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 [5] and followed by many proposals based on different

* ENS, CNRS & INRIA – UMR 8548.

K. Kurosawa and G. Hanaoka (Eds.): PKC 2013, LNCS 7778, pp. 272–291, 2013.

cryptographic assumptions (see [1, 8] 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 [3] (see also [2,19]). 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 [8]. 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 [8] but the two notions are very similar. In particular, the new primitive enables privacy-preserving 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 [8] 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 [17] in 2003. This approach was applied to the UC framework by Canetti, Halevi, Katz, Lindell, and MacKenzie [11], and improved by Abdalla, Chevalier and Pointcheval [1]. It makes use of the *smooth projective hash functions* (SPHF), introduced by Cramer and Shoup [14]. 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 [1] 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 [20] 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 [11], using the equivocability of the commitment.

Language Definition. In [1], 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 full version [6]: 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 the four following 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 [10, 11] for details. More precisely, we will work in the UC framework with joint state proposed by Canetti and Rabin [12] (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 [4], 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 full version [6]). We will use the Linear Cramer-Shoup encryption scheme [13,21]. We will then patch it, using a technique inspired from [20], 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 [14] in order to build a chosen-ciphertext secure encryption scheme. They have thereafter been extended [1,7,17] 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: we will focus on the DLin-based primitives, and thus the Linear Cramer-Shoup scheme (see the full version [6]), 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 [20]. See the full version [6] 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 denoted by $(g_1, g_2, g_3) \xleftarrow{\$} \mathbb{G}^3$, 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.

Double Linear Cramer-Shoup Commitment Schemes. To make it equivocable, we double the commitment process, in two steps. The CRS additionally contains a scalar $\aleph \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, 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 full version [6] is

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

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 $\mathsf{DLCSCom}$ denotes the use of DLCS with the encryption key ek. The usual commit/decommit processes are described in the full version [6]. On Figure 1, one can find the $\mathsf{DLCSCom}'$ 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 $\mathsf{DLCSCom}'$ scheme is not a real commitment scheme (not formally extractable/binding): in $\mathsf{DLCSCom}'$, 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 full version [6]). We will denote them n-$\mathsf{DLCSCom}'$ or n-$\mathsf{DLCSCom}$ 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 $\varepsilon_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$ 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

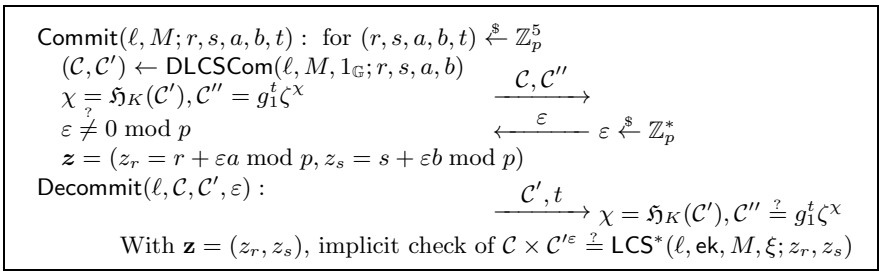

Fig. 1. DLCSCom′ Commitment Scheme for SPHF

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 [1], 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 [18] 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 full version [6].

4.1 Commitments of Signatures

Let us consider the Waters signature [22] in a symmetric bilinear group, 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 [7], 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 [8], since one can also commit scalars bit-by-bit. In the full version [6], 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 $\mathcal{A}_i = g^{\mathfrak{z}_i}$ 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 [11]) 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

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 if there is a record $(P_j, P_i, W_j, L_j, L'_i, \mathsf{initiator})$, then record the tuple $(P_j, P_i, W_j, L_j, L'_i, \mathsf{initiator}, W_i, L_i, L'_j, \mathsf{receiver})$ and send the answer $(\mathsf{NewSession}; \mathsf{sid}, P_i, P_j, \mathsf{pub}, \mathsf{receiver})$ 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.

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

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 [11], the adversary was given access to a $\mathsf{TestPwd}$-query, which modeled the on-line dictionary attack. But it is known since [4] that it is equivalent to use the split functionality model [4], 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

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.

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

(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 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 [1]. The basic idea is to make the player commit to their private information (for the expected 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 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 [4]). 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 full version [6], and is in the same vein as the one in [1,11]. 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

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 ℓ_i the label $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_i, \mathsf{VK}_j)$ and by ℓ_j the label $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, \mathsf{pub}, \mathsf{VK}_j, \mathsf{VK}_i)$, 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:

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

(*) 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.

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

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 [1]). 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, 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

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 $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\varepsilon}$ 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 $\mathsf{Com}_i = \mathcal{C}_i \times \mathcal{C}_i'^{\varepsilon}$. P_i computes K_i and P_j computes K_j as follows:

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

Fig. 5. Password-based Authenticated Key Exchange

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 [8], 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 [16] 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.* [9]), where our PAKE requires less than 20 group elements. Anyway they say, their PAKE scheme is less efficient than [11], 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 [11] 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 [2]): 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 [19] but in a weaker security model). Finally, our scheme supports revocation and can handle roles as in [2].

Standard secret handshakes, like [2], 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 [15] 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 [7], it is also possible to design an *unlinkable* Secret Handshakes protocol [19] 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

1. Abdalla, M., Chevalier, C., Pointcheval, D.: Smooth Projective Hashing for Conditionally Extractable Commitments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 671–689. Springer, Heidelberg (2009)
2. Ateniese, G., Kirsch, J., Blanton, M.: Secret handshakes with dynamic and fuzzy matching. In: NDSS 2007. The Internet Society (February/March 2007)
3. Balfanz, D., Durfee, G., Shankar, N., Smetters, D.K., Staddon, J., Wong, H.-C.: Secret handshakes from pairing-based key agreements. In: IEEE Symposium on Security and Privacy, pp. 180–196. IEEE Computer Society (2003)
4. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure Computation Without Authentication. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 361–377. Springer, Heidelberg (2005)
5. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy, pp. 72–84. IEEE Computer Society Press (May 1992)
6. Ben Hamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: Efficient UC-Secure Authenticated Key-Exchange for Algebraic Languages. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 272–291. Springer, Heidelberg (2013), Full version available from the web page of the authors or from http://eprint.iacr.org/2012/284

7. Blazy, O., Pointcheval, D., Vergnaud, D.: Round-Optimal Privacy-Preserving Protocols with Smooth Projective Hash Functions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 94–111. Springer, Heidelberg (2012)
8. Camenisch, J., Casati, N., Gross, T., Shoup, V.: Credential Authenticated Identification and Key Exchange. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 255–276. Springer, Heidelberg (2010)
9. Camenisch, J., Krenn, S., Shoup, V.: A Framework for Practical Universally Composable Zero-Knowledge Protocols. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 449–467. Springer, Heidelberg (2011)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (October 2001)
11. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally Composable Password-Based Key Exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005)
12. Canetti, R., Rabin, T.: Universal Composition with Joint State. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
13. Cramer, R., Kiltz, E., Padró, C.: A Note on Secure Computation of the Moore-Penrose Pseudoinverse and Its Application to Secure Linear Algebra. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 613–630. Springer, Heidelberg (2007)
14. Cramer, R., Shoup, V.: Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002)
15. Ducas, L.: Anonymity from Asymmetry: New Constructions for Anonymous HIBE. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 148–164. Springer, Heidelberg (2010)
16. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. Journal of Cryptology 19(2), 169–209 (2006)
17. Gennaro, R., Lindell, Y.: A Framework for Password-Based Authenticated Key Exchange. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 524–543. Springer, Heidelberg (2003)
18. Groth, J., Sahai, A.: Efficient Non-interactive Proof Systems for Bilinear Groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
19. Jarecki, S., Liu, X.: Private Mutual Authentication and Conditional Oblivious Transfer. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 90–107. Springer, Heidelberg (2009)
20. Lindell, Y.: Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 446–466. Springer, Heidelberg (2011)
21. Shacham, H.: A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074 (2007)
22. Waters, B.: Efficient Identity-Based Encryption Without Random Oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)