adversary, who sends its *flow-two* $= (\textit{flow-two}, \mathcal{C}_j, \boldsymbol{\varepsilon}, \mathsf{hp}_i, \sigma_i)$, from which $\mathcal{S}$ first checks the signature, and thereafter extracts the committed triple $(\mathsf{priv}_j, \mathsf{priv}'_i, W_j)$. $\mathcal{S}$ then sends the query $(\mathsf{NewSession} : \text{ssid}', P_j, P_i, W_j, L_j = L(\mathsf{pub}, \mathsf{priv}_j), L'_i = L(\mathsf{pub}, \mathsf{priv}'_i), \mathsf{receiver})$ to $\mathcal{F}_{\text{LAKE}}$ on behalf of P_j.

Step $(I3)$. $\mathcal{S}$ makes a NewKey query to the functionality to know whether the protocol should succeed. It indeed receives the answer in the name of P_j. In case of a success, $\mathcal{S}$ generates a word $V_i \in L(\mathsf{pub}, \mathsf{priv}'_i)$ and uses $\mathsf{priv}_i = \mathsf{priv}'_i$ for this initiator session (we have assumed the existence of a trapdoor in the CRS to generate such valid words) and then uses the equivocability trapdoor to update $\mathcal{C}'_i$ and t in order to contain the new consistent tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$ with respect to the challenge $\boldsymbol{\varepsilon}$. If the protocol should be a success, then $\mathcal{S}$ computes initiator session key honestly, and makes a SendKey to P_i. Otherwise, $\mathcal{S}$ makes a SendKey to P_i with a random key that will anyway not be used. $\mathcal{S}$ sends the *flow-three* message $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to $\mathcal{A}$ on behalf of P_i, where σ_i is the signature on all the previous information.

Step $(R4)$. This step is taken care of by the adversary.

Case 3: P_i and P_j are honest. In this case, $\mathcal{S}$ has to simulate the concrete messages in the real-life from the two honest players P_i and P_j, for which it has simulated the *pre-flow* and thus knows the signing keys. But since no player is controlled by $\mathcal{A}$, the NewKey query will not provide any answer to the simulator. But thanks to the semantic security of the commitments, dummy values can be committed, no external adversary will make any difference.

Step $(I1)$. The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid′). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{initiator})$ and generates a *flow-one* message by committing to a dummy tuple $(\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$. It gives this commitment $(\mathcal{C}_i, \mathcal{C}''_i)$ to $\mathcal{A}$ on behalf of (P_i, ssid').

Step $(R2)$. The NewSession query for this player (P_i, ssid') has been automatically transferred from the split functionality $s\mathcal{F}_{\text{LAKE}}$ to $\mathcal{F}_{\text{LAKE}}$ (transforming the session identifier from ssid to ssid′). $\mathcal{S}$ receives the answer $(\mathsf{NewSession} : \text{ssid}, P_j, P_i, \mathsf{pub}, \mathsf{receiver})$ and generates a commitment $\mathcal{C}_j$ on a dummy tuple $(\mathsf{priv}_j, \mathsf{priv}'_i, V_j)$. It then generates a challenge value $\boldsymbol{\varepsilon}$ and the hashing keys $(\mathsf{hk}_i, \mathsf{hp}_i)$ on $\mathcal{C}_i$. It sends the *flow-two* message $(\mathcal{C}_j, \boldsymbol{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ to $\mathcal{A}$ on behalf of P_j, where σ_j is the signature on all the previous information.

Step $(I3)$. When the session $(P_i; \text{ssid}')$ receives the message $m = (\textit{flow-two}, \mathcal{C}_j, \boldsymbol{\varepsilon}, \mathsf{hp}_i, \sigma_j)$ from its peer session $(P_j; \text{ssid}')$, the signature is necessarily correct. Then, $\mathcal{S}$ makes a SendKey to P_i with a random key that will anyway not be used, since no player is corrupted. $\mathcal{S}$ sends the *flow-three* message $(\mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ to $\mathcal{A}$ on behalf of P_i, where σ_i is the signature on all the previous information.

STEP ($R4$). When the session $(P_j; \mathrm{ssid}')$ receives the message $m = (\textit{flow-three}, \mathcal{C}'_i, t, \mathsf{hp}_j, \sigma_i)$ from its peer session $(P_i; \mathrm{ssid}')$, the signature is necessarily correct. $\mathcal{S}$ makes a $\mathsf{SendKey}$ to P_j with a random key that will anyway not be used, since no player is corrupted.

D.4 Description of the Games

We now provide the complete proof by a sequence of games, where we replace the triple $(\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$ by the notation T_i, and the triple $(\mathsf{priv}_j, \mathsf{priv}'_i, V_j)$ by the notation T_j, with component-wise operations to simplify notations. Similarly, for cleaner notations, we use non-vector notations for the ciphertexts, the random coins and the challenge ε, but all the computations are assumed to be performed component-wise, and thus implicitly use vectors.

We insist that we are considering static corruptions only, and with the split-functionality, we already know which players are corrupted and verification keys for the one-time signatures are known to the two players, and fixed: either honestly generated (honest player) or adversary-generated (corrupted players).

Game $\boldsymbol{G}_0$: This is the real game, where every flow from honest players are generated correctly by the simulator which knows the inputs sent by the environment to the players. There is no use of the ideal functionality for the moment.

Game $\boldsymbol{G}_1$: In this game, the simulator knows the decryption key for $\mathcal{C}_i$ when generating the CRS. But this game is almost the same as the previous one except the way sk_j is generated when P_i is corrupted and P_j honest. In all the other cases, the simulator does as in $\boldsymbol{G}_0$ by playing honestly (still knowing its private values). When P_i is corrupted and P_j honest, $\mathcal{S}$ does as before until ($R4$), but then, it extracts the values committed to by the adversary in Com_i (using the decryption key for $\mathcal{C}_i$) and checks whether the private parts of the languages are consistent with the values sent to P_j by the environment. If the languages are not consistent (or decryption rejects), P_j is given a random session key sk_j.

This game is statistically indistinguishable from the former one thanks to the smoothness of the SPHF on Com_i.

Game $\boldsymbol{G}_2$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ when generating the CRS. This game is almost the same as the previous one except that $\mathcal{S}$ extracts the values committed to by the adversary in $\mathcal{C}_i$ to check consistency of the languages, and does not wait until Com_i. If the languages are not consistent (or decryption rejects), P_j is given a random session key sk_j.

The game is indistinguishable from the previous one except if Com_i contains consistent values whereas $\mathcal{C}_i$ does not, but because of the unpredictability of ε, and the Pedersen commitment that is computationally binding under the discrete logarithm problem, the probability is bounded by $1/q$.

The distance between the two games is thus bounded by the probability to break the binding property of the Pedersen commitment.

Game $\boldsymbol{G}_3$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ when generating the CRS, as in $\boldsymbol{G}_2$. Actually, in the above game, when P_i is corrupted and P_j honest, if extracted languages from $\mathcal{C}_i$ are not consistent, P_j does not have to compute hash values. The random coins are not needed anymore. In this game, in this particular case, $\mathcal{S}$ generates $\mathcal{C}_j$ with dummy values T'_j.

This game is computationally indistinguishable from the former one thanks to the $\mathsf{IND}-\mathsf{CPA}$ property of the encryption scheme involved in $\mathcal{C}_j$. To prove this indistinguishability, one makes q hybrid games, where q is the number of such sessions where P_i is corrupted and P_j is honest but extracted languages from $\mathcal{C}_i$ are not consistent with inputs to P_j. More precisely, in the k-th hybrid game G_k (for $1 \leq k \leq q$), in all such sessions before the k-th one, $\mathcal{C}_j$ is generated by encrypting T'_j, in all sessions after the k-th one, $\mathcal{C}_j$ is generated by encrypting T_j, and in the k-th session, $\mathcal{C}_j$ is generated by calling the left-or-right encryption oracle on (T_j, T'_j). It is clear that the game $\boldsymbol{G}_2$ correponds to G_1 with the "left" oracle, and the game $\boldsymbol{G}_3$ corresponds to G_q with the "right" oracle. And each time, G_k with the right oracle is identical to G_{k+1} with the "left" oracle, while every game G_k is an $\mathsf{IND}-\mathsf{CPA}$ game. It is possible to use the encryption oracle because the random coins are not needed in these sessions.

Game $\boldsymbol{G}_4$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ when generating the CRS, as in $\boldsymbol{G}_2$. Now, when P_i is corrupted and P_j honest, if extracted languages from $\mathcal{C}_i$ are consistent, $\mathcal{S}$ knows priv_j and priv'_i (the same as the values sent by the environment). It furthermore generates a random valid word V_j,

and uses it to generate the ciphertext $\mathcal{C}_j$ instead of re-randomizing the word W_j sent by the environment. $\mathcal{S}$ can compute the correct value sk_j from the random coins, and gives it to P_j.

This game is perfectly indistinguishable from the former one thanks to the self-randomizable property of the language.

Note that the value sk_j computed by $\mathcal{S}$ can be computed by the adversary if the latter indeed sent a valid word W_i in $\mathcal{C}_i$ (that is not explicitly checked in this game). Otherwise, sk_j looks random from the smoothness of the SPHF. As a consequence, on this game, sessions where P_i is corrupted and P_j is honest look ideal, while one does not need anymore the inputs from the environment sent to P_j to simulate honest players.

Game $\boldsymbol{G}_5$: We now consider the case where P_i is honest. The simulator has to simulate P_i behavior. To do so, it will know the equivocability trapdoor for the Pedersen commitment. But for other cases, the simulator still knows the decryption key for $\mathcal{C}_i$ when generating the CRS. In $(I1)$, the simulator still encrypts $T_i = (\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$ from the environment to produce $\mathcal{C}_i$. It chooses at random a dummy value $\mathcal{C}'_i$ and computes honestly the equivocable commitment $\mathcal{C}''_i$, knowing the random value t_i. In $(I3)$, after receiving ε from P_j, it chooses random coins z_i and computes Com_i as the encryption of $T_i = (\mathsf{priv}_i, \mathsf{priv}'_j, V_i)$ with the random coins z_i. (Since this is a double encryption scheme, it uses the redundancy from $\mathcal{C}_i$: namely for DLCS, it uses ξ from $\mathcal{C}_i$). Thanks to the homomorphic property, it can compute $\mathcal{C}'_i$ as $(\mathsf{Com}_i/\mathcal{C}_i)^{1/\varepsilon}$, and equivocate $\mathcal{C}"_i$. $\mathcal{C}'_i$ should be an encryption of $1_{\mathbb{G}}$ under the random coins r'_i that are implicitly defined, but unknown.

Thanks to the properties of the different commitments recalled in Section D.1, and the perfect-hiding property of the Pedersen commitment, this is a perfect simulation. It then computes the hash values honestly, using z_i.

Game $\boldsymbol{G}_6$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS. When P_i is honest, $\mathcal{S}$ generates the commitment $\mathcal{C}_i$ by choosing dummy values T'_i instead of T_i. Everything else is unchanged from $\boldsymbol{G}_5$.

This game is thus indistinguishable from the former one thanks to the $\mathsf{IND}-\mathsf{CCA}$ property of the encryption scheme involved in $\mathcal{C}_i$. As for the proof of indistinguishability of Game $\boldsymbol{G}_3$, we do a sequence of hybrid games, where $\mathcal{C}_i$ is generated be either encrypting T_i or T'_i, or asking the left-or-right oracle on (T_i, T'_i). We replace the decryption key for $\mathcal{C}_i$ by access to the decryption oracle on $\mathcal{C}_i$. Then, one has to take care that no decryption query is asked on one of the challenge ciphertexts involved in the sequence of games. This would mean that the adversary would replay in another session a ciphertext oracle-generated in another session. Because of the label which contains the verification key oracle-generated, one can safely reject the ciphertext.

Game $\boldsymbol{G}_7$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS. When P_i is honest, $\mathcal{S}$ generates the commitment $\mathcal{C}_i$ by choosing dummy values T'_i. It then computes $\mathcal{C}'_i$ by encrypting the value $(T_i/T'_i)^{1/\varepsilon}$ with randomness $z_i - r_i/\varepsilon$. This leads to the same computations of $\mathcal{C}_i$ and $\mathcal{C}'_i$ as in the former game. The rest is done as above.

This game is perfectly indistinguishable from the former one.

Game $\boldsymbol{G}_8$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS. When P_i and P_j are both honest (both initiation flows where oracle-generated), if the words and languages are correct, players are both given the same random session key $\mathsf{sk}_i = \mathsf{sk}_j$. If the words and languages are not compatible, random independent session keys are given.

Since the initiation flows ($I0$ and $R0$) contained oracle-generated verification keys, unless the adversary managed to forge signatures, all the flows are oracle-generated. First, because of the pseudo-randomness of the SPHF, H_i is unpredictable, and independent of H'_j, hence sk_i looks random. Then, if the words and languages are compatible, we already has $\mathsf{sk}_j = \mathsf{sk}_i$ in the previous game. However, if they are not compatible, either H'_i is independent of H_i, or H'_j is independent of H_j, and in any case, sk_j where already independent of sk_i in the previous game.

This game is thus computationally indistinguishable from the former one, under the pseudo-randomness of the two SPHF.

Game $\boldsymbol{G}_9$: In this above game, the hash values do not have to be computed anymore when P_i and P_j are both honest. The random coins are not needed anymore.

In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS. When P_i and P_j are both honest, $\mathcal{S}$ generates $\mathcal{C}'_i$ and $\mathcal{C}_j$ with

dummy values T'_i and T'_j. In this game, sessions where P_i and P_j are both honest look ideal, while one does not need anymore the inputs from the environment sent to P_i and P_j to simulate honest players.

This game is computationally indistinguishable from the former one thanks to the $\mathsf{IND}-\mathsf{PD}-\mathsf{CCA}$ and $\mathsf{IND}-\mathsf{CPA}$ properties of the encryption schemes involved in $\mathcal{C}'_i$ and $\mathcal{C}_j$. For the proof on indistinguishability between the two games, we make two successive sequences of hybrid games, as for the proof of indistinguishability of Game $\boldsymbol{G}_3$. One with the $\mathsf{IND}-\mathsf{PD}-\mathsf{CCA}$ game: a sequence of hybrid games, where $\mathcal{C}_i$ is generated by encrypting T'_i, and $\mathcal{C}'_i$ by encrypting either T_i or T'_i, but in the critical session, one asks for the left-or-right oracle $\mathsf{Encrypt}$ on (T'_i, T'_i), and the left-or-right oracle $\mathsf{Encrypt}'$ on (T_i, T'_i). The decryption key for $\mathcal{C}_i$ is replaced by an access to the decryption oracle on $\mathcal{C}_i$. As above, one has to take care that no decryption query is asked on a challenge ciphertext $\mathcal{C}'_i$, but the latter cannot be valid since it is computed from $\mathcal{C}_i$ values not controlled by the adversary. The second hybrid sequence uses $\mathsf{IND}-\mathsf{CPA}$ games on $\mathcal{C}_j$ exactly as in the proof of indistinguishability of Game $\boldsymbol{G}_3$.

Game $\boldsymbol{G}_{10}$: In this game, the simulator still knows the decryption key for $\mathcal{C}_i$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS, but also the decryption key for $\mathcal{C}_j$. When P_i is honest and P_j corrupted, $\mathcal{S}$ extracts the values committed to by the adversary in $\mathcal{C}_j$. It checks whether they are consistent with the values sent to P_i by the environment. If the words and languages are not consistent (or decryption rejects), P_i is given a random session key sk_i.

This game is statistically indistinguishable from the former one thanks to the smoothness of the SPHF.

Game $\boldsymbol{G}_{11}$: In this game, the simulator still knows the decryption keys for $\mathcal{C}_i$ and $\mathcal{C}_j$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS.

In the above game, when P_i is honest and P_j corrupted, if extracted values from $\mathcal{C}_j$ are not consistent, P_i does not have to compute hash values. The random coins are not needed anymore. In this game, in this particular case, $\mathcal{S}$ generates $\mathcal{C}'_i$ with dummy values T'_i.

This game is computationally indistinguishable from the former one thanks to the $\mathsf{IND}-\mathsf{PD}-\mathsf{CCA}$ property of the encryption scheme involved in $\mathcal{C}'_i$. The proof uses the same sequence of hybrid games with the $\mathsf{IND}-\mathsf{PD}-\mathsf{CCA}$ game on $(\mathcal{C}_i, \mathcal{C}'_i)$ as in the proof of indistinguishability of Game $\boldsymbol{G}_9$.

Game $\boldsymbol{G}_{12}$: In this game, the simulator still knows the decryption keys for $\mathcal{C}_i$ and $\mathcal{C}_j$ and the equivocability trapdoor for the Pedersen commitment when generating the CRS. Now, when P_i is honest and P_j corrupted, if extracted values from $\mathcal{C}_j$ are consistent, $\mathcal{S}$ knows priv_i and priv'_j (the same as the values sent by the environment). It furthermore generates a random valid word V_i, and uses it to generate the ciphertext $\mathcal{C}'_i$ instead of re-randomizing the word W_j sent by the environment. $\mathcal{S}$ can compute the correct value sk_i from the random coins, and gives it to P_i. In this game, sessions where P_i is honest and P_j is corrupted look ideal, while one does not need anymore the inputs from the environment sent to P_i to simulate honest players.

This game is perfectly indistinguishable from the former one thanks to the self-randomizable property of the language.

Game $\boldsymbol{G}_{13}$: In this game, $\mathcal{S}$ now uses the ideal functionality: $\mathsf{NewSession}$-queries for honest players are automatically forwarded to the ideal functionality, for corrupted players, they are done by $\mathcal{S}$ using the values extracted from $\mathcal{C}_i$ or $\mathcal{C}_j$. In order to check consistency of the words and languages, $\mathcal{S}$ asks for a NewKey. When one player is corrupted, it learns the outcome: success or failure. It can continue the simulation in an appropriate way.

E Complexity

In the Table 1, we give the number of elements to be sent (group elements or scalars) and the number of exponentiations to compute for each operation (commitment and SPHF), where we consider the Equality Test, and the Linear Pairing Product Equations. One has to commit all the private inputs, and then the cost for relations is just the additional overhead due to the projection keys and hashing computations once the elements are already committed: an LCSCom commitment is 5 group elements, and a $\mathsf{DLCSCom}'$ is twice more, plus the Pedersen commitment (one group element), the challenge ε (a scalar) and the opening t (a scalar). Note that a simple Linear commitment is just 3 group elements.

If the global language is a conjunction of several languages, one should simply add all the costs, and consider the product of all the sub-hashes as the final hash value from the SPHF.

DLin	$\mathbb{G}$	$\mathbb{Z}_p$	Exp.	CSCom	$\mathbb{G}$	$\mathbb{Z}_p$	Exp.
LCSCom	$5n$	0	$7n+2$	CSCom	$4n$	0	$4n+1$
DLCSCom	$10n+1$	2	$18n+6$	DCSCom	$8n+1$	2	$12n+5$
Equality	2	0	14	Equality	1	0	10
LPPE	$2n+1$	0	$10n+11$	LPPE	$n+1$	0	$7n+9$

Table1. Computational and Communication Costs

PAKE. Two users want to prove to each other they possess the same password. In this case $W_i = \mathsf{priv}'_j = \mathsf{priv}_i = \mathsf{priv}_j = \mathsf{priv}'_i = W_j$. So P_i will commit to his password, and thus a unique DLCSCom commitment for W_i, priv_i and priv'_i. P_j can use a simple Linear commitment. They then send projection keys for equality tests: 13 group elements and 2 scalars for Com_i and 5 group elements for Com_j, plus VK_i and σ_i. This leads to 18 group elements and two scalars our PAKE scheme. The DDH-based variant would use 11 group elements and 2 scalars only in total, which is far more efficient than existing solutions, and namely [ACP09] that uses a bit-per-bit commitment to provide equivocability.

Verifier-based PAKE. As explained earlier, we do a PAKE with the common password $(g^{\mathsf{pw}}, h^{\mathsf{pw}})$, where h has been chosen by the server: the commitment Com_i needs 21 group elements plus 2 scalars, and 4 additional group elements to check it; the commitment Com_j needs 6 group elements, and 4 additional elements to check it. Because of the ephemeral h, one has to send in total 35 group elements and 2 scalars, plus the one-time signatures. The DDH-based variant would use 25 group elements and 2 scalars only in total.

Secret Handshake. The users want to check their partner possesses a valid signature on their public identity or pseudonym (in pub) under some valid but private verification key (affiliation-hiding). More precisely, P_i wants to prove he possesses a valid signature σ on the public message m (his identity or a pseudonym) under a private verification key vk: we thus have m in the pub part, $\mathsf{priv}_i = \mathsf{vk}$ and $W = \sigma$. This is the same for P_j. Using Waters signature, $\sigma = (\sigma_1, \sigma_2)$, where σ_1 only has to be encrypted, because σ_2 does not contain any information, it can thus be sent in clear. In addition, as noticed from the security proof, σ_2 does not need to be encrypted in an $\mathsf{IND-PD-CCA}$ manner, but with a simple $\mathsf{IND-CPA}$ encryption scheme in the third round. To achieve unlinkability, one can rerandomize this signature σ to make the σ_2 values different and independent each time.

As a consequence, the committed values are: vk that can be any group element, since with the master secret key s such that $h = g^s$ for the global parameters of the Waters signature (see the Appendix A.3) one can derive the signing key associated to any verification key, and thus generate a valid word in the language; and σ_1 in $\mathsf{IND-CPA}$ only. One additionally sends σ_2 in clear, and so 14 group elements plus 2 scalars for Com_i, and 7 group elements for Com_j. The languages to be verified are $\mathsf{priv}_i = \mathsf{priv}'_i$, on the committed $\mathsf{priv}_i = \mathsf{vk}_i$ with the expected $\mathsf{priv}'_i = \mathsf{vk}'_i$, and the Linear Pairing Product Equation for the committed signature σ_i, but under the expected vk'_i: 5 group elements for the projection keys in both directions: 31 group elements plus 2 scalars are sent in total.