Almost Optimal Oblivious Transfer from QA-NIZK

Céline Chevalier, Paul Germouty, Olivier Blazy

Almost Optimal Oblivious Transfer from QA-NIZK

Olivier Blazy[1], Céline Chevalier[2], and Paul Germouty[1]

[1] Université de Limoges, XLim, France
olivier.blazy@unilim.fr, paul.germouty@unilim.fr
[2] CRED, Université Panthéon-Assas Paris II
celine.chevalier@u-paris2.fr

Abstract. We show how to build a UC-Secure Oblivious Transfer in the presence of Adaptive Corruptions from Quasi-Adaptive Non-Interactive Zero-Knowledge proofs. Our result is based on the work of Jutla and Roy at Asiacrypt 2015, where the authors proposed a constant-size very efficient PAKE scheme. As a stepping stone, we first show how a two-flow PAKE scheme can be generically transformed in an optimized way, in order to achieve an efficient three-flow Oblivious-Transfer scheme. We then compare our generic transformations to existing OT constructions and see that we manage to gain at least a factor 2 to the best known constructions. To the best of our knowledge, our scheme is the first UC-secure Oblivious Transfer with a constant size flow from the receiver, and nearly optimal size for the server.

Keywords. OT, UC, QA-NIZK, Pairing-based Cryptography.

1 Introduction

Sharing data securely and efficiently is at the core of modern cryptography. The concept of Oblivious Transfer was introduced in 1981 by Rabin [Rab81]. It allows to retrieve information from a database without privacy risk. Such protocols allow a receiver to ask and have access to a single line of a database. This exchange is oblivious in two ways. On the one hand, it allows to preserve the privacy of the receiver by preventing the sender from learning which part of the database has been asked. On the other hand, it allows the receiver to recover only one line, the one he previously asked for. The receiver is said to be oblivious to the values he is not allowed to get and the sender is oblivious to the information received. Several instantiations and optimizations of such protocols have appeared in the literature, like [NP01, CLOS02]. More recently, new instantiations have been proposed, trying to reach round-optimality [HK07], or low communication costs [PVW08]. Recent schemes like [ABB+13,BC15,BC16] manage to achieve round-optimality while maintaining a small communication cost. Choi *et al.* [CKWZ13] also propose a generic method and an efficient instantiation secure against adaptive corruptions in the CRS model with erasures, but it is only 1-out-of-2 and it does not scale to 1-out-of-n OT, for $n > 2$.

In *Password-Authenticated Key Exchange (*PAKE*)* protocols, proposed in 1992 by Bellovin and Merritt [BM92], authentication is done using a simple password, possibly drawn from a small space subject to exhaustive search. Since then, many schemes have been proposed and studied in various models, and ultimately proven in the UC framework.

Smooth Projective Hash Functions [CS02,GL03] have found several applications in the context of UC Secure protocols [ACP09,KV11,BBC+13], and also more particularly in the context of Password Authenticated Key Exchange and Oblivious-Transfer Proposals [ABB+13, BC15]. These functions are defined such that their output can be computed in two different ways if the input belongs to a particular subset (the *language*), either using a private hashing key or a public projection key along with a private witness ensuring that the input belongs to the language. The value obtained is indistinguishable from random when the input does not belong to the language (this is called the *smoothness*) and in case the input does belong to the language but no witness is known (this is called the *pseudo-randomness*).

[GS08] proposed a technique to achieve *Non-Interactive Zero-Knowledge Proofs of Knowledge*. Since then, a tremendous amount of work has been dedicated to improving those proofs [GSW10, BFI+10, EG14]. And more recently, a trend studied Quasi-Adaptive NIZK allowing more efficient protocols [JR13, LPJY14, JR14,

ABP15, KW15, JR15]. In this last paper, the authors proposed one of the first PAKE UC-Secure even in the presence of adaptive corruptions, that manages to have flows independent of the password length.

Both Smooth Projective Hash Functions and Quasi-Adaptive Non-Interactive Zero-Knowledge allow to produce very efficient protocols, with a very similar structure, and thus security arguments.

Contributions: Our paper proposes a generic transformation from Password-Authenticated Key Exchange to Oblivious Transfer. As the reverse transformation was already studied in [CDVW12], this allows to study one protocol for the other. Our framework allows to transform a UC-secure PAKE into a UC-secure OT with the same level of security (for instance resistance to adaptive corruptions).

We choose to focus on the transformation of a 2-round PAKE to a 3-round OT since this kind of PAKE is the most commonly encountered and the most efficient one. Furthermore, this allows us to give a generic optimization of our transformation in this specific case, exploiting the fact that the server does not need to hide his "password", since his password is a (public) number of line in the database. Note that our generic transformation could allow to transform an m-round PAKE into an $(m+1)$-round OT, but the optimization would then have to be tailored to the considered protocol.

After showing an application of our technique on an already existing PAKE scheme from [ABB+13], which allows us to immediately recover the associated OT scheme given in their article, we show that our transformation also applies to the PAKE scheme from [JR15], allowing us to give a nearly optimal OT scheme.

Note that our technique works with every possible PAKE scheme, be they elliptic-curve-based or not. Furthermore, also note that it also allows to transform a BPR-secure PAKE [BPR00] into an OT secure in a game-based security model. In order to illustrate these last two points, we apply our framework to a (non-UC) secure lattice-based PAKE scheme proposed by [KV09], giving us a (non-UC) secure lattice-based OT scheme (see Appendix C).

Comparison with other existing OT schemes: The table given in Figure 1 shows the costs of various known OT schemes based on elliptic curves.

While Barreto-Naehrig curves [BN06] are considered less and less secure for efficient parameters, we consider for the asymptotic scaling that elements in $\mathbb{G}_2$ are smaller than those in $\mathbb{G}_1$, hence, we switched some elements to $\mathbb{G}_2$ for big enough values of n.

Thanks to the use of a QA-NIZK technique, we manage to get rid of the extra logarithm overhead. And we end up being approximately four times more efficient than existing solutions without this overhead.

Communication cost comparisons of various Elliptic Curve based OT schemes

Paper	Assumption	# Group elements	# Rounds
Static Security (1-out-of-2)			
[PVW08] + [GWZ09]	SXDH	51	8
[CKWZ13]	SXDH	$26 + 7\ \mathbb{Z}_p$	4
Adaptive Security (1-out-of-2)			
[ABB+13]	SXDH	$12\ \mathbb{G}_1 + 1\ \mathbb{G}_2 + 2\mathbb{Z}_p$	3
[BC15][1]	DDH	$15\ \mathbb{G} + 2\ \mathbb{Z}_p$	3
[BC16]	SXDH	$12\ \mathbb{G}_1 + 4\ \mathbb{G}_2 + 2\ \mathbb{Z}_p$	3
This paper	SXDH	$6\ \mathbb{G}_1 + 2\ \mathbb{G}_2 + 2\ \mathbb{Z}_p$	3
Adaptive Security (1-out-of-n)			
[ABB+13]	SXDH	$\log n\ \mathbb{G}_1 + (n + 8\log n)\ \mathbb{G}_2 + n\ \mathbb{Z}_p$	3
[BC15][1]	DDH	$(n + 9\log n + 4)\ \mathbb{G} + 2n\ \mathbb{Z}_p$	3
[BC16]	SXDH	$4\ \mathbb{G}_1 + (4n + 4)\ \mathbb{G}_2 + n\ \mathbb{Z}_p$	3
This paper	SXDH	$4\ \mathbb{G}_1 + (n + 2)\ \mathbb{G}_2 + n\ \mathbb{Z}_p$	3

Fig. 1. Comparison to existing Oblivious Transfer

Organization of the paper: To present our fast Oblivious Transfer scheme in 3 flows, we first recall basic definitions and security notions needed for the understanding of the paper. Then, we propose our generic transformation from a 2-round PAKE scheme to a 3-round OT scheme and show how it can be generically improved by some optimizations. We then apply our framework to the PAKE from [ABB+13], and show it directly leads to their existing OT. We then recall the PAKE scheme described in [JR15], and show how to construct from it our new nearly optimal OT by applying our framework.

Finally, we sketch in Appendix C a brief example to show how our framework behave on a non-UC lattice-based PAKE and how we get a game-based OT scheme from the BPR-secure PAKE scheme presented in [KV09].

2 Preliminaries

2.1 Definitions and Notations

In the following, in asymmetric protocols the sender is going to be called $\mathcal{S}$ while the receiver will be noted $\mathcal{R}$. For symmetrical ones like PAKE, we will rather use P_i and P_j. We will note the security parameter $\mathfrak{K}$, and n the size of the database (number of lines) for Oblivious Transfer schemes.

We are going to work in an asymmetric bilinear setting denoted as $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of order p for a $\mathfrak{K}$-bit prime p, g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerated) bilinear map. Define $g_T := e(g_1, g_2)$, which is a generator in $\mathbb{G}_T$.

Definition 1 (DDH assumption). *Assuming a multiplicative group $\mathbb{G}$ of prime order p and generator g, we say that the* DDH *assumption is hard in $\mathbb{G}$ if, given a tuple $(g^x, g^y, g^z) \in \mathbb{G}^3$, it is hard to decide whether $z = xy$ or not.*

Definition 2 (Symmetric External Diffie Hellman (SXDH [ACHdM05]) assumption). *This variant of* DDH*, used mostly in bilinear groups $(\mathbb{G}_1, \mathbb{G}_2)$ in which no computationally efficient homomorphism exists from $\mathbb{G}_2$ to $\mathbb{G}_1$ or $\mathbb{G}_1$ to $\mathbb{G}_2$, states that the* DDH *assumption is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

Definition 3 ((Labelled) Quasi Adaptive Non-Interactive Zero Knowledge Argument). *Assuming a randomness distribution $\mathcal{D}_{\mathsf{param}}$ (for a public set of parameters* param*) and a set of languages $\{\mathfrak{L}_\rho\}_\rho$ parameterized by a randomness $\rho \leftarrow \mathcal{D}_{\mathsf{param}}$ and associated with a relation $\mathcal{R}_\rho$ (meaning that a word x belongs to $\mathfrak{L}_\rho$ if and only if there exists a witness w such that $\mathcal{R}_\rho(x, w)$ holds), a* QA-NIZK *argument Π for this set of languages $\{\mathfrak{L}_\rho\}_\rho$ consists of five probabilistic polynomial time (PPT) algorithms $\Pi = (\mathsf{Gen}_{\mathsf{param}}, \mathsf{Gen}_{\mathsf{crs}}, \mathsf{Prove}, \mathsf{Sim}, \mathsf{Ver})$ defined as follows:*

- $\mathsf{Gen}_{\mathsf{param}}(\mathfrak{K})$ *returns the public parameters* param*.*
- $\mathsf{Gen}_{\mathsf{crs}}(\mathsf{param}, \rho)$ *returns a common reference string* crs *and a trapdoor* tk*. We assume that* crs *contain the parameters* param*, a description of the language $\mathfrak{L}_\rho$ and a description of a label space $\mathcal{L}$.*
- Prove *takes as input* crs*, a label $\ell \in \mathcal{L}$, a word x of the language $\mathfrak{L}_\rho$ and a witness w corresponding to this word. If $(x, w) \notin \mathcal{R}_\rho$, it outputs failure. Otherwise, it outputs a proof π.*
- Ver *takes as input* crs*, a label ℓ, a word x and a proof π. It outputs a bit b (either 1 if the proof is correct, or 0 otherwise).*
- Sim *takes as input* crs*, a trapdoor* tk*, a label $\ell \in \mathcal{L}$ and a word x. It outputs a proof π for x (not necessarily in $\mathfrak{L}_\rho$) with respect to the label ℓ.*

These algorithms must satisfy the following properties

[1] It should be noted that the [BC15] OT does not rely on pairings. Classical implementations suggest that the elements in one of the groups in our scheme will be at least twice as big as those from the non-pairing curve, which would give roughly the same efficiency between both schemes in the 1-out-of-2 case, with an edge for the [BC15] construction in term of computational requirements. However, the scaling in their paper is then worse when increasing the number of lines (see the 1-out-of-n case).

Perfect Completeness: *for all PPT adversary $\mathcal{A}$, all security parameter $\mathfrak{K}$, all public parameters* $\mathsf{param} \leftarrow \mathsf{Gen}_{\mathsf{param}}(\mathfrak{K})$*, all randomness* $\rho \leftarrow \mathcal{D}_{\mathsf{param}}$*, all label* $\ell \in \mathcal{L}$ *and all* $(x, w) \in \mathcal{R}_\rho$*, the following holds:*

$$\Pr[(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\mathsf{param}, \rho); \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \ell, x, w) : \mathsf{Ver}(\mathsf{crs}, \ell, x, \pi) = 1] = 1$$

Computational Adaptive Soundness: *For a given security parameter $\mathfrak{K}$, a scheme achieves computational adaptive soundness if for all PPT adversary $\mathcal{A}$, the probability*

$$\Pr[\mathsf{param} \leftarrow \mathsf{Gen}_{\mathsf{param}}(\mathfrak{K}); \rho \leftarrow \mathcal{D}_{\mathsf{param}}; (\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\mathsf{param}, \rho);$$
$$(\ell, x, \pi) \leftarrow \mathcal{A}(\mathsf{param}, \mathsf{crs}, \rho) : x \notin \mathfrak{L}_\rho \wedge \mathsf{Ver}(\mathsf{crs}, \ell, x, \pi) = 1]$$

is negligible.

Perfect Zero-Knowledge: *for all $\mathfrak{K}$, all* $\mathsf{param} \leftarrow \mathsf{Gen}_{\mathsf{param}}(\mathfrak{K})$*, all* $\rho \leftarrow \mathcal{D}_{\mathsf{param}}$*, all* $(\mathsf{crs}, \mathsf{tk}) \leftarrow \mathsf{Gen}_{\mathsf{crs}}(\mathsf{param}, \rho)$*, all label $\ell \in \mathcal{L}$ and all $(x, w) \in \mathcal{R}_\rho$, the distributions* $\mathsf{Prove}(\mathsf{crs}, \ell, x, w)$ *and* $\mathsf{Sim}(\mathsf{crs}, \mathsf{tk}, \ell, x)$ *are the same.*

2.2 Security Models

UC Framework. The goal of the UC framework [Can01] is to ensure that UC-secure protocols will continue to behave in the ideal way even if executed in a concurrent way in arbitrary environments. It is a simulation-based model, relying on the indistinguishability between the real world and the ideal world. In the ideal world, the security is provided by an ideal functionality $\mathcal{F}$, capturing all the properties required for the protocol and all the means of the adversary. In order to prove that a protocol Π emulates $\mathcal{F}$, one has to construct, for any polynomial adversary $\mathscr{A}$ (which controls the communication between the players), a simulator $\mathscr{S}$ such that no polynomial environment $\mathcal{Z}$ can distinguish between the real world (with the real players interacting with themselves and $\mathscr{A}$ and executing the protocol π) and the ideal world (with dummy players interacting with $\mathscr{S}$ and $\mathcal{F}$) with a significant advantage. The adversary can be either *adaptive*, *i.e.* allowed to corrupt users whenever it likes to, or *static*, *i.e.* required to choose which users to corrupt prior to the execution of the session sid of the protocol. After corrupting a player, $\mathscr{A}$ has complete access to the internal state and private values of the player, takes its entire control, and plays on its behalf.

Simple UC Framework. Canetti, Cohen and Lindell formalized a simpler variant in [CCL15], that we use here. This simplifies the description of the functionalities for the following reasons (in a nutshell): All channels are automatically assumed to be authenticated (as if we worked in the $\mathcal{F}_{\text{AUTH}}$-hybrid model); There is no need for *public delayed outputs* (waiting for the adversary before delivering a message to a party), neither for an explicit description of the corruptions. We refer the interested reader to [CCL15] for details.

Password Authenticated Key Exchange. Before the UC framework, the classical security model for PAKE protocols was the so-called BPR model [BPR00]. The first ideal functionality for PAKE protocols in the UC framework [Can01, CK02] was proposed by Canetti *et al.* [CHK+05], who showed how a simple variant of the Gennaro-Lindell methodology [GL03] could lead to a secure protocol.

We present the PAKE ideal functionality $\mathcal{F}_{pwKE}$ on Figure 2. It was described in [CHK+05]. The main idea behind this functionality is as follows: If neither party is corrupted and the adversary does not attempt any password guess, then the two players both end up with either the same uniformly-distributed session key if the passwords are the same, or uniformly-distributed independent session keys if the passwords are distinct. In addition, the adversary does not know whether this is a success or not. However, if one party is corrupted, or if the adversary successfully guessed the player's password (the session is then marked as `compromised`), the adversary is granted the right to fully determine its session key. There is in fact nothing lost by allowing it to determine the key. In case of a wrong guess (the session is then marked as `interrupted`), the two players are given independently-chosen random keys. A session that is nor `compromised` nor `interrupted` is called `fresh`, which is its initial status.

Finally notice that the functionality is not in charge of providing the password(s) to the participants. The passwords are chosen by the environment which then hands them to the parties as inputs. This guarantees

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

- **Upon receiving a query (`NewSession`, sid, ssid, P_i, P_j, pw) from party P_i:**
  Send (`NewSession`, sid, ssid, P_i, P_j) to $\mathcal{S}$. If this is the first `NewSession` query, or if this is the second `NewSession` query and there is a record (sid, ssid, P_j, P_i, pw′), then record (sid, ssid, P_i, P_j, pw) and mark this record `fresh`.
- **Upon receiving a query (`TestPwd`, sid, ssid, P_i, pw′) from the adversary $\mathcal{S}$:**
  If there is a record of the form (P_i, P_j, pw) which is `fresh`, then do: If pw = pw′, mark the record `compromised` and reply to $\mathcal{S}$ with "correct guess". If pw ≠ pw′, mark the record `interrupted` and reply with "wrong guess".
- **Upon receiving a query (`NewKey`, sid, ssid, P_i, sk) from the adversary $\mathcal{S}$:**
  If there is a record of the form (sid, ssid, P_i, P_j, pw), and this is the first `NewKey` query for P_i, then:
  - If this record is `compromised`, or either P_i or P_j is corrupted, then output (sid, ssid, sk) to player P_i.
  - If this record is `fresh`, and there is a record (P_j, P_i, pw′) with pw′ = pw, and a key sk′ was sent to P_j, and (P_j, P_i, pw) was `fresh` at the time, then output (sid, ssid, sk′) to P_i.
  - In any other case, pick a new random key sk′ of length $\mathfrak{K}$ and send (sid, ssid, sk') to P_i.

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

Fig. 2. Ideal Functionality for PAKE $\mathcal{F}_{\text{pwKE}}$

security even in the case where two honest players execute the protocol with two different passwords: This models, for instance, the case where a user mistypes its password. It also implies that the security is preserved for all password distributions (not necessarily the uniform one) and in all situations where the passwords are related passwords and/or used in different protocols. Also note that allowing the environment to choose the passwords guarantees forward secrecy.

In case of corruption, the adversary learns the password of the corrupted player; After the `NewKey`-query, it additionally learns the session key.

Oblivious Transfer. The ideal functionality of an Oblivious Transfer (OT) protocol was given in [Can01, CKWZ13, ABB+13]. We recall it in simple UC in Figure 3 using the functionality introduced in [BCG16]. Note that the BPR model [BPR00] given for PAKE protocols can be adapted to give a game-based security model for OT schemes but this is well beyond the scope of this paper.

The party P_i is the sender $\mathcal{S}$, while the party P_j is the receiver $\mathcal{R}$. The former is provided with a database consisting of a set of n lines $(L_1,\ldots,L_n)$, while the latter is querying a particular line L_s (with $s \in \{1,\ldots,n\}$). Since there is no communication between them (the functionality deals with everything), it automatically ensures the oblivious property on both sides (the sender does not learn which line was queried, while the receiver does not learn any line other than L_s).

The functionality $\mathcal{F}_{(1,n)\text{-OT}}$ is parametrized by a security parameter $\mathfrak{K}$. It interacts with an adversary $\mathcal{S}$ and a set of parties $P_1,\ldots,P_N$ via the following queries:

- **Upon receiving an input (`Send`, sid, ssid, P_i, P_j, $(L_1,\ldots,L_n)$) from party $\mathcal{P}_i$**, with $L_i \in \{0,1\}^{\mathfrak{K}}$ for all i: record the tuple (sid, ssid, P_i, P_j, $(L_1,\ldots,L_n)$) and reveal (`Send`, sid, ssid, P_i, P_j) to the adversary $\mathcal{S}$. Ignore further `Send`-message with the same ssid from P_i.
- **Upon receiving an input (`Receive`, sid, ssid, P_i, P_j, s) from party P_j**: ignore the message if (sid, ssid, P_i, P_j, $(L_1,\ldots,L_n)$) is not recorded. Otherwise, reveal (`Receive`, sid, ssid, P_i, P_j) to the adversary $\mathcal{S}$ and send (`Received`, sid, ssid, P_i, P_j, L_s) to P_j and ignore further `Receive`-message with the same ssid from P_j.

Fig. 3. Ideal Functionality for 1-out-of-n Oblivious Transfer $\mathcal{F}_{(1,n)\text{-OT}}$

3 Generic Construction of an Oblivious Transfer from a UC-Secure PAKE

There is a trend in proven cryptography that consists in finding the link between primitives and show which primitives can be used to build other ones. A well-known example is the transformation of an IBE encryption scheme into a signature scheme, which was proposed by Naor as recalled in [BF01], and its reverse transformation from an affine MAC or signature scheme into an IBE scheme given in [BKP14].

A neat result, presented by Canetti et al. in [CDVW12], shows how a UC-secure Oblivious Transfer scheme can be transformed into a UC-secure PAKE scheme. In this section we are going to go the other way, and show how a UC-secure PAKE can be transformed into a UC-secure Oblivious Transfer scheme. While considering this direction may seem like transforming a very strong primitive into a weaker one, this will allow us to prepare a framework to propose the most efficient Oblivious Transfer scheme to date. This also echoes to the results from [Ngu05] on a simulation-based transform.

3.1 Description of a UC-Secure PAKE Scheme

As explained in the introduction, we are going to consider PAKE protocols in two flows, since there already exists a multitude of PAKE schemes satisfying this requirement, and since minimizing the number of flows is one of the most important issues in modern instantiations. This allows us to generically give an optimization, leading to more efficient protocols.

In order to present the framework or our transformation, we formally split a PAKE scheme into the following four algorithms[2].

- Setup: An algorithm that generates the initial crs.
- $\mathsf{flow}_1(\mathsf{pw}_i; \rho_i, \rho'_i)$: The algorithm run by the user P_i, using some randomness ρ_i and ρ'_i and possessing the password pw_i, to generate the first flow of the protocol. We note ρ_i the part of the randomness involved in hiding the password, and ρ'_i the remainder.
- $\mathsf{flow}_2(\mathsf{flow}_1, \mathsf{pw}_j; \rho_j, \rho'_j)$: The algorithm run by the user P_j, using some randomness ρ_j and possessing the password pw_j, after receiving the flow flow_1, to generate the second flow of the protocol. This algorithm also produces P_j's view of the key: $K_i^{(j)}$.
- $\mathsf{Decap}(\mathsf{flow}_2, \mathsf{pw}_i, f(\rho_i, \rho'_i))$: The algorithm run by P_i when receiving the flow flow_2 from P_j, using his password pw_i and a function $f(\rho_i, \rho'_i)$ of the randomness initially used and the remainder (which function can be anything from the identity to the null function), to recover P_i's view of the key: $K_j^{(i)}$.

3.2 Generic Construction of a UC-Secure OT Scheme

We denote by $\mathsf{DB} = (L_1, \ldots, L_n)$ the database of the server containing n lines, and by s the line requested by the user in an oblivious way. We assume the existence of a CPA-Secure encryption scheme, whose public parameters are going to be included in the public common reference string crs generated by the setup algorithm of the PAKE scheme. Such an encryption scheme $\mathcal{E}$ is described through four algorithms ($\mathsf{Setup}_{\text{cpa}}$, $\mathsf{KeyGen}_{\text{cpa}}$, $\mathsf{Encrypt}_{\text{cpa}}$, $\mathsf{Decrypt}_{\text{cpa}}$):

- $\mathsf{Setup}_{\text{cpa}}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters $\mathsf{param}_{\text{cpa}}$ of the scheme;
- $\mathsf{KeyGen}_{\text{cpa}}(\mathsf{param}_{\text{cpa}})$ outputs a pair of keys: a (public) encryption key pk and a (private) decryption key sk;
- $\mathsf{Encrypt}_{\text{cpa}}(\mathsf{pk}, M; \rho)$ outputs a ciphertext $\boldsymbol{c} = \mathcal{E}(M)$ on the message M, under the encryption key pk, using the randomness ρ;
- $\mathsf{Decrypt}_{\text{cpa}}(\mathsf{sk}, \boldsymbol{c})$ outputs the plaintext M encrypted in the ciphertext $\boldsymbol{c}$, or $\perp$.

We also assume the existence of a Pseudo-Random Generator (PRG) F with input size equal to the plaintext size, and output size equal to the size of the messages in the database.

Using the PAKE scheme, the encryption scheme and the PRG, one can now obtain an OT scheme from a PAKE scheme, as described in Figure 4. The idea of the protocol is as follows:

[2] Since it was shown in [CHK+05] that UC-secure PAKE is impossible in the plain model, we focus on the CRS model for the ease of readability.

– First, a setup phase enables to generate the CRS, both for the PAKE and encryption schemes.
– The pre-flow is a technical requirement for the UC proof.
– When querying for line s, the receiver proceeds in three steps. The first one consists in generating a masking value R (technical requirement for the UC proof), the second one consists in running the first flow of the PAKE scheme for password s to generate the first flow denoted[3] as flow_0, and the third one consists in erasing anything but the needed values, and sending flow_0 to the sender.
– When answering to this query, the sender also proceeds in three steps. The first one consists in recovering the value R and the second one consists in running, for each line $k \in \{1, \ldots, n\}$, the second flow of the PAKE scheme to generate the second flow denoted as flow_k (using k as the password) as well as $\mathcal{S}$'s view of the session key, denoted as $(K_k)_{\mathcal{R}}^{(\mathcal{S})}$. Finally, in the third step, the server computes a xor Q_k of the line L_k and this session key and the value R and sends the set (flow_k, Q_k) back to $\mathcal{R}$.
– When receiving this set of values, the receiver uses flow_s to run the decapsulation algorithm of the PAKE scheme (with password s) to recover his view of the session key, denoted as $(K_s)_{\mathcal{S}}^{(\mathcal{R})}$. He finally uses R, Q_s and $(K_s)_{\mathcal{S}}^{(\mathcal{R})}$ to recover the expected line L_s.

The correctness easily follows from the correctness of the PAKE protocol, since the views of $\mathcal{R}$ and $\mathcal{S}$ of the session key for the PAKE scheme executed for password s are the same. The scheme is oblivious with respect to $\mathcal{S}$ since the first flow of the PAKE scheme executed by $\mathcal{R}$ does not reveal any information on s. And it is oblivious with respect to $\mathcal{R}$ since the correctness of the PAKE scheme gives him no information on the session keys $(K_k)_{\mathcal{R}}^{(\mathcal{S})}$ for $k \neq s$. The security is stated in Theorem 4.

CRS generation:
$\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^{\lambda}), \mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^{\lambda})$.

Pre-Flow:

1. $\mathcal{S}$ generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$, stores sk and completely erases the random coins used by $\mathsf{KeyGen}_{\mathrm{cpa}}$.
2. $\mathcal{S}$ publishes pk.

Index query on s:

1. $\mathcal{R}$ picks a random J, computes $R \leftarrow F(J)$ and sets $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, J)$.
2. $\mathcal{R}$ picks a random (ρ, ρ'), and runs $\mathsf{flow}_1(s; \rho, \rho')$ to obtain flow_0.
3. $\mathcal{R}$ stores $f(\rho, \rho')$ needed for the Decap algorithm and R and completely erases the rest, including the random coins used by $\mathsf{Encrypt}_{\mathrm{cpa}}$ and sends (c, flow_0) to $\mathcal{S}$.

Database answer:

1. $\mathcal{S}$ decrypts $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and computes $R \leftarrow F(J)$.
2. For each line k:
   – $\mathcal{S}$ picks a random (ρ_k, ρ'_k), runs $\mathsf{flow}_2(\mathsf{flow}_0, k; \rho_k, \rho'_k)$ to generate flow_k and $(K_k)_{\mathcal{R}}^{(\mathcal{S})}$.
   – $\mathcal{S}$ then computes $Q_k = L_k \oplus (K_k)_{\mathcal{R}}^{(\mathcal{S})} \oplus R$.
3. $\mathcal{S}$ erases everything except $(\mathsf{flow}_k, Q_k)_{k \in [\![1,n]\!]}$ and sends it to $\mathcal{R}$.

Data recovery:
$\mathcal{R}$ then using $f(\rho, \perp)$ computes $(K_s)_{\mathcal{S}}^{(\mathcal{R})} = \mathsf{Decap}(\mathsf{flow}_s, s, f(\rho, \rho'))$, sets $L_s = Q_s \oplus (K_s)_{\mathcal{S}}^{(\mathcal{R})} \oplus R$, and erases everything else.

Fig. 4. UC-Secure 1-out-of-n OT from a UC-Secure PAKE

Theorem 4. *Under the* UC-*security*[4] *of the* PAKE *protocol, the existence of a Pseudo-Random Generator (*PRG*) F with input size equal to the plaintext size, and output size equal to the size of the messages in*

[3] Note that we now denote as flow_0 (and not flow_1) the flow generated by the PAKE algorithm flow_1, in order to avoid the confusion with the flow_1 message generated by the sender (for line 1) during the database answer phase.

[4] Note that the transformation also allows to obtain an OT scheme from a BPR-secure PAKE scheme in a game-based security model, but since this is of less interest, we do not formally prove it due to lack of space.

the database, and the IND-CPA *security of the encryption scheme, the transformation presented in Figure 4 achieves a* UC*-secure 1-out-of-n Oblivious Transfer scheme, with the same handling of corruptions as in the* PAKE *protocol.*

The high-level idea of the proof is quite simple. Recall that in a UC proof, one has to simulate one of the two players in front of a corrupted player. The first flow allows the simulated server to extract the requested line. One will then be able to send random noise on all the lines besides the correct one. The PAKE functionality ensures that this is possible. Then again for honest users, the simulator will replace the remaining line by a random noise. In case of corruptions of the server, one relies on the security of the CPA encryption, to ensure that everything was done honestly (the adversary is not able to test the validity of the retrieved R, and so for the correct line as $L \oplus K \oplus R$ where L and K are the honest values, and R is simply set as $Q \ominus (L \oplus K)$). In case of corruption of the user, we rely on the PAKE handling of user corruption to adapt the memory.

A more detailed proof can be found in Appendix B .

3.3 Generic Optimization

It should be noted that in the previous transformation we never used the fact that the first user in the PAKE scheme (here, the receiver) does not learn the password from the second (here, the sender). This is a property required by PAKE but not useful in the transformation (since the sender executes n parallel PAKE schemes, using the (public) number k of the lines as the password). This argument is the key of the optimization we now propose in Figure 5. The difference is that in the index query, the receiver does not pick the second randomness ρ' and in the database answer, the randomness ρ_k is not used anymore. The proof remains the same as the previous one.

CRS generation:
$\mathsf{crs} \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}}), \mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^{\mathfrak{K}})$.

Pre-Flow:

1. $\mathcal{S}$ generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$, stores sk and completely erases the random coins used by $\mathsf{KeyGen}_{\mathrm{cpa}}$.
2. $\mathcal{S}$ publishes pk.

Index query on s:

1. $\mathcal{R}$ picks a random J, computes $R \leftarrow F(J)$ and sets $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, J)$.
2. $\mathcal{R}$ picks a random ρ, and runs $\mathsf{flow}_1(s; \rho, \perp)$ to obtain flow_0.
3. $\mathcal{R}$ stores $f(\rho, \perp)$ needed for the Decap algorithm and R and completely erases the rest, including the random coins used by $\mathsf{Encrypt}_{\mathrm{cpa}}$ and sends (c, flow_0) to $\mathcal{S}$.

Database answer:

1. $\mathcal{S}$ decrypts $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and computes $R \leftarrow F(J)$.
2. For each line k:
   - $\mathcal{S}$ picks a random ρ'_k, and runs $\mathsf{flow}_2(\mathsf{flow}_0, k; \perp, \rho'_k)$ to generate flow_k and $(K_k)_{\mathcal{R}}^{(\mathcal{S})}$.
   - $\mathcal{S}$ then computes $Q_k = L_k \oplus (K_k)_{\mathcal{R}}^{(\mathcal{S})} \oplus R$.
3. $\mathcal{S}$ erases everything except $(\mathsf{flow}_k, Q_k)_{k \in [\![1,n]\!]}$ and sends it to $\mathcal{R}$.

Data recovery: $\mathcal{R}$ then using $f(\rho, \perp)$ computes $(K_s)_{\mathcal{S}}^{(\mathcal{R})} = \mathsf{Decap}(\mathsf{flow}_s, s, f(\rho, \perp))$, sets $L_s = Q_s \oplus (K_s)_{\mathcal{S}}^{(\mathcal{R})} \oplus R$, and erases everything else.

Fig. 5. Optimized UC-Secure 1-out-of-n OT from a UC-Secure PAKE

4 Warm-up: Applying the framework on [ABB+13]

4.1 Notations

Since in [ABB+13], the authors proposed both a UC-secure PAKE and a UC-secure Oblivious Transfer constructions, this section serves as an example of application of our framework. More precisely, we apply here our (optimized) transformation to their UC-secure PAKE and show that we obtain exactly their UC-secure Oblivious Transfer.

There protocols make use of Commitment schemes and Smooth Projective Hash Functions. Rigorous definitions can be found in Appendix A but we give here the notations used in the following.

A commitment scheme is a two-party primitive (between a committer and a receiver) divided into two phases. In a first *commit* phase, the committer gives the receiver an analogue of a sealed envelope containing a value m, while in the second *opening* phase, the committer reveals m in such a way that the receiver can verify it was indeed m that was contained in the envelope. It is required that a committer cannot change the committed value (*i.e.*, he should not be able to open to a value different from the one he committed to), this is called the *binding* property. It is also required that the receiver cannot learn anything about m before the opening phase, this is called the *hiding* property.

Formally, a *non-interactive labelled commitment scheme* $\mathcal{C}$ is defined by three algorithms:

- $\mathsf{SetupCom}(1^{\mathfrak{K}})$ takes as input the security parameter $\mathfrak{K}$ and outputs the global parameters, passed through the CRS ρ to all other algorithms;
- $\mathsf{Com}^{\ell}(x;\rho)$ takes as input a label ℓ, a message x and a random ρ, and outputs a pair (C,δ), where C is the commitment of x for the label ℓ, and δ is the corresponding opening data (a.k.a. decommitment information).
- $\mathsf{VerCom}^{\ell}(C,x,\delta)$ takes as input a commitment C, a label ℓ, a message x, and the opening data δ and outputs 1 (true) if δ is a valid opening data for C, x and ℓ. It always outputs 0 (false) on $x=\perp$.

Smooth projective hash functions are defined such as their value can be computed in two different ways if the input belongs to a particular subset (the *language*), either using a private hashing key or a public projection key along with a private witness ensuring that the input belongs to the language. The projection key is derived from the hashing key. The hash value obtained is indistinguishable from random in case the input does not belong to the language (property of *smoothness*) and in case the input does belong to the language but no witness is known (property of *pseudo-randomness*).

Formally, a Smooth Projective Hash Function over a language $\mathfrak{L} \subset X$, onto a set $\mathcal{G}$, is defined by five algorithms ($\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$):

- $\mathsf{Setup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathfrak{L}$;
- $\mathsf{HashKG}(\mathfrak{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathfrak{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, derives the projection key hp from the hashing key hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathcal{G}$, thanks to the hashing key hk and W.
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathcal{G}$, thanks to the projection key hp and the witness w that $W \in \mathfrak{L}$.

4.2 Description of Their UC-Secure PAKE Scheme

Description of the scheme: Both parties (denoted as P_b and $P_{\bar{b}}$) want to share a key by using a common password pw. This will thus fix a language $\mathfrak{L}_{\mathsf{pw}}$, which is the set of valid commitments of pw, for a smooth projective hash function. Both flows are done simultaneously. In flow_b, each party P_b will generate a pair of hash keys $(\mathsf{hk}_b, \mathsf{hp}_b)$ and then commit to the password pw_b in C_b. Note that P_b stores the witness δ_b that this value C_b is a valid commitment of pw_b and sends (hp_b, C_b) to the other party.

In the decapsulation phase, each party P_b then receives the commitment and the projection key of the other party and computes two hash values: On the one hand, the projected hash value using his own commitment C_b

along with the witness δ_b and the projected key $\mathsf{hp}_{\bar{b}}$ sent by the other party, and on the other hand, the (regular) hash value using the received commitment $C_{\bar{b}}$ and the secret hash key hk_b. The shared session key is then defined as the product of these two values.

The commitment scheme used has specific properties explained in [ABB+13] well beyond the scope of the paper. It needs to be compatible with SPHF, and both extractable and equivocable.

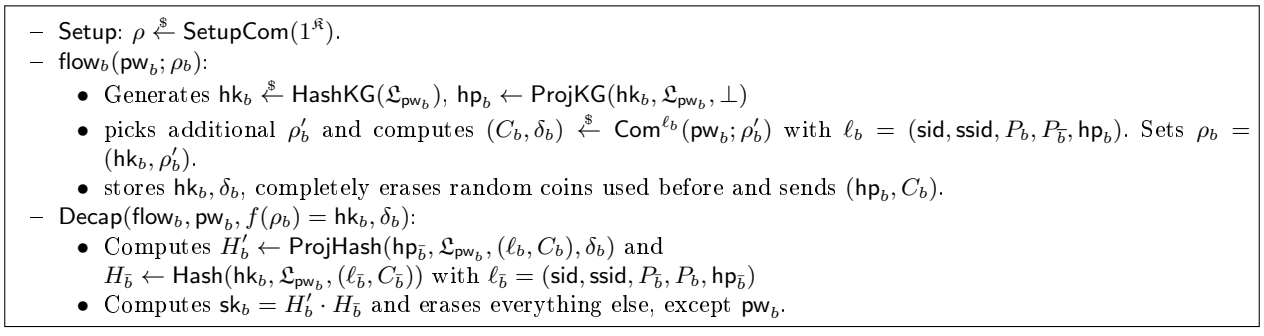

Fig. 6. UC-Secure PAKE from [ABB+13]

Completeness: If the two parties used the same password pw, then at the end of the protocol they will share the same value. Thanks to the property of the SPHF, the hash value and the projected value (on the same commitment C_b) will be equal because the commitment belongs to the good language $\mathfrak{L}_{\mathsf{pw}}$ with the good witness δ_b.

4.3 Applying the Framework to Obtain a UC-Secure OT Scheme

Applying our (optimized) framework, we manage to fall back directly to the Oblivious Transfer that was proposed simultaneously in [ABB+13].

Theorem 5. *As the* PAKE *was UC-Secure in presence of adaptive corruption under* SXDH*, the Oblivious Transfer scheme presented in figure 7 is a secure 1-out-of-n Oblivious Transfer, with the same handling of corruptions as the* PAKE *protocol under* SXDH *and the* IND-CPA *security of the encryption scheme.*

5 New Efficient Oblivious Transfer based on QA-NIZK

5.1 Description of Their UC-Secure PAKE Scheme

We now consider the instantiation from [JR15] in which the UC-PAKE allows for each party to hide his own password. As before, we first recall their protocol and then show how to use it in order to instantiate a new UC-secure Oblivious Transfer. This OT scheme ends up being the most efficient to date, as we show by comparing to the recent [BC16] in Table 8.

Interestingly this protocol does not use Smooth Projective Hash Function, but Quasi Adaptive Non-Interactive Zero Knowledge proofs. It is presented in Figure 9.

As explained in the original paper, the scheme is loosely-based on Quasi-Adaptive NIZK, in the sense that the flows use computation very close to what would be expected. In the figure 9, $\mathcal{H}$ is a family of hash functions.

5.2 Applying the Framework to Obtain a UC-Secure OT Scheme

Using the instantiation from [JR15] described in Figure 9, we apply our framework and directly obtain the protocol described in Figure 10.

The function G is assumed to map line numbers to group elements, and it should be collision-resistant and efficiently computable. One can use for example, the G function from Lindell in [Lin11].

CRS generation:
$\rho \xleftarrow{\$} \mathsf{SetupCom}(1^\mathfrak{K}), \mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^\mathfrak{K})$.

Pre-flow :

1. $\mathcal{S}$ generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$, stores sk and completely erases the random coins used by $\mathsf{KeyGen}_{\mathrm{cpa}}$.
2. $\mathcal{S}$ publishes pk.

Index query on s:

1. $\mathcal{R}$ chooses a random value J, computes $R \leftarrow F(J)$ and encrypts J under pk: $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, J)$
2. $\mathcal{R}$ computes $(C, \delta) \xleftarrow{\$} \mathsf{Com}^\ell(s)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{R}, \mathcal{S})$
3. $\mathcal{R}$ stores δ_s, R, completely erases the random coins used and sends C and c to $\mathcal{S}$

Database answer $(L_1, \ldots, L_n)$:

1. $\mathcal{S}$ decrypts $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and gets $R \leftarrow F(J)$
2. For each line $k = 1, \ldots, n$,
   - $\mathcal{S}$ computes $\mathsf{hk}_k \xleftarrow{\$} \mathsf{HashKG}(\mathfrak{L}_k)$, $\mathsf{hp}_k \leftarrow \mathsf{ProjKG}(\mathsf{hk}_k, \mathfrak{L}_k, (\ell, C))$, $K_k \leftarrow \mathsf{Hash}(\mathsf{hk}_k, \mathfrak{L}_k, (\ell, C))$,
   - $\mathcal{S}$ computes $Q_k \leftarrow R \oplus K_k \oplus L_k$
3. $\mathcal{S}$ erases everything except $(\mathsf{hp}_k, Q_k)_{k=1,\ldots,n}$ and sends it to $\mathcal{R}$.

Data recovery:
Upon receiving $(\mathsf{hp}_k, Q_k)_{k=1,\ldots,n}$, $\mathcal{R}$ computes $K_s \leftarrow \mathsf{ProjHash}(\mathsf{hp}_s, \mathfrak{L}_s, (\ell, C), \delta)$ and gets $L_s \leftarrow R \oplus K_s \oplus Q_s$.
Then $\mathcal{R}$ erases everything except L_s.

Fig. 7. UC-Secure 1-out-of-n OT from an SPHF-Friendly Commitment

Paper	Pre-Flow	Query	Answer
[BC15]	2 $\mathbb{G}$	$9 \log n + 2$ $\mathbb{G}$	n $\mathbb{G} + 2n$ $\mathbb{Z}_p$
[BC16]	2 $\mathbb{G}_2$	4 $\mathbb{G}_1$ + 2 $\mathbb{G}_2$	$4n$ $\mathbb{G}_2 + n$ $\mathbb{Z}_p$
This paper	2 $\mathbb{G}_2$	4 $\mathbb{G}_1$	n $\mathbb{G}_2 + n$ $\mathbb{Z}_p$

Fig. 8. Comparison to previous schemes

CRS generation:
$g_1 \xleftarrow{\$} \mathbb{G}_1, g_2 \xleftarrow{\$} \mathbb{G}_2, a, c, o, d, e, u_1, u_2 \xleftarrow{\$} \mathbb{Z}_q, H \xleftarrow{\$} \mathcal{H}$
Set $A = g_1^a, D = g_1^d, E = g_1^e, U_1 = g_1^{u_1}, U_2 = g_1^{u_2} \in \mathbb{G}_1$
And $B = g_2^c, O = g_2^o, V_1 = g_2^{u_1 c - d - oa}, V_2 = g_2^{u_2 c - e} \in \mathbb{G}_2$
$\mathsf{crs} := (g_1, g_2, A, B, O, F, D, E, U_1, U_2, V_1, V_2, H)$

$\mathsf{flow}_b(\mathsf{pw}; (r_b, s_b))$:

1. P_b computes $R_b = g_1^{r_b}, S_b = \mathsf{pw} \cdot A^{r_b}, T_b = (D \cdot E^{h_b})^{r_b}, \hat{\rho}_b = B^{s_b}, W_b = (U_1 \cdot U_2^{h_b})^{r_b}$, where $h_b = H(\mathbf{sid}, \mathbf{ssid}, P_b, P_{\bar{b}}, R_1, S_1, \hat{\rho}_b)$
2. P_b erases r_b, sends $R_b, S_b, T_b, \hat{\rho}_b$ and keeps W_b.

Key computing:

1. P_b computes $h_{\bar{b}} = H(\mathbf{sid}, \mathbf{ssid}, P_{\bar{b}}, P_b, R_{\bar{b}}, S_{\bar{b}}, \hat{\rho}_{\bar{b}})$
   Checks that all elements are consistent, and computes:
   $\rho = g_2^{s_b}, \theta = O^{s_b}, \gamma = (V_1 \cdot V_2^{h_{\bar{b}}})^{s_b}$
   $\mathsf{sk} = e(T_{\bar{b}}, \rho) \cdot e(S_{\bar{b}}/\mathsf{pw}, \theta) \cdot e(R_{\bar{b}}, \gamma) \cdot e(W_b, \hat{\rho}_{\bar{b}})$

Fig. 9. UC secure PAKE from [JR15]

Theorem 6. *As the* PAKE *was UC-Secure in presence of adaptive corruption under* SXDH*, and ElGamal is* IND-CPA *under* SXDH *too, the Oblivious Transfer scheme presented in Figure 10 is a secure 1-out-of-n Oblivious Transfer, with the same handling of corruptions as the* PAKE *protocol under* SXDH.

CRS generation:
$g_1 \xleftarrow{\$} \mathbb{G}_1, g_2 \xleftarrow{\$} \mathbb{G}_2, a, c, o, d, e, u_1, u_2 \xleftarrow{\$} \mathbb{Z}_p, H \xleftarrow{\$} \mathcal{H}$
Sets $A = g_1^a, D = g_1^d, E = g_1^e, U_1 = g_1^{u_1}, U_2 = g_1^{u_2} \in \mathbb{G}_1$
And $B = g_2^c, O = g_2^o, V_1 = g_2^{u_1 c - d - oa}, V_2 = g_2^{u_2 c - e} \in \mathbb{G}_2$
$\mathsf{crs} := (g_1, g_2, A, B, O, F, D, E, U_1, U_2, V_1, V_2, H)$

Pre-Flow:

1. $\mathcal{S}$ generates $(\mathsf{pk} = f_1 = g_1^\alpha, \mathsf{sk} = \alpha \xleftarrow{\$} \mathbb{Z}_p)$
2. $\mathcal{S}$ erases random coins, keeps sk and sends pk.

Index query on i:

1. $\mathcal{R}$ picks $J \xleftarrow{\$} \mathbb{G}_1, s \xleftarrow{\$} \mathbb{Z}_p$, sets $\mathsf{mask} \leftarrow F(J)$ and $c = (f_1^s J, g_1^s)$
2. $\mathcal{R}$ picks $r \xleftarrow{\$} \mathbb{Z}_p$ computes $R = g_1^r, S = G(i) \cdot A^r, T = (D \cdot E^{h_{\mathcal{R},\mathcal{S}}})^r, W = (U_1 \cdot U_2^{h_{\mathcal{R},\mathcal{S}}})^r$, where $h_{\mathcal{R},\mathcal{S}} = H(\mathbf{sid}, \mathbf{ssid}, \mathcal{R}, \mathcal{S}, R, S, c)$
3. $\mathcal{R}$ erases r, s, J, sends $c, C = (R, S, T)$ and keeps W, mask.

Database answer:

1. $\mathcal{S}$ sets $J = c_1 / c_2^\alpha$, $\mathsf{mask} = F(J)$ and $h_{\mathcal{R},\mathcal{S}} = H(\mathbf{sid}, \mathbf{ssid}, \mathcal{R}, \mathcal{S}, R, S, c)$
2. For $k = 1, \ldots, n$:
   (a) Picks $s_k \xleftarrow{\$} \mathbb{Z}_p$
   (b) Sets $\rho_k = B^{s_k}, \rho'_k = g_2^{s_k}, \theta_k = O^{s_k}, \gamma_k = (V_1 \cdot V_2^{h_{\mathcal{R},\mathcal{S}}})^{s_k}$
   (c) Computes $K_{\mathcal{R},\mathcal{S},k} = e(T, \rho'_k) \cdot e(S/G(k), \theta_k) \cdot e(R, \gamma_k)$
   (d) Sets $Q_k = L_k \oplus \mathcal{H}(K_{\mathcal{R},\mathcal{S},k} \oplus \mathsf{mask})$
3. $\mathcal{S}$ sends all $(Q_k, \rho_k)_{k \in [\![1,n]\!]}$ and erases everything else.

Data recovery:
$\mathcal{R}$ computes $K_{\mathcal{R},\mathcal{S},i} = e(W, \rho_i)$, and recovers $L_i = Q_i \oplus \mathcal{H}(K_{\mathcal{R},\mathcal{S},i} \oplus \mathsf{mask})$

Fig. 10. UC secure OT from QA-NIZK

The PAKE scheme directly fits in the framework, hence the security is inherited from the initial PAKE proven secure in [JR15].

The same basic arguments apply: We first start by switching the crs so as to be able to simulate password openings, which will allow an honest receiver to commit to a dummy line value, and retroactively compute the opening value for any possible line s. On the other hand, an honest server will extract the committed line number, and provide dummy answers for all the other ones. If the receiver is honest, then the server will also send a dummy answer for the valid line, since the encrypted value S provides the extra degree of freedom to open to any line value L_k provided belatedly by the environment.

6 Conclusion

We have just presented a framework for generically transforming any PAKE protocol into an Oblivious Transfer protocol. While doing so, we achieved to instantiate the most-efficient Oblivious-Transfer UC-Secure on elliptic curves.

We also propose an Oblivious-Transfer scheme on lattices in Appendix C , showing that our framework is not limited to regular elliptic curve cryptography. What remains now, is to find a UC-Secure Lattice based PAKE scheme.

References

[ABB+13] Michel Abdalla, Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, and David Pointcheval. SPHF-friendly non-interactive commitments. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 214–234. Springer, Heidelberg, December 2013.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Disjunctions for hash proof systems: New constructions and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 69–100. Springer, Heidelberg, April 2015.

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/2005/385`.

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, Heidelberg, August 2009.

[BBC+13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, Heidelberg, August 2013.

[BC15] Olivier Blazy and Céline Chevalier. Generic construction of UC-secure oblivious transfer. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 65–86. Springer, Heidelberg, June 2015.

[BC16] Olivier Blazy and Céline Chevalier. Structure-preserving smooth projective hashing. In *Advances in Cryptology - Proceedings of ASIACRYPT '16 (eprint 2016/258)*, Hanoi, Vietnam, December 2016. Springer.

[BCG16] Olivier Blazy, Céline Chevalier, and Paul Germouty. Adaptive oblivious transfer and generalization. In *Advances in Cryptology - Proceedings of ASIACRYPT '16 (eprint 2016/259)*, Hanoi, Vietnam, December 2016. Springer.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BFI+10] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pages 218–235. Springer, Heidelberg, June 2010.

[BKP14] Olivier Blazy, Eike Kiltz, and Jiaxin Pan. (Hierarchical) identity-based encryption from affine message authentication. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 408–425. Springer, Heidelberg, August 2014.

[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.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, Heidelberg, August 2006.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[CDVW12] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012.

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

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, Heidelberg, April / May 2002.

[CKWZ13] Seung Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. Efficient, adaptively secure, and composable oblivious transfer with a single, global CRS. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 73–88. Springer, Heidelberg, February / March 2013.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[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, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

[EG14] Alex Escala and Jens Groth. Fine-tuning Groth-Sahai proofs. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 630–649. Springer, Heidelberg, March 2014.

[GH08] Matthew Green and Susan Hohenberger. Universally composable adaptive oblivious transfer. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 179–197. Springer, Heidelberg, December 2008.

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

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

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

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Groth-Sahai proofs revisited. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 177–192. Springer, Heidelberg, May 2010.

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 505–523. Springer, Heidelberg, August 2009.

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007.

[JR13] Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2013.

[JR14] Charanjit S. Jutla and Arnab Roy. Switching lemma for bilinear tests and constant-size NIZK proofs for linear subspaces. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 295–312. Springer, Heidelberg, August 2014.

[JR15] Charanjit S. Jutla and Arnab Roy. Dual-system simulation-soundness with applications to UC-PAKE and more. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 630–655. Springer, Heidelberg, November / December 2015.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 78–95. Springer, Heidelberg, May 2005.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, December 2009.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, March 2011.

[KW15] Eike Kiltz and Hoeteck Wee. Quasi-adaptive NIZK for linear subspaces revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 101–128. Springer, Heidelberg, April 2015.

[Lin11] Yehuda Lindell. Highly-efficient universally-composable commitments based on the DDH assumption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 446–466. Springer, Heidelberg, May 2011.

[LPJY14] Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Non-malleability from malleability: Simulation-sound quasi-adaptive NIZK proofs and CCA2-secure encryption from homomorphic signatures. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 514–532. Springer, Heidelberg, May 2014.

[Ngu05] Minh-Huyen Nguyen. The relationship between password-authenticated key exchange and other cryptographic primitives. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 457–475. Springer, Heidelberg, February 2005.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR81, Harvard University, 1981.

A Additional Definitions

A.1 Encryption

An encryption scheme $\mathcal{C}$ is described through four algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}$):

- $\mathsf{Setup}(1^{\mathfrak{K}})$, where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme;
- $\mathsf{KeyGen}(\mathsf{param})$ outputs a pair of keys, a (public) encryption key pk and a (private) decryption key dk;
- $\mathsf{Encrypt}(\mathsf{ek}, M; \rho)$ outputs a ciphertext $\mathcal{C}$, on M, under the encryption key pk, with the randomness ρ;
- $\mathsf{Decrypt}(\mathsf{dk}, \mathcal{C})$ outputs the plaintext M, encrypted in the ciphertext $\mathcal{C}$ or $\perp$.

Such encryption scheme is required to have the following security properties:

- *Correctness*: For every pair of keys $(\mathsf{ek}, \mathsf{dk})$ generated by KeyGen, every messages M, and every random ρ, we should have $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{Encrypt}(\mathsf{ek}, M; \rho)) = M$.
- *Indistinguishability under Chosen Plaintext Attack* IND-CPA ([GM84]):

- IND-CPA: An adversary should not be able to efficiently guess which message has been encrypted even if he chooses the two original plaintexts.

$\mathsf{Exp}^{\mathsf{ind\text{-}cpa}-b}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$
1. $\mathsf{param} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
2. $(\mathsf{pk}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\mathsf{param})$
3. $(M_0, M_1) \leftarrow \mathcal{A}(\texttt{FIND} : \mathsf{ek})$
4. $c^* \leftarrow \mathsf{Encrypt}(\mathsf{ek}, M_b)$
5. $b' \leftarrow \mathcal{A}(\texttt{GUESS} : c^*)$
6. $\texttt{RETURN}\ b'$

A.2 Commitments

Formal definitions and results from [ABB+13] are given in Appendix **??** but we give here an informal overview to help the unfamiliar reader with the following. A *non-interactive labelled commitment scheme* $\mathcal{C}$ is defined by three algorithms:

- $\mathsf{SetupCom}(1^{\mathfrak{K}})$ takes as input the security parameter $\mathfrak{K}$ and outputs the global parameters, passed through the CRS ρ to all other algorithms;
- $\mathsf{Com}^{\ell}(x; \rho)$ takes as input a label ℓ, a message x and a random ρ, and outputs a pair (C, δ), where C is the commitment of x for the label ℓ, and δ is the corresponding opening data (a.k.a. decommitment information).
- $\mathsf{VerCom}^{\ell}(C, x, \delta)$ takes as input a commitment C, a label ℓ, a message x, and the opening data δ and outputs 1 (true) if δ is a valid opening data for C, x and ℓ. It always outputs 0 (false) on $x = \perp$.

The basic properties required for commitments are *correctness* (for all correctly generated CRS ρ, all commitments and opening data honestly generated pass the verification VerCom test), the *hiding property* (the commitment does not leak any information about the committed value) and the *binding property* (no adversary can open a commitment in two different ways).

A commitment scheme is said *equivocable* if it has a second setup algorithm $\mathsf{SetupComT}(1^{\mathfrak{K}})$ that additionally outputs a trapdoor τ, and two algorithms

- $\mathsf{SimCom}^{\ell}(\tau)$ that takes as input the trapdoor τ and a label ℓ and outputs a pair (C, eqk), where C is a commitment and eqk an equivocation key;
- $\mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, x)$ that takes as input a commitment C, a label ℓ, a message x, an equivocation key eqk, and outputs an opening data δ for C and ℓ on x.

such as the following properties are satisfied: *trapdoor correctness* (all simulated commitments can be opened on any message), *setup indistinguishability* (one cannot distinguish the CRS ρ generated by $\mathsf{SetupCom}$ from the one generated by $\mathsf{SetupComT}$) and *simulation indistinguishability* (one cannot distinguish a real commitment (generated by Com) from a fake commitment (generated by SCom), even with oracle access to fake commitments), denoting by SCom the algorithm that takes as input the trapdoor τ, a label ℓ and a message x and which outputs $(C, \delta) \xleftarrow{\$} \mathsf{SCom}^{\ell}(\tau, x)$, computed as $(C, \mathsf{eqk}) \xleftarrow{\$} \mathsf{SimCom}^{\ell}(\tau)$ and $\delta \leftarrow \mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, x)$.

A commitment scheme $\mathcal{C}$ is said *extractable* if it has a second setup algorithm $\mathsf{SetupComT}(1^{\mathfrak{K}})$ that additionally outputs a trapdoor τ, and a new algorithm

- $\mathsf{ExtCom}^{\ell}(\tau, C)$ which takes as input the trapdoor τ, a commitment C, and a label ℓ, and outputs the committed message x, or $\perp$ if the commitment is invalid.

such as the following properties are satisfied: *trapdoor correctness* (all commitments honestly generated can be correctly extracted: for all ℓ, x, if $(C, \delta) \xleftarrow{\$} \mathsf{Com}^{\ell}(x)$ then $\mathsf{ExtCom}^{\ell}(C, \tau) = x$), *setup indistinguishability* (as above) and *binding extractability* (one cannot fool the extractor, *i.e.*, produce a commitment and a valid opening data to an input x while the commitment does not extract to x).

A.3 Smooth Projective Hashing and Languages

Smooth projective hash functions (SPHF) were introduced by Cramer and Shoup in [CS02] for constructing encryption schemes. A projective 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) *projected* key one can only compute the function on a special subset of its domain. Such a family is deemed *smooth* if the value of the hash function on any point outside the special subset is independent of the projected key. The notion of SPHF has already found applications in various contexts in cryptography (*e.g.* [GL03, Kal05, ACP09]). A Smooth Projective Hash Function over a language $\mathfrak{L} \subset X$, onto a set $\mathcal{G}$, is defined by five algorithms $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$:

- $\mathsf{Setup}(1^{\mathfrak{K}})$ where $\mathfrak{K}$ is the security parameter, generates the global parameters param of the scheme, and the description of an $\mathcal{NP}$ language $\mathfrak{L}$;
- $\mathsf{HashKG}(\mathfrak{L}, \mathsf{param})$, outputs a hashing key hk for the language $\mathfrak{L}$;
- $\mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, derives the projection key hp from the hashing key hk.
- $\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W)$, outputs a hash value $v \in \mathcal{G}$, thanks to the hashing key hk and W.
- $\mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w)$, outputs the hash value $v' \in \mathcal{G}$, thanks to the projection key hp and the witness w that $W \in \mathfrak{L}$.

In the following, we consider $\mathfrak{L}$ as a hard-partitioned subset of X, *i.e.* it is computationally hard to distinguish a random element in $\mathfrak{L}$ from a random element in $X \setminus \mathfrak{L}$. An SPHF should satisfy the following properties:

- *Correctness*: Let $W \in \mathfrak{L}$ and w a witness of this membership. Then, for all hashing keys hk and associated projection keys hp we have

$$\mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) = \mathsf{ProjHash}(\mathsf{hp}, (\mathfrak{L}, \mathsf{param}), W, w).$$

- *Smoothness*: For all $W \in X \setminus \mathfrak{L}$ the following distributions are statistically indistinguishable:

$$\Delta_0 = \left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), \\ v = \mathsf{Hash}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W) \end{array} \right\}$$

$$\Delta_1 = \left\{ (\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) \;\middle|\; \begin{array}{l} \mathsf{param} = \mathsf{Setup}(1^{\mathfrak{K}}), \mathsf{hk} = \mathsf{HashKG}(\mathfrak{L}, \mathsf{param}), \\ \mathsf{hp} = \mathsf{ProjKG}(\mathsf{hk}, (\mathfrak{L}, \mathsf{param}), W), v \xleftarrow{\$} \mathcal{G} \end{array} \right\}.$$

  This is formalized by: $\mathsf{Adv}^{\mathsf{smooth}}_{\mathsf{SPHF}}(\mathfrak{K}) = \sum_{V \in \mathbb{G}} |\Pr_{\Delta_1}[v = V] - \Pr_{\Delta_0}[v = V]|$ is negligible.
- *Pseudo-Randomness*: If $W \in \mathfrak{L}$, then without a witness of membership the two previous distributions should remain computationally indistinguishable. For any adversary $\mathcal{A}$ within reasonable time, this advantage is negligible:

$$\mathsf{Adv}^{\mathsf{pr}}_{\mathsf{SPHF},\mathcal{A}}(\mathfrak{K}) = |\Pr_{\Delta_1}[\mathcal{A}(\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1] - \Pr_{\Delta_0}[\mathcal{A}(\mathfrak{L}, \mathsf{param}, W, \mathsf{hp}, v) = 1]|$$

B Proof of the Generic Framework

To prove this theorem, we exhibit a sequence of games. The sequence starts from the real game, where the adversary $\mathscr{A}$ interacts with real players and ends with the ideal game, where we have built a simulator $\mathscr{S}$ that makes the interface between the ideal functionality $\mathcal{F}$ and the adversary $\mathscr{A}$. We prove the adaptive version of the protocol. The proof of the static framework version can be obtained by removing the parts related to adaptive version from the proof below.

When the sender submits its values, the simulator can extract all the message thanks to the crs trapdoor and get the witnesses for each indices. This allows to simulate the `Send`-query to the ideal functionality.

Eventually, when simulating the honest senders, the simulator recovers the expected line value s from flow_0, to set flow_s and L_s honestly, the other values can be random. More details follow:

Game G_0: This is the real game.

Game G_1: In this game, the simulator generates correctly every flow from the honest players, as they would do themselves, knowing the inputs $(L_1, \ldots, L_n)$ and s sent by the environment to the sender and the receiver. In all the subsequent games, the players use the label $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$. In case of corruption, the simulator can give the internal data generated on behalf of the honest players.

Game G_2: In this game, we just replace the setup algorithm (if any), by the one used in the PAKE proof, as this is transparent in the PAKE protocol, this is also transparent in the Oblivious Transfer.

Game G_3: We first deal with **honest senders** P_i: when receiving a first flow flow_0, the simulator recovers the prospective line value s (this is because, we assume the UC-PAKE secure, hence after flow_1, the simulator should be able to send $\mathtt{TestPwd}, \mathsf{sid}, \mathsf{ssid}, P_i, s$, hence knows said s). Instead of computing the key K_t, for $t = 1, \ldots, n$ for all possible line/passwords, it chooses K_t at random in the key space for $t \neq s$. With an hybrid proof, for every honest sender, on every index $t \neq s$, since flow_0 leads to s, for any $t \neq s$, the new key is indistinguishable from a random value, under the PAKE security.
In case of corruption, everything is either already handled by the PAKE, or erased. This game is thus indistinguishable from the previous one under the PAKE functionality.

Game G_4: Still in this case, when receiving flow_0, the simulator recovers the expected value s. Instead of proceeding as the sender would do on lines $(L_1, \ldots, L_n)$, the simulator proceeds on $(L'_1, \ldots, L'_n)$, with $L'_s = L_s$, but $m'_t = 0$ for all $t \neq s$. Since the masks K_t, for $t \neq s$, are random, this game is perfectly indistinguishable from the previous one.

Game G_5: We now deal with **honest receivers** P_j: we replace flow_1 for in Step 1 of the index query phase of honest receivers by simulated flow_1, using the fact that the UC-Proof requires the simulator at this step to send a non committing flow compatible with every possible password, and keep the PAKE memory.
Using the PAKE security, one can show the indistinguishability of the two games. In case of corruption of the receiver, one learns the already known value s.

Game G_6: We deal with **the generation of R for honest senders P_i on honestly-generated queries (adaptive case only)**: if P_i and P_j are honest at least until P_i received the second flow, the simulator sets $R = F(S')$ for both P_i and P_j, with S' a random value, instead of $R = F(S)$.
With an hybrid proof, applying the IND-CPA property for each session, one can show the indistinguishability of this game with the previous one.

Game G_7: Still in the same case, the simulator sets R as a random value, instead of $R = F(S')$.
With an hybrid proof, applying the PRF property for each session, one can show the indistinguishability of this game with the previous one.

Game G_8: We now deal with **the generation of K_s for honest senders P_i on honestly-generated queries**:

- in the static case (the pre-flow is not necessary, and thus we assume $R = 0$) the simulator chooses K_s at random in the Key Space (for $t \neq s$, the simulator already chooses K_t random), where s is the index given by the ideal functionality to the honest receiver P_j.
  Under the PAKE security, for every honest sender, the key value is indistinguishable from a random value, because the adversary does not know any secret information on flow_0;
- in the adaptive case, and thus with the additional random mask R, one can send a random Q_s, and K_s can be computed later (when P_j actually receives its flow, and the simulator possibly learns the expected L_s).
  As above, but only if P_j has not been corrupted before receiving its flow, the simulator chooses K_s at random. Once again, invoking the PAKE security, and the lack of secret information known by the adversary, this value is indistinguishable from a random value. If the player P_j gets corrupted we are not in this case, and we can thus abort it.
  In case of corruption of P_i, everything has been erased and / or handled by the PAKE simulator. In case of corruption of the receiver P_j, and thus receiving the value L_s, the simulator chooses R (because it was a random value unknown to the adversary and all the other K_t are independent random values too) such that $R \oplus K_s \oplus Q_s = L_s$.

This game is thus indistinguishable from the previous one under the PAKE Security (External Adversary, Adaptive Corruption of the second User).

Game G_9: Still in this case, the simulator proceeds on $(L'_1, \ldots, L'_n)$, with $L'_t = 0$ for all t. Since the masks $K_t \oplus R$, for any $t = 1, \ldots, n$, are independent random values (the K_t, for $t \neq s$ are independent random values, and K_s is also independently random in the static case, while R is independently random in the adaptive case), this game is perfectly indistinguishable from the previous one.

We remark that it is therefore no more necessary to know the index s given by the ideal functionality to the honest receiver P_j, to simulate P_i (but it is still necessary to simulate P_j).

Game G_{10}: We do not use anymore the knowledge of s when simulating an **honest receiver** P_j: the simulator generates an equivocable first flow flow_0 during the index query phase of honest receivers. We essentially break the atomic flow_1 from the PAKE in the two separated processes SimFlow and OpenFlow, used by the simulator in the PAKE proof, and stores the equivocation information Λ, instead of just $f(\rho)$. This does not change anything from the previous game since the memory before flow_0 is never revealed. It can be updated with correct values in case of corruption of the receiver.

When it thereafter receives the message $(\texttt{Send}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, (Q_1, \ldots, Q_n))$ from the adversary, the simulator computes, for $i = 1, \ldots, n$, the value $f(\rho_i) \leftarrow \mathsf{OpenFlow}^{\ell}(\mathsf{flow}_0, i, \Lambda)$, $K_i \leftarrow \mathsf{Decap}(\mathsf{flow}_i, i, f(\rho_i))$ and $m_i = K_i \oplus R \oplus Q_i$. This provides the database submitted by the sender.

Game G_{11}: We can now make use of the functionality, which leads to the following simulator:

- when receiving a `Send`-message from the ideal functionality, which means that an honest sender has sent a pre-flow, the simulator generates a key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^{\mathfrak{K}})$ and sends pk as pre-flow;
- after receiving a pre-flow pk (from an honest or a corrupted sender) and a `Receive`-message from the ideal functionality, which means that an honest receiver has sent an index query, the simulator generates $(\mathsf{flow}_0, \Lambda) \xleftarrow{\$} \mathsf{SimFlow}^{\ell}()$ and $c \xleftarrow{\$} \mathsf{Encrypt}(\mathsf{pk}, S)$, with $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ and R a random value, to send flow_1 and c during the index query phase of the honest receiver;
- when receiving flow_0 and a ciphertext c, generated by the adversary (from a corrupted receiver), the simulator recovers the committed value s, and uses it to send a `Receive`-message to the ideal functionality (and also decrypts the ciphertext c as S, and computes $R = F(S)$);
- when receiving $(\mathcal{Q}_1, \ldots, Q_n)$ from the adversary (a corrupted sender), the simulator computes, for $i = 1, \ldots, n$, $f(\rho_i) \leftarrow \mathsf{OpenFlow}^{\ell}(\mathsf{flow}_0, i, \Lambda)$, $K_i \leftarrow \mathsf{Decap}(\mathsf{flow}_i, i, f(\rho_i))$ and $L_i = K_i \oplus R \oplus Q_i$. It uses them to send a `Send`-message to the ideal functionality.
- when receiving a `Received`-message from the ideal functionality, together with L_s, on behalf of a corrupted receiver, from the recovered s, instead of proceeding as the sender would do on $(L_1, \ldots, L_n)$, the simulator proceeds on $(L'_1, \ldots, L'_n)$, with $L'_s = L_s$, but $L'_i = 0$ for all $i \neq s$;
- when receiving a a flow flow_0 and a ciphertext c, generated by an honest sender (*i.e.*, by the simulator itself), the simulator proceeds as above on $(L'_1, \ldots, L'_k)$, with $L'_i = 0$ for all i, but it chooses R uniformly at random instead of choosing it as $R = F(S)$; in case of corruption afterward, the simulator will adapt R such that $R \oplus K_s \oplus Q_s = L_s$, where L_s is the message actually received by the receiver.

Any corruption either reveals s earlier, which allows a correct simulation of the receiver, or reveals $(L_1, \ldots, L_n)$ earlier, which allows a correct simulation of the sender. When the sender has sent his flow, he has already erased all his random coins (or at least enough to be compatible with the PAKE key). However, there would have been an issue when the receiver is corrupted after the sender has sent is flow, but before the receiver receives it, since he has kept some witness $f(\rho)$: this would enable the adversary to recover L_s from Q_s. This is the goal of the ephemeral mask R that provides a secure channel.

As a consequence, the distance between the first and the last games is bounded by q times the PAKE security, and the encryption CPA indistinguishability.

Some interesting to be noted: We never use the fact, that the first user in the PAKE does not learn the password from the second. This is a property required by PAKE but not useful in the transformation. This argument is key, in the optimizations we propose later on.

C Beyond UC, applying the framework to a BPR Lattice PAKE

C.1 Description of their secure PAKE Scheme

We are now going to show what happens when our general technique is used even on lattice-based PAKE. As an interesting example, we will apply it to the PAKE from [KV09]. Lattices are drastically lacking a UC-PAKE, however we are going to apply our framework to this BPR secure scheme [BPR00].

We can see that this PAKE protocol is made of three flows. However the last flow is there to perform a key confirmation. The approximate SPHF used ensures that the two values $(\mathsf{tk}, \mathsf{tk}')$ completed are close-enough (in term of Hamming weight), so that the random key sk generated and coded into c can then be decoded.

CRS generation:
$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{param})$ for $\mathcal{E}$. $\mathsf{crs} = (\mathsf{pk})$
$\mathsf{flow}_1(\mathsf{pw}; r_1)$:

1. P_1 computes $C_1 = \mathsf{Encrypt}(\mathsf{pk}, \ell_1, \mathsf{pw}; r_1)$ where $\ell_1 = (\mathsf{sid}, \mathsf{ssid}, P_1, P_2)$
2. P_1 sends C_1 to P_2

$\mathsf{flow}_2(\mathsf{pw}; r_2)$:

1. P_2 computes $C_2 = \mathsf{Encrypt}(\mathsf{pk}, \epsilon, \mathsf{pw}; r_2)$.
2. P_2 generates $\mathsf{hk}_2 \leftarrow \mathsf{HashKG}(\mathfrak{L}_{\mathsf{pw},\ell_1})$, $\mathsf{hp}_2 \leftarrow \mathsf{ProjKG}(\mathsf{hk}_2, \mathfrak{L}_{\mathsf{pw},\ell_1}, C_2)$.
3. P_2 sends (C_2, hp_2) to P_1.

$\mathsf{flow}_3(\mathsf{pw}; r_2)$:

1. P_1 generates $\mathsf{hk}_1 \xleftarrow{\$} \mathsf{HashKG}(\mathfrak{L}_{\mathsf{pw},\epsilon})$, $\mathsf{hp}_1 \xleftarrow{\$} \mathsf{ProjKG}(\mathsf{hk}_1, \mathfrak{L}_{\mathsf{pw},\epsilon}, C_1)$.
2. P_1 computes $\mathsf{tk} = \mathsf{Hash}(\mathsf{hk}_1, \mathfrak{L}_{\mathsf{pw},\epsilon}, C_2) \cdot \mathsf{ProjHash}(\mathsf{hp}_2, \mathfrak{L}_{\mathsf{pw},\ell_1}, C_1, r_1)$
3. P_1 chooses $\mathsf{sk} \xleftarrow{\$} \{0,1\}^k$, computes $c = \mathsf{ECC}(\mathsf{sk})$
4. P_1 set $\Delta = \mathsf{tk} \oplus c$.
5. P_1 sends (Δ, hp_1) to P_2.

Key computing for P_2:

1. P_2 computes $\mathsf{tk} = \mathsf{Hash}(\mathsf{hk}_2, \mathfrak{L}_{\mathsf{pw},\ell_1}, C_1) \cdot \mathsf{ProjHash}(\mathsf{hp}_1, \mathfrak{L}_{\mathsf{pw},\ell_2}, C_2, r_2,)$
2. P_2 recover the key $\mathsf{sk} = \mathsf{ECC}^{-1}(\mathsf{tk} \oplus \Delta)$

Fig. 11. Lattice-based PAKE from [KV09]

C.2 Applying the Framework to Obtain a Secure OT Scheme

However, since with our framework the server does not need to commit to his password (the line index here), we can slightly modify the protocol and end up with the OT protocol exposed on Figure 12.

The security proof of this construction is a little beyond the point of the paper, as the resulting scheme is not UC-Secure[5], however the BPR security of the PAKE ensures some minimum requirements.

As a participant in the PAKE can not learn the other password, this ensures that even after seeing the receiver's flows, the server does not learn the line requested.

Similarly, since in case of password mismatch, a user cannot guess the key computed by the other one, this ensures that the user accesses at most one line.

In terms of efficiency, the resulting Oblivious Transfer ends up being slightly more efficient than both UC constructions from [GH08, BC15], but this comes at no surprise considering the security requirement gap. However this remains promising for the validity of our framework.

[5] There is a problem to simulate an honest user. The Index query cannot be equivocated as the approximate SPHF Smoothness prevents the simulator to this circumvent this requirement.

CRS generation:
$\mathsf{param}' \xleftarrow{\$} \mathsf{Setup}(1^{\mathfrak{K}}), (\mathsf{pk}', \mathsf{sk}') \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{param}')$
CRS: $\rho \xleftarrow{\$} \mathsf{SetupCom}(1^{\mathfrak{K}}), \mathsf{param}_{\mathrm{cpa}} \xleftarrow{\$} \mathsf{Setup}_{\mathrm{cpa}}(1^{\mathfrak{K}}), \mathsf{pk}'$.

Pre-flow :

1. $\mathcal{S}$ generates a LWE key pair $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathrm{cpa}}(\mathsf{param}_{\mathrm{cpa}})$ for $\mathcal{E}$
2. $\mathcal{S}$ stores sk, completely erase random coins used by $\mathsf{KeyGen}_{\mathrm{cpa}}$, and sends pk to $\mathcal{R}$

Index query on s:

1. $\mathcal{R}$ chooses a random J, computes $R \leftarrow F(J)$ and $c \xleftarrow{\$} \mathsf{Encrypt}_{\mathrm{cpa}}(\mathsf{pk}, J)$
2. $\mathcal{R}$ computes $C \leftarrow \mathsf{Encrypt}(\mathsf{pk}', \ell, s; r)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, \mathcal{R}, \mathcal{S})$
3. $\mathcal{R}$ stores r, R, completely erases the random coins used and sends C and c to $\mathcal{S}$.

Database input $(L_1, \ldots, L_n)$:

1. $\mathcal{S}$ computes $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and gets $R \leftarrow F(J)$
2. For $k = 1, \ldots, n$
   - $\mathcal{S}$ computes $\mathsf{hk}_k \xleftarrow{\$} \mathsf{HashKG}(\mathfrak{L}_{k,\ell})$, $\mathsf{hp}_k \leftarrow \mathsf{ProjKG}(\mathsf{hk}_k, \mathfrak{L}_{k,\ell}, C)$, $\mathsf{tk} \leftarrow \mathsf{Hash}(\mathsf{hk}_k, \mathfrak{L}_{k,\ell}, C)$,
   - $\mathcal{S}$ chooses $\mathsf{sk}_k \xleftarrow{\$} \{0,1\}^m$, sets $e_k = \mathsf{ECC}(\mathsf{sk}_k)$, and $\Delta_k = \mathsf{tk}_k \oplus e_k$
   - $\mathcal{S}$ computes $Q_k \leftarrow R \oplus \mathsf{sk}_k \oplus L_k$
3. $\mathcal{S}$ erases everything except $(\mathsf{hp}_k, Q_k, \Delta_k)_{k=1,\ldots,n}$ and sends it to $\mathcal{R}$

Data recovery:
Upon receiving $(\mathsf{hp}_t, Q_t)_{t=1,\ldots,k}$, $\mathcal{R}$ computes $\mathsf{tk}_s \leftarrow \mathsf{ProjHash}(\mathsf{hp}_s, \mathfrak{L}_{G(s),\ell}, C, r)$, $\mathsf{sk}_s = \mathsf{ECC}^{-1}(\Delta_s \oplus \mathsf{tk}_s)$, $L_s \leftarrow R \oplus \mathsf{sk}_s \oplus Q_s$, and erases everything besides L_s.

Fig. 12. 1-out-of-n OT from Lattice-based PAKE