Generic Construction of UC-Secure Oblivious Transfer

Olivier Blazy[1] and Céline Chevalier[2]

[1] Université de Limoges, XLim, France
[2] Université Panthéon-Assas, Paris, France

Abstract. We show how to construct a completely generic UC-secure oblivious transfer scheme from a collision-resistant chameleon hash scheme (CH) and a CCA encryption scheme accepting a smooth projective hash function (SPHF).
Our work is based on the work of Abdalla *et al.* at Asiacrypt 2013, where the authors formalize the notion of *SPHF-friendly* commitments, *i.e.* accepting an SPHF on the language of valid commitments (to allow *implicit* decommitment), and show how to construct from them a UC-secure oblivious transfer in a generic way. But Abdalla *et al.* only gave a DDH-based construction of SPHF-friendly commitment schemes, furthermore highly relying on pairings. In this work, we show how to generically construct an SPHF-friendly commitment scheme from a collision-resistant CH scheme and an SPHF-friendly CCA encryption scheme. This allows us to propose an instanciation of our schemes based on the DDH, as efficient as that of Abdalla *et al.*, but without requiring any pairing. Interestingly, our generic framework also allows us to propose an instantiation based on the learning with errors (LWE) assumption. For the record, we finally propose a last instanciation based on the decisional composite residuosity (DCR) assumption.

Keywords. Commitments, Smooth Projective Hash Functions, CCA encryption, Oblivious Transfer, UC Framework.

1 Introduction

Oblivious Transfer (OT) was introduced in 1981 by Rabin [Rab81] as a way to allow a receiver to get exactly one out of k messages sent by another party, the sender. In these schemes, the receiver should be oblivious to the other values, and the sender should be oblivious to which value was received. This primitive has been widely used and studied in the community, and recently, the authors of [ABB+13] propose a generic way to obtain a UC-secure oblivious transfer scheme from an SPHF-friendly commitment scheme, and an instantiation based on DDH. In this paper, our goal is to strengthen their result to obtain a truly generic way to obtain a UC-secure oblivious transfer scheme, so we follow their path of construction from commitment schemes.

Commitment schemes have become a very useful tool used in cryptographic protocols. These two-party primitives (between a committer and a receiver) are 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. El Gamal [ElG84] or Cramer-Shoup [CS02] encryptions are famous examples of perfectly binding commitments, and Pedersen encryption [Ped91] is the most known example of perfectly hiding commitments.

In many applications, for example password-based authenticated key-exchange in which the committed value is a password, one wants the decommitment to be implicit, which means that the committer does not really open its commitment, but rather convinces the receiver that it actually committed to the value it pretended to. In [ACP09], the authors achieved this property thanks to the notion of *Smooth Projective Hash Functions* [CS02, GL03], which has been widely used since then (see [KV11, BBC^{+}13b, ABB^{+}13] for instance). These 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 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*).

An additional difficulty arises when one wants to prove the protocols in the universal composability framework proposed in [Can01]. In a nutshell, security in the UC framework is captured by an ideal functionality (in an ideal world) and a protocol is proven secure if, given any adversary to the protocol in the real world, one can construct a simulator of this adversary such that no environment can distinguish between the execution in the ideal world (between dummy players, the ideal functionality and the simulator of the adversary) and the execution in the real world (between the real players executing the real protocol and interacting between themselves and the adversary) in a non-negligible way. Skipping the details, when the protocol makes use of commitments, this usually forces those commitments to be both *extractable* (meaning that a simulator can recover the value m committed to thanks to a trapdoor) and *equivocable* (meaning that a simulator can open a commitment to a value m' different from the value m it committed to thanks to a trapdoor), which is quite a difficult goal to achieve.

The now classical way [CF01, ACP09, ABB^{+}13] to achieve both extractability and equivocability is to combine an equivocable CPA encryption scheme (such as Pedersen [Ped91]) and an extractable CCA encryption scheme (such as Cramer-Shoup [CS02]) and to link them with an SPHF in order to obtain an implicit decommitment. What we show in this paper is that we can broaden the class of primitives that can be used for the equivocable part, by using chameleon hashes (introduced in [KR00]), which can be seen as conceptually easier building blocks to understand and to construct.

Related Work. The first UC-secure commitment schemes were given by [CF01] and [DN02] and the former were the first to formalize the methodology described in the previous section (combining an equivocable primitive and an extractable primitive). Building on their idea, the authors of [ACP09] add the notion of smooth projective hash function to obtain implicit decommitment and obtain the first UC-secure password-authenticated key-exchange in the standard model as an application. Many works have been done in the same field since then, for instance [Lin11, FLM11, BCPV13] for the UC-commitment schemes and [KV11, BBC^{+}13b] for the UC PAKE schemes, in which the relations between commitments and SPHF have proven being very useful. This relation was formalized in [ABB$^+$13] by the notion of SPHF-*friendly commitments*, expliciting the properties to be fulfilled by the commitment in order to accept an SPHF (and thus to be very useful for all kinds of applications). The authors also prove that their new notion of SPHF-friendly commitments is strictly stronger than the notion of UC commitments and give an example of such a commitment scheme based on Haralambiev commitment [Har11, Section 4.1.4] and Cramer-Shoup encryption, in a pairing-friendly setting. They also propose a generic way to construct UC one-round PAKE and oblivious transfer scheme from this primitive.

Many oblivious transfer schemes have been proposed since [Rab81], including some in the UC framework [NP01,CLOS02]. Recently, some instantiations have tried to reach round-optimality [HK07], or low communication costs [PVW08]. As already explained, the authors of [ABB$^+$13] propose a generic way to obtain a UC-secure oblivious transfer scheme from an SPHF-friendly commitment scheme, and an instantiation based on DDH. Choi *et al.* [CKWZ13] also propose a generic method and an efficient instantiation secure against adaptive corruptions in the CRS model with erasures, based on DDH, but it is only 1-out-of-2 and it does not scale to 1-out-of-k OT, for $k > 2$.

Contributions[1]**.** Our first contribution is to give a generic construction of SPHF-friendly commitments, which have proven since [ABB$^+$13] to be an extremely useful primitive, from two simple blocks: a collision-resistant chameleon hash (CH) function which is verifiable (either publicly or for the receiver only) and an SPHF-friendly CCA encryption scheme. The extra requirement on the CH function is simple to achieve as soon as only classical algebraic operations are applied to the randomness, and SPHF-friendly encryption is now well-known since [CS02], with several instances (contrary to SPHF-friendly commitments, which is a difficult task). We then give three instantiations of this SPHF-friendly scheme, respectively based on DDH, LWE and DCR.

Our construction thus allows us to provide, as a second and main contribution, a generic way to obtain a UC-secure OT scheme from the same building blocks (CH and CCA encryption) and three concrete instantiations from DDH, LWE and DCR. While the construction in [ABB$^+$13] is an ad hoc solution with pairings, ours is generic and does not specifically induce pairings. Furthermore, our 3 instantiations come straightforward from our generic framework (and [ABB$^+$13] can be derived from it).

Concerning complexity comparisons, the most studied assumptions in the literature are variants of DDH. Our version of 1-out-of-t oblivious transfer is apparently almost equivalent to that given in [ABB$^+$13] in raw number of elements because we need a communication complexity of $9m+6$ elements in $\mathbb{G}$ and a scalar, compared to $8m+4$[2] in $\mathbb{G}_1$, m in $\mathbb{G}_2$ and a scalar (with $t = 2^m$), but since we do not need a pairing-friendly setting, none of our elements have to be bigger, hence the comparison is in favor of our new proposal (by an equivalent of $m/2 - 1$ elements). (Those numbers do not take into account in both cases the last message that transmits the database, adding an additional m elements in both cases).

[1] This is an extended abstract. The full paper [BC15] is available at the Cryptology Eprint Archive, `http://eprint.iacr.org`.

[2] It should be noted that their original computation was off by one scalar, probably half the projection key was missing.

To compare with existing protocols in the case of 1-out-of-2 under SXDH, [ABB$^+$13] needs 12 elements in $\mathbb{G}_1$, and 1 in $\mathbb{G}_2$ during 3 rounds (some elements previously in $\mathbb{G}_2$ can be transferred into $\mathbb{G}_1$ in this case, and one can be skipped), [CKWZ13] requires 26 group elements and 7 scalars in 4 rounds ; and using [GWZ09] to achieve a constant-size CRS, [PVW08] requires 8 rounds and 51 elements. Using plain DDH, we need 15 group elements (but because [ABB$^+$13] requires one in $\mathbb{G}_2$ we have strictly the same communication cost with a better scaling and no pairing computation) hence under classical instantiation both schemes require to transmit roughly 3200 bits of data.

Communication cost comparisons of various Elliptic Curve based OT schemes

Paper	Assumption	# Group elements	# Rounds
Static Security			
[PVW08] + [GWZ09]	SXDH	51	8
[CKWZ13]	SXDH	26 + 7s	4
Adaptive Security			
[ABB$^+$13]	SXDH	12 $\mathbb{G}_1$ + 1 $\mathbb{G}_2$	3
This paper	DDH	15	3

Considering classical instantiations on Barreto-Naehrig Curves [BN05], elements on a DDH curve are at least twice smaller than the big ones on a SXDH one, making our scheme have a better scaling for 1-out-of-m OT. With recent attacks exploiting the existence of a pairing, managing to maintain the efficiency while removing the need for a pairing structure is a strong asset of elliptic curve based cryptography. For constructions based on generic hypothesis, the construction of [PVW08] leads to a non constant size CRS (in the number of user), while ours achieve constant (and small) CRS size.

2 Definitions

In this section we recall classical definitions and tools that are going to be useful in the rest of the paper.

Commitments. Formal definitions and results from [ABB$^+$13] are given in the full version 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;
- $\mathsf{Com}^{\ell}(x)$ takes as input a label ℓ and a message x, 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). This is a probabilistic algorithm.
- $\mathsf{VerCom}^{\ell}(C, x, \delta)$ 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 $x = \bot$.

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 $\mathsf{SetupComT}(1^{\mathfrak{K}})$ 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 $(C, \delta) \overset{\$}{\leftarrow} \mathsf{SCom}^{\ell}(\tau, x)$, computed as $(C, \mathsf{eqk}) \overset{\$}{\leftarrow} \mathsf{SimCom}^{\ell}(\tau)$ and $\delta \leftarrow \mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, x)$.

A commitment scheme $\mathcal{C}$ is said *extractable* if it has a second setup $\mathsf{SetupComT}(1^{\mathfrak{K}})$ 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 $(C, \delta) \overset{\$}{\leftarrow} \mathsf{Com}^{\ell}(x)$ 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).

We recall in Section 3 the difficulties implied by a commitment being both equivocable and extractable and give a construction of such a commitment.

Smooth Projective Hash Function. 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 $X \setminus \mathfrak{L}$.

A Smooth Projective Hash Function 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:

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

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

Labelled Encryption Scheme. A labelled public-key encryption scheme $\mathcal{E}$ is defined by four algorithms:

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

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

- *Correctness*: for all key pair $(\mathsf{pk}, \mathsf{sk})$, any label ℓ, all random coins r and all messages m, $\mathsf{Decrypt}^{\ell}(\mathsf{sk}, \mathsf{Encrypt}^{\ell}(\mathsf{pk}, m; r)) = m$.

– *Indistinguishability under chosen-ciphertext attacks*: this security notion IND-CCA can be formalized by the following experiments $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ind\text{-}cca}-b}(\mathfrak{K})$, where the adversary $\mathcal{A}$ transfers some internal state state between the various calls `FIND` and `GUESS`, and makes use of the oracle ODecrypt:

```
$\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ind\text{-}cca}-b}(\mathfrak{K})$
    $\mathsf{param} \overset{\$}{\leftarrow} \mathsf{Setup}(1^{\mathfrak{K}})$
    $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(\mathsf{param})$
    $(\ell^*, m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{ODecrypt}^{\cdot}(\cdot)}(\texttt{FIND} : \mathsf{pk})$
    $c^* \leftarrow \mathsf{Encrypt}^{\ell^*}(\mathsf{pk}, m_b)$
    $b' \leftarrow \mathcal{A}^{\mathsf{ODecrypt}^{\cdot}(\cdot)}(\mathsf{state}, \texttt{GUESS} : c^*)$
    If $((\ell^*, c^*) \in \mathcal{CT})$    Return 0
    Else    Return $b'$
```

  - $\mathsf{ODecrypt}^{\ell}(c)$: This oracle outputs the decryption of c under the label ℓ and the challenge decryption key sk. The input queries (ℓ, c) are added to the list $\mathcal{CT}$.

  These experiments implicitly define the advantages $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cca}}(\mathcal{A}, \mathfrak{K})$ and $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cca}}(t)$. One sometimes uses the notation $\mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind\text{-}cca}}(q_d, t)$ to bound the number of decryption queries.

In the following we also want two additional properties. First we want an additional functionality, we want to be able to supersede the decryption, by an implicit decommitment. So we require the encryption to admit an efficient implicit decommitment. We will call an SPHF-friendly encryption, an encryption where there exists an SPHF for the Language of valid ciphertexts of a message m using as sole witness the randomness used in the encryption.

We then are going to want to strengthen the idea of ind-cca encryption. In the sense that we are going to encrypt vector of messages, and when the challenges vectors shares some component we want to provide the randomness used specifically for those components to the adversary. (Intuitively this would be done to allow an honest computation of the SPHF on this part). In [ABB+13], they call such property `VIND-PO-CCA` for Partial Opening, and show that Cramer-Shoup encryption obeys such property. We recall this security notion in the full version for the sake of completeness. We denote by $\mathsf{nEncrypt}^{\ell}(\mathsf{pk}, m; r)$ and $\mathsf{nDecrypt}^{\ell}(\mathsf{sk}, c)$ the corresponding algorithms for encryption or decryption of vectors of n bits.

Chameleon Hash. A Chameleon Hash Function is traditionally defined by three algorithms $\mathsf{CH} = (\mathsf{KeyGen}, \mathsf{CH}, \mathsf{Coll})$:

– $\mathsf{KeyGen}(\mathfrak{K})$: Outputs the chameleon hash key ck and the trapdoor tk;
– $\mathsf{CH}(\mathsf{ck}, m; r)$: Picks a random r, and outputs the chameleon hash a.
– $\mathsf{Coll}(\mathsf{ck}, m, r, m', \mathsf{tk})$: Takes as input the trapdoor tk, a start message and randomness pair (m, r) and a target message m' and outputs a target randomness r' such that $\mathsf{CH}(\mathsf{ck}, m; r) = \mathsf{CH}(\mathsf{ck}, m'; r')$.

The standard security notion for CH is collision resistance, which means it is infeasible to find $(m_1, r_1), (m_2, r_2)$ such that $\mathsf{CH}(\mathsf{ck}, m_1, r_1) = \mathsf{CH}(\mathsf{ck}, m_2, r_2)$ and $m_1 \neq m_2$ given only the Chameleon hash key ck. Formally, CH is $(t, \varepsilon) - \mathsf{coll}$ if for the adversary $\mathcal{A}$ running in time at most t we have:

$$\Pr\begin{bmatrix}(\mathsf{ck}, \mathsf{tk}) \overset{\$}{\leftarrow} \mathsf{KeyGen}(\mathfrak{K}); ((m_1, r_1), (m_2, r_2)) \overset{\$}{\leftarrow} \mathcal{A}(\mathsf{ck}) \\ \wedge \quad \mathsf{CH}(\mathsf{ck}, m_1; r_1) = \mathsf{CH}(\mathsf{ck}, m_2; r_2) \wedge m_1 \neq m_2\end{bmatrix} \leq \varepsilon.$$

However, any user in possession of the trapdoor tk is able to find a collision using Coll. Additionally, Chameleon Hash functions have the uniformity property, which means the hash value leaks nothing about the message input. Formally, for all pair of messages m_1 and m_2 and the randomly chosen r, the probability distributions of the random variables $\mathsf{CH}(\mathsf{ck}, m_1, r)$ and $\mathsf{CH}(\mathsf{ck}, m_2, r)$ are computationally indistinguishable.

We need here the hash value to be verifiable, so that we add two $\mathsf{VKeyGen}$ and Valid algorithms (executed by the receiver) and we modify the existing algorithms as follows:

- $\mathsf{VKeyGen}(\mathsf{ck})$: Outputs the chameleon designated verification key vk and the trapdoor vtk. This trapdoor can be empty or public if the chameleon hash is publicly verifiable.
- $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m; r)$: Picks a random r, and outputs the chameleon hash a as well as the *witness* d, *i.e.* the corresponding data needed to verify a.
- $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk})$: Allows to check that the sender knows how to open a Chameleon Hash a to a specific value m for the witness d. The verification can be public if vtk is empty or public, or specific to the receiver otherwise.
- $\mathsf{Coll}(\mathsf{ck}, \mathsf{vk}, m, r, m', \mathsf{tk})$: Takes as input the public keys, the trapdoor tk, a start message m and randomness r and a target message m' and outputs a target randomness r' such that if $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m; r) = (a, d)$, then $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m'; r') = (a, d')$.

Once again, we expect the chameleon hash to be collision resistant on the first part of the output, which means it is infeasible to find $(m_1, r_1), (m_2, r_2)$ such that $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m_1, r_1) = (a, d_1)$ and $\mathsf{CH}(\mathsf{ck}, m_2, r_2) = (a, d_2)$ and $m_1 \neq m_2$ given only the Chameleon public keys ck and vk.

We expect the verification to be sound, which means that, given a tuple (m, a, d) satisfying $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk})$, there always exists at least one tuple (r, d') such that $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m; r) = (a, d')$.

Protocols in the UC Framework. The goal of the UC framework 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 $\mathit{\Pi}$ emulates $\mathcal{F}$, one has to construct, for any polynomial adversary $\mathcal{A}$ (which controls the communication between the players), a simulator $\mathcal{S}$ such that no polynomial environment $\mathcal{Z}$ (the distinguisher) can distinguish between the real world (with the real players interacting with themselves and $\mathcal{A}$ and executing the protocol π) and the ideal world (with dummy players interacting with $\mathcal{S}$ 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, $\mathcal{A}$ has complete access to the internal state and private values of the player, takes its entire control, and plays on its behalf.

UC-Secure Oblivious Transfer. The ideal functionality of an Oblivious Transfer (OT) protocol is depicted in Figure 1. It is inspired from [CKWZ13, ABB$^+$13].

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

- **Upon receiving an input $(\mathsf{Send}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, (m_1, \ldots, m_k))$ from party P_i,** with $m_i \in \{0,1\}^{\mathfrak{K}}$: record the tuple $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, (m_1, \ldots, m_k))$ and reveal $(\mathsf{Send}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ to the adversary $\mathcal{S}$. Ignore further Send-message with the same ssid from P_i.
- **Upon receiving an input $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, s)$ from party P_j,** with $s \in \{1, \ldots, k\}$: record the tuple $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, s)$, and reveal $(\mathtt{Receive}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ to the adversary $\mathcal{S}$. Ignore further Receive-message with the same ssid from P_j.
- **Upon receiving a message $(\mathtt{Sent}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ from the adversary $\mathcal{S}$**: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, (m_1, \ldots, m_k))$ or $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, s)$ is not recorded; otherwise send $(\mathtt{Sent}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ to P_i and ignore further Sent-message with the same ssid from the adversary.
- **Upon receiving a message $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j)$ from the adversary $\mathcal{S}$**: ignore the message if $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, (m_1, \ldots, m_k))$ or $(\mathsf{sid}, \mathsf{ssid}, P_i, P_j, s)$ is not recorded; otherwise send $(\mathtt{Received}, \mathsf{sid}, \mathsf{ssid}, P_i, P_j, m_s)$ to P_j and ignore further Received-message with the same ssid from the adversary.

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

3 Generic Construction of UC-Secure Oblivious Transfer

In this section, we show how to construct in a generic way a UC-secure oblivious transfer from any collision-resistant chameleon hash and CCA-2 encryption scheme.

In [ABB$^+$13], the authors give a way to construct such a UC-secure oblivious transfer protocol from an SPHF-friendly commitment, but they only give an instantiation of such an SPHF-friendly commitment in a DDH-based setting, using Haralambiev commitment scheme [Har11] and Cramer-Shoup encryption scheme [CS02].

Our goal is thus to strengthen the generic part of the construction, by showing how to construct, in a generic way, a UC-secure SPHF-friendly commitment scheme in any setting, from a collision-resistant chameleon hash and a CCA-2 encryption scheme.

3.1 From Commitment to Oblivious Transfer

Introduction. In an oblivious transfer scheme, we consider the interaction between a server, possessing a database called DB containing $t = 2^m$ lines, and a user, willing to request the line j of the database in an oblivious way. Informally, this implies that the user will gain no information about the other lines of the database, and also that the server will obtain no information about the specific line the user wants to obtain.

In the protocol described in [ABB$^+$13], from a high point of view[3], the user sends to the server a commitment of the number j of the line it is willing to obtain. The server then computes a pair of keys for a smooth projective hash function (SPHF) adapted to the commitment. It keeps secret the hash key and sends the projection key to the user,

[3] Note that we omit here for the sake of simplicity the creation of a secure channel between the user and the server (this is only needed in the adaptive version of the protocol).

along with the hash value of all the lines of the database. Thanks to the properties of the SPHF, the user will then be able to recover the particular line it wants, using the public projection key and the secret random coins it used to create its committed value in the first place. The properties of the SPHF also ensure that the server has no idea about the line the user is requiring, and that the user cannot obtain any information from the hash values of the other lines of DB, which are exactly the requirements of a secure OT.

The authors of this protocol prove its security in the UC framework, which implies the use of a commitment with strong security properties. Indeed, the simulator of a user needs to be able to change its mind about the line required, hence an *equivocable* commitment; and the simulator of a server also needs to be able to extract the line required by the user, hence an *extractable* commitment. Unfortunately, combining both equivocability and extractability on the same commitment scheme, especially if we require this commitment scheme to admit an SPHF, is a difficult task and requires more security properties, as we recall in the following.

Properties for Commitments. We informally recall these specific properties, defined in [ABB$^+$13] and formally stated in the full version. We call a commitment scheme $\mathtt{E}^2$ (for *extractable and equivocable* and the necessary properties) if the indistinguishable setup algorithm outputs a common trapdoor that allows both equivocability and extractability, and the two following properties are satisfied: *strong simulation indistinguishability* (one cannot distinguish a real commitment (generated by Com) from a fake commitment (generated by SCom), even with oracle access to the extraction oracle (ExtCom) and to fake commitments (using SCom)) and *strong binding extractability* (one cannot fool the extractor, *i.e.*, produce a commitment and a valid opening data (not given by SCom) to an input x while the commitment does not extract to x, even with oracle access to the extraction oracle (ExtCom) and to fake commitments (using SCom)).

A commitment is said to be *robust* if one cannot produce a commitment and a label that extracts to x' (possibly $x' = \perp$) such that there exists a valid opening data to a different input x, even with oracle access to the extraction oracle (ExtCom) and to fake commitments (using SCom).

Finally, a commitment is said to be SPHF-*friendly* if it is an $\mathtt{E}^2$ commitment that admits an SPHF on the languages $L_x = \{(\ell, C) | \exists \delta, \mathsf{VerCom}^\ell(C, x, \delta) = 1\}$, and that is both strongly-simulation-indistinguishable and robust.

3.2 Generic Construction of SPHF-Friendly Commitment

Introduction. We start by a high-level description of the (Cramer-Shoup-based) commitment given in [ABB$^+$13] in the pairing-friendly setting $(\mathcal{G}_1, g_1, h_1, \mathcal{G}_2, g_2, \mathcal{G}_T, p, e)$. They set $T = {g_2}^t$, t being a value chosen at random in $\mathcal{Z}_p$. We omit the labels for the sake of simplicity. First, they cut the message M to be committed into bits, denoted here as $\boldsymbol{M} = (M_i)_i \in \{0, 1\}^m$. They then compute a TC4 Haralambiev [Har11] equivocable commitment of each bit M_i: $\boldsymbol{a} = (a_i)_i$ with $a_i = {g_2}^{r_{i,M_i}} T^{M_i}$ with r_{i,M_i} chosen at random in $\mathcal{Z}_p$ and $r_{i,\overline{M_i}} = 0$. The opening values (for decommitment) are the values $d_{i,j} = {g_1}^{r_{i,j}}$. They then compute a multi-Cramer-Shoup encryption $\boldsymbol{b} = (b_{i,j})_{i,j}$ of $\boldsymbol{d} = (d_{i,j})_{i,j}$ with randomness $\boldsymbol{s} = (s_{i,j})_{i,j}$. The commitment is $(\boldsymbol{a}, \boldsymbol{b})$, the opening information being $\boldsymbol{s}$. To open the commitment, the receiver checks the validity of the ciphertexts b_{i,M_i}, extracts each value d_{i,M_i} from b_{i,M_i} and s_{i,M_i} and finally checks whether the equality $e(g_1, a_i/T^{M_i}) = e(d_{i,M_i}, g_2)$ holds.

The equivocability of the commitment is ensured by the knowledge of t, enabling the sender to set $r_{i,\overline{M_i}} = r_{i,M_i} \pm t$ rather than $r_{i,\overline{M_i}} = 0$. The extractability is ensured by the knowledge of the decryption keys of the Cramer-Shoup encryption.

Our first goal, in this concrete instantiation, is to get rid of the pairing setting, and in particular of the pairing verification, in order to be able to propose constructions in other settings. To this aim, we change the TC4 commitment of M_i for a *verifiable* chameleon hash of M_i. Making this change enables us to get a generic version of this commitment, requiring only "compatible" chameleon hash (playing the role of the TC4 scheme above) and CCA encryption schemes (playing the role of the Cramer-Shoup above). The chameleon hash can either be publicly verifiable (which gives us a non-interactive commitment), or verifiable by the receiver, which requires a pre-flow, in which the server generates a verification key and its trapdoor and sends the verification key to the sender.

Building Blocks. We assume the existence of compatible CCA-encryption $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ and chameleon hash $(\mathsf{KeyGen}, \mathsf{VKeyGen}, \mathsf{CH}, \mathsf{Coll}, \mathsf{Valid})$, in the sense that is feasible to compute a CCA-encryption of the opening value of the chameleon hash. For example, a Pedersen Chameleon Hash is not compatible with Cramer Shoup encryption, as we would need to encrypt the randomness as a scalar, while the decryption algorithm only allows us to recover group elements.

In order for our commitment to accept an SPHF, we require the CCA-encryption to accept an SPHF on the language of valid ciphertexts. The precise language needed will depend on the way the chameleon hash is verified, but will be easily constructed by combining several simple languages as described in [BBC$^+$13a].

We require the chameleon hash to be verifiable by the receiver. For the sake of concision, we describe here the case where the chameleon hash is only verifiable by the server. In this case, we need a pre-flow, in which the server is assumed to execute the algorithm VKeyGen to generate a verification key and its trapdoor and send the verification key to the sender. This makes the commitment not completely non-interactive anymore but it should be noted that if the global protocol is not one-round, these values can be sent by the receiver during the first round of the protocol. In the case where the chameleon hash is publicly verifiable, one simply has to consider the keys vk and vtk empty, and ignore the pre-flow.

Construction. We now describe the different algorithms of our chameleon-hashed targeted commitment protocol CHCS from player P to Q (see Section 2 for the notations of the algorithms).

- **Setup and simulated setup algorithms:** $\mathsf{SetupComT}(1^{\mathfrak{K}})$ (the algorithm for setup with trapdoors) generates the various parameters param, for the setting of the SPHF-friendly labelled CCA-encryption scheme and the chameleon hash scheme. It then generates the corresponding keys and trapdoors: $(\mathsf{ck}, \mathsf{tk})$ for the chameleon hash scheme and $(\mathsf{ek}, \mathsf{dk})$ for the encryption scheme.
  For $\mathsf{SetupCom}(1^{\mathfrak{K}})$ (the algorithm for setup without trapdoors), the setting and the keys are generated the same way, but forgetting the way the keys were constructed (such as the scalars, in a DDH-based setting), thus without any trapdoor.

The algorithms both output the CRS $\rho = (\mathsf{ek}, \mathsf{ck}, \mathsf{param})$. In the first case, τ denotes the trapdoors $(\mathsf{dk}, \mathsf{tk})$.

- **Pre-flow (verification key generation algorithm):** player Q executes $\mathsf{VKeyGen}(\mathsf{ck})$ to generate the chameleon designated verification key vk and the trapdoor vtk and sends vk to the sender P.
- **Targeted commitment algorithm:** $\mathsf{Com}^{\ell}(\boldsymbol{M}; Q)$ from player P to player Q, for $\boldsymbol{M} = (M_i)_i \in \{0,1\}^m$ and a label ℓ, works as follows:
  - For $i \in [\![1, m]\!]$, it chooses r_{i,M_i} at random and computes $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, M_i; r_{i,M_i})$ to obtain the hash value a_i and the corresponding opening value d_{i,M_i}. It samples at random the values $r_{i,1-M_i}$ and $d_{i,1-M_i}$. We denote as $\boldsymbol{a} = (a_1, \ldots, a_m)$ the tuple of commitments and $\boldsymbol{d} = (d_{i,j})_{i,j}$.
  - For $i \in [\![1, m]\!]$ and $j = 0, 1$, it gets $\boldsymbol{b} = (b_{i,j})_{i,j} = \mathsf{2mEncrypt}^{\ell'}_{\mathsf{pk}}(\boldsymbol{d}; \boldsymbol{s})$, where $\boldsymbol{s}$ is taken at random and $\ell' = (\ell, \boldsymbol{a})$.

  The commitment is $C = (\boldsymbol{a}, \boldsymbol{b})$, and the opening information is the m-tuple $\delta = (s_{1,M_1}, \ldots, s_{m,M_m})$.
- **Verification algorithm:** $\mathsf{VerCom}^{\ell}(\mathsf{vtk}, C, \boldsymbol{M}, \delta)$ first checks the validity of the ciphertexts b_{i,M_i} with randomness s_{i,M_i}, then extracts d_{i,M_i} from b_{i,M_i} and s_{i,M_i}, and finally checks the chameleon hash a_i with opening value d_{i,M_i}, for $i \in [\![1, m]\!]$, via the algorithm $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, M_i, a_i, d_{i,M_i}, \mathsf{vtk})$.
- **Simulated targeted commitment algorithm:** $\mathsf{SimCom}^{\ell}(\tau; Q)$ from the simulator to player Q, takes as input the equivocation trapdoor, namely tk, from $\tau = (\mathsf{dk}, \mathsf{tk})$, and outputs the commitment $C = (\boldsymbol{a}, \boldsymbol{b})$ and equivocation key $\mathsf{eqk} = \boldsymbol{s}$, where
  - For $i \in [\![1, m]\!]$, it chooses $r_{i,0}$ at random, computes $(a_i, d_{i,0}) = \mathsf{CH}(\mathsf{ck}, \mathsf{vk}, 0; r_{i,0})$, and uses the equivocation trapdoor tk to compute $r_{i,1}$ used to open the chameleon hash to 1 such that $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, 1; r_{i,1})$ is equal to $(a_i, d_{i,1})$. This leads to $\boldsymbol{a}$ and $\boldsymbol{d}$, making $d_{i,j}$ the opening value for $a_{i,j}$ for all $i \in [\![1, m]\!]$ and $j = 0, 1$.
  - $\boldsymbol{b}$ is built as above: $\boldsymbol{b} = (b_{i,j})_{i,j} = \mathsf{2mEncrypt}^{\ell'}_{\mathsf{pk}}(\boldsymbol{d}; \boldsymbol{s})$, where $\mathsf{eqk} = \boldsymbol{s}$ is taken at random and $\ell' = (\ell, \boldsymbol{a})$.
- **Equivocation algorithm:** $\mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, \boldsymbol{M})$ simply uses part of the equivocation key eqk (computed by the SimCom algorithm) to obtain the opening information $\delta = (s_{1,M_1}, \ldots, s_{m,M_m})$ in order to open to $\boldsymbol{M} = (M_i)_i$.
- **Extraction algorithm:** $\mathsf{ExtCom}^{\ell}(\tau, \mathsf{vtk}, C)$ takes as input the extraction trapdoor, namely the decryption key dk, from $\tau = (\mathsf{dk}, \mathsf{tk})$, the verification trapdoor vtk and a commitment $C = (\boldsymbol{a}, \boldsymbol{b})$. For $i \in [\![1, m]\!]$ and $j = 0, 1$, it first extracts the value $d_{i,j}$ from the ciphertext $b_{i,j}$, using the decryption key dk. Then, for $i \in [\![1, m]\!]$, it checks the chameleon hash a_i with opening values $d_{i,0}$ and $d_{i,1}$ with the help of the algorithm $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, j, a_i, d_{i,j}, \mathsf{vtk})$ for $j = 0, 1$. If only one opening value $d_{i,j}$ satisfies the verification equality of the chameleon hash, then $j = M_i$. If this condition holds for each $i \in [\![1, m]\!]$, then the extraction algorithm outputs $(M_i)_i$. Otherwise (either if $\boldsymbol{b}$ could not be correctly decrypted, or there was an ambiguity while checking $\boldsymbol{a}$, with at least one chameleon hash a_i with two possible opening values $d_{i,0}$ and $d_{i,1}$), it outputs $\perp$.

Security Result. Given a publicly verifiable collision-resistant chameleon hash and a secure CCA-encryption accepting an SPHF on the language of valid ciphertexts, the above construction provides a commitment scheme which is SPHF-friendly.

Proof. According to the results recalled at the beginning of this section, page 10, we first need to prove that this $\mathtt{E}^2$ commitment is *strongly-simulation-indistinguishable* and *robust*. Due to lack of space, the proof of this result is postponed to the full version.

One then additionally needs to construct an SPHF on the languages $\mathrm{L}_M = \{(\ell, C) | \exists \delta$ such that $\mathsf{VerCom}^\ell(\mathsf{vtk}, C, M, \delta) = 1\}$. Recall that the CCA-encryption admits an SPHF on the languages $\mathrm{L}_M^{\mathrm{enc}} = \{(\ell, C) | \exists r$ such that $\mathsf{Encrypt}^\ell(\mathsf{pk}, M; r)) = C\}$, directly giving us the required SPHF since the algorithm VerCom, on input $C = (\boldsymbol{a}, \boldsymbol{b})$, first checks the CCA-encryptions b_{i,M_i} and then verifies the chameleon hashes a_i for all i. More precisely, the required language is as follows: $\mathrm{L}_M = \{(\ell, C) | \forall i \in \{1, \ldots, m\}$ $\exists r_{i,M_i}, s_{i,M_i}, d_{i,M_i}$ such that $\mathsf{mEncrypt}^{*,\ell}(\mathsf{pk}, (d_{i,M_i})_i; (s_{i,M_i})_i) = (b_{i,M_i})_i$ and that $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, M_i; r_{i,M_i}) = (a_i, d_{i,M_i})\}$, on which one can easily construct an SPHF by disjunction using the method described in [ACP09, BBC+13a][4].

3.3 Generic Construction of UC-Secure Oblivious Transfer

Introduction. We denote by DB the database of the server containing $t = 2^m$ lines, and j the line requested by the user in an oblivious way. We 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 and a IND-CPA encryption scheme $\mathcal{E} = (\mathsf{Setup}_{\mathrm{cpa}}, \mathsf{KeyGen}_{\mathrm{cpa}}, \mathsf{Encrypt}_{\mathrm{cpa}}, \mathsf{Decrypt}_{\mathrm{cpa}})$ with plaintext size at least equal to the security parameter. We also assume the existence of compatible CCA-encryption and chameleon hash with the properties described in the former section, and we generically obtain from them the SPHF-friendly commitment scheme given above.

Protocol. We exactly follow the construction given in [ABB+13], giving the protocol presented on Figure 2. The only difference is that we take advantage of the pre-flow to ask the server to generate the CH verification keys $(\mathsf{vk}, \mathsf{vtk})$. For the sake of simplicity, we only give the version for adaptive security, in which the server generates pk and c to create a somewhat secure channel (they would not be used in the static version).

Security Result. The oblivious transfer scheme described in Figure 2 is UC-secure in the presence of adaptive adversaries, assuming reliable erasures and authenticated channels, as soon as the commitment scheme is constructed from a secure publicly-verifiable chameleon hash and a secure CCA encryption scheme admitting an SPHF on the language of valid ciphertexts, as described in the former section.

The proof remains the same; It is given in the full version for completeness.

[4] The notation $\mathsf{mEncrypt}^{*,\ell}(\mathsf{pk}, (d_{i,M_i})_i; (s_{i,M_i})_i)$ simply means that we compute $2\mathsf{mEncrypt}^\ell(\mathsf{pk}, (d_{i,j})_{i,j}; (s_{i,j})_{i,j})$ and take the m components corresponding to $j = M_i$ for every i.

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

Pre-flow:

1. Server 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 KeyGen
2. Server generates a verification key pair $(\mathsf{vk}, \mathsf{vtk}) \xleftarrow{\$} \mathsf{VKeyGen}(\mathsf{ck})$ for CH, stores vtk and completely erases the random coins used by $\mathsf{VKeyGen}$
3. Server sends pk and vk to User

Index query on j:

1. User 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. User computes $(C, \delta) \xleftarrow{\$} \mathsf{Com}^{\ell}(j)$ with $\ell = (\mathsf{sid}, \mathsf{ssid}, P_i, P_j)$
3. User stores δ and completely erases J, R and the random coins used by Com and $\mathsf{Encrypt}_{\mathrm{cpa}}$ and sends C and c to Server

Database input $(n_1, \ldots, n_t)$**:**

1. Server decrypts $J \leftarrow \mathsf{Decrypt}_{\mathrm{cpa}}(\mathsf{sk}, c)$ and then $R \leftarrow F(J)$
2. For $s = 1, \ldots, t$: Server computes $\mathsf{hk}_s \xleftarrow{\$} \mathsf{HashKG}(L_s, \mathsf{param})$, $\mathsf{hp}_s \leftarrow \mathsf{ProjKG}(\mathsf{hk}_s, (L_s, \mathsf{param}), (\ell, C))$, $K_s \leftarrow \mathsf{Hash}(\mathsf{hk}_s, (L_s, \mathsf{param}), (\ell, C))$, and $N_s \leftarrow R \oplus K_s \oplus n_s$
3. Server erases everything except $(\mathsf{hp}_s, N_s)_{s=1,\ldots,t}$ and sends them over a secure channel

Data recovery:
Upon receiving $(\mathsf{hp}_s, N_s)_{s=1,\ldots,t}$, User computes
$K_j \leftarrow \mathsf{ProjHash}(\mathsf{hp}_j, (L_j, \mathsf{param}), (\ell, C), \delta)$ and gets $n_j \leftarrow R \oplus K_j \oplus N_j$.

Fig. 2. UC-Secure 1-out-of-t OT from an SPHF-Friendly Commitment (for Adaptive Security)

4 Instantiation Based on Cramer-Shoup Encryption (DDH)

Let us now show how to build SPHF-friendly commitment schemes from various assumptions. While it may seem to be a tremendously far-fetched idea for a construction, we are going to show throughout the following sections that in fact such schemes can be easily built on any of the main modern fields of cryptographic hypotheses.

We start with the construction based on DDH: Since it is easier to understand, it will help to underline the key points. This commitment revisits the one used in [ABB$^+$13] but we remove the pairing used in it thanks to the methods described in the previous section, by generating vtk on the fly. For the chameleon hash, we are going to use a CDH-based Pedersen encryption scheme. However as such CH is not designated verifier, we are going to transform it in an Haralambiev way [Har11, Section 4.1.4]. For the CCA encryption we will rely on an extended version of Cramer-Shoup encryption.

4.1 Building Blocks

CDH-based Chameleon Hash[5]

- $\mathsf{KeyGen}(\mathfrak{K})$: Outputs the chameleon hash key $\mathsf{ck} = (g, h)$ and the trapdoor $\mathsf{tk} = \alpha$, where $g^\alpha = h$;
- $\mathsf{VKeyGen}(\mathsf{ck})$: Generates $\mathsf{vk} = f$ and $\mathsf{vtk} = \log_g(f)$
- $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, m; r)$: Picks a random $r \in \mathbb{Z}_p$, and outputs the chameleon hash $a = h^r g^m$. Sets $d = f^r$.
- $\mathsf{Coll}(m, r, m', \mathsf{tk})$: outputs $r' = r + (m - m')/\alpha$.
- $\mathsf{Valid}(\mathsf{ck}, \mathsf{vk}, m, a, d, \mathsf{vtk})$: The user outputs d, so that one can check if $a = h^m \cdot d^{1/\mathsf{vtk}}$.

The trivial way to check this CH requires a pairing instead of knowing vtk. Note that this trivial verification indeed leads to the protocol described in [ABB+13]. Instead, we let the verifier (the server in latter use) picks a new f and its discrete logarithm.

$2m$-labelled multi twisted Cramer-Shoup Encryption Scheme
We first recall the Cramer-Shoup encryption scheme, which is $\mathsf{IND\text{-}CCA}$ under the DDH assumption.

- $\mathsf{KeyGen}(\mathfrak{K})$: Assuming two independent generators g and h, for random scalars $x_1, x_2, y_1, y_2, z \xleftarrow{\$} \mathbb{Z}_p$, we set $\mathsf{sk} = (x_1, x_2, y_1, y_2, z)$ to be the private decryption key and $\mathsf{ek} = (g_1, g_2, c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h_1 = g_1^z, \mathcal{H})$ to be the public encryption key, where $\mathcal{H}$ is a random collision-resistant hash function from $\mathcal{H}$.
- If $M \in \mathbb{G}$, the Cramer-Shoup encryption is defined as $\mathsf{CS}^\ell(\mathsf{pk}, M; r) = (u = g_1^r, v = g_2^r, e = h^r \cdot M, w = (cd^\theta)^r)$, where $\theta = H(\ell, u, v, e)$.
- Such a ciphertext is decrypted by $M = e/u^z$, after having checked the validity of the ciphertext: $w \stackrel{?}{=} u^{x_1 + \theta y_1} v^{x_2 + \theta y_2}$.

The above scheme can be extended naturally to encrypt vectors of group elements $\boldsymbol{D} = (D_1, \ldots, D_{2m}) \in \mathbb{G}^{2m}$, by having $2m$ tuples of random scalars in the secret key, and a global value θ for the encryption. The authors of [ABB+13] proved that this scheme is `VIND-PO-CCA` under the DDH assumption.

4.2 Diffie-Hellman Based Commitment Scheme

We simply apply the construction described in Section 3 to obtain the commitment scheme from these blocks.

- $\mathsf{SetupComT}(1^{\mathfrak{K}})$ generates a multiplicative group $\mathsf{param} = (p, \mathbb{G}, g)$;
  $\mathsf{ek} = (g_1, g_2, c, d, h_1, \mathcal{H})$ and the decryption key dk corresponding to the various discrete log in basis g, $\mathsf{ck} = (g, h)$, tk the respective discrete logarithm.
  For $\mathsf{SetupCom}(1^{\mathfrak{K}})$, the CRS is generated the same way, but forgetting the scalars, and thus without any trapdoor.
  The algorithms both output $\rho = (\mathsf{ek}, \mathsf{ck}, \mathsf{param})$.

[5] As there is no pairing in our construction, we do not really need the linear based version of both schemes, but similar variants can be imagined based on the linear assumption or even on any matrix assumption [EHK+13].

- Pre-flow: During the preflow, the server Q runs $\mathsf{VKeyGen}(\mathsf{ck})$ and outputs $\mathsf{vk} = f$ and keeps its discrete logarithm vtk.
- $\mathsf{Com}^{\ell}(\boldsymbol{M}; Q)$ from player P to player Q, for $\boldsymbol{M} = (M_i)_i \in \{0,1\}^m$ and a label ℓ, works as follows:
  - For $i \in [\![1, m]\!]$, it chooses a random $r_{i,M_i} \in \mathbb{Z}_p$, a random $r_{i,1-M_i}$, and computes $a_i = g^{M_i} h^{r_{i,M_i}}$ and sets $d_{i,j} = f^{r_{i,j}}$ for $j = 0, 1$, which makes d_{i,M_i} part of the opening value for a_i to M_i. Let us write $\boldsymbol{a} = (a_1, \ldots, a_m)$, the tuple of commitments.
  - For $i \in [\![1, m]\!]$ and $j = 0, 1$, it gets $\boldsymbol{b} = (b_{i,j})_{i,j} = \mathsf{2mEncrypt}^{\ell'}(\mathsf{pk}, \boldsymbol{d}; \boldsymbol{s})$, where $\boldsymbol{s}$ is from the random string and $\ell' = (\ell, \boldsymbol{a})$.

  The commitment is $C = (\boldsymbol{a}, \boldsymbol{b})$, and the opening information is the m-tuple $\delta = (s_{M_1}, \ldots, s_{M_m})$.
- $\mathsf{VerCom}^{\ell}(C, \boldsymbol{M}, \delta)$ checks the validity of the ciphertexts b_{i,M_i} with s_{M_i}, extracts d_{i,M_i} from b_{i,M_i} and s_{i,M_i}, and checks whether $(a_i/g^{M_i})^{\mathsf{vtk}} = d_{i,M_i}$.
- $\mathsf{SimCom}^{\ell}(\tau)$ takes as input the equivocation trapdoor, namely tk, and outputs $C = (\boldsymbol{a}, \boldsymbol{b})$ and $\mathsf{eqk} = \boldsymbol{s}$, where
  - For $i \in [\![1, m]\!]$, it chooses a random $r_{i,0}$, sets $a_i = g^{r_{i,0}}$, and uses the equivocation trapdoor to computes the randomness $r_{i,1} = r_{i,0} - 1/\mathsf{tk}$. This leads to $\boldsymbol{a}$ and $\boldsymbol{d}$;
  - $\boldsymbol{b}$ is built as above: $\boldsymbol{b} = (b_{i,j})_{i,j} = \mathsf{2mEncrypt}^{\ell'}(\mathsf{pk}, \boldsymbol{d}; \boldsymbol{s})$, with random scalars $\mathsf{eqk} = (s_{*,i,j})_{i,j}$.
- $\mathsf{OpenCom}^{\ell}(\mathsf{eqk}, C, \boldsymbol{M})$ simply uses eqk to set the opening value $\delta = (s_{M_1}, \ldots, s_{M_m})$ in order to open to $\boldsymbol{M} = (M_i)_i$.
- $\mathsf{ExtCom}^{\ell}(\tau, C)$ takes as input the extraction trapdoor, namely the decryption key dk and the chameleon verification trapdoor vtk. Given $\boldsymbol{b}$, it can decrypt all the $b_{i,j}$ into $d_{i,j}$ and checks consistency with $(a_i/g^j)^{\mathsf{vtk}} \overset{?}{=} d_{i,j}$ or not. If, for each i, exactly one $j = M_i$ satisfies the equality, then the extraction algorithm outputs $(M_i)_i$, otherwise (no correct decryption or ambiguity with several possibilities) it outputs $\perp$.

4.3 The SPHF Associated with the Commitment Scheme

For the sake of simplicity, we first give an explicit writing of the said SPHF when the strings are of length one.

This SPHF is defined on Cramer-Shoup encryption (see for instance [BBC⁺13b]), except that it is done on an encryption of "an encryption of M, such that the projected hash value of this encryption is the value sent in the commitment of M", rather than simply on an encryption of M. But the internal language is easily verifiable, making this SPHF having the good properties simply applying the methodology described in [BBC⁺13b].

- $\mathsf{Com}^{\ell}(b; Q)$: A commitment to a bit m_i, can now be written as $C = h^{r_{m_i}} g^{m_i}, b_{1,0} = (h_1^{s_0} g^{r_0}, g_1^{s_0}, g_2^{s_0}, (cd^{\beta})^{s_0}), b_{1,1} = (h_1^{s_1} g^{r_1}, g_1^{s_1}, g_2^{s_1}, (cd^{\beta})^{s_1})$.
  where $\beta = \mathcal{H}(h^{r_b} g^{m_i}, (h_1^{s_j} g^{r_j}, g_1^{s_j}, g_2^{s_j})_{j \in [\![0,1]\!]})$ and the session id.
- $\mathsf{VerCom}^{\ell}(C, b, \delta)$:

- $\mathsf{ProjKG}(C, b; Q)$: To implicitly check if the commitment is a valid commitment to b, one simply has to compute projection keys $\mathsf{hp} = h^\lambda f^\mu, \mathsf{hp}_{m_i} = h_1^\mu g_1^{\mu_{m_i}} g_2^{\nu_{m_i}} (cd^\beta)^{\theta_{m_i}}$, where all new Greek letters are random scalars. And the hash value $H_{m_i} = (C/g^{m_i})^\lambda \cdot \boldsymbol{b}_{m_i}^{\mathsf{hk}_{m_i}}$.)
- $\mathsf{ProjHash}(C, b, \mathsf{hp}_{m_i}; P)$: The prover will compute $H'_{m_i} = \mathsf{hp}_{m_i}^{s_{m_i}} \mathsf{hp}^{r_{m_i}}$.

If everything was done honestly, those two values are equal, otherwise they are seemingly random. To see why this is smooth, considering the number of free variables in the system of equations generated by the public view of the projection key hp guarantees that not enough information leaks about the hashing keys in order to weaken the smoothness.

In the real protocol where the string is cut into bits, one simply has to do an AND of all those languages, where $H = \prod H_{i,m_i}$, and where one uses a vector of projections keys hp_{i,m_i}. To optimize the construction on bit strings, one can simply use the polynomial trick from [BBC+13a], where they provide hp_1, a random scalar ϵ and assume that $\mathsf{hp}_i = \mathsf{hp}_1^{\epsilon(i-i)}$, a classical inversion argument on the matrices of discrete logarithm of the given exponents will show that the SPHF remains smooth.

Efficiency consideration shows that the pre-flow requires 2 group elements (1 for pk, 1 for vk), for each bit we need 9 elements (1 for a_i and 2*4 for $b_{i,\{0,1\}}$, we also have the additional encryption for the verification linked to the pre-flow (so 2 elements). We now need to give two elements for the hp, and in case of more that one bit, a random scalar ϵ. Overall this leads to $9m + 6$ group elements and a scalar.

5 Instantiation Based on Dual Regev Encryption (LWE)

Lattices present an interesting challenge, since because of the noise many properties are harder to achieve. However, our construction requires only two simple blocks to work.

Chameleon Hash

We present here a Chameleon Hash constructed from the SIS assumption, following the chameleon hash given in [CHKP10] but using the Micciancio-Peikert trapdoor generation [MP12]. We here only present the scheme, since the security proof comes directly following the proof of Lemma 4.1 in [CHKP10].

Let $k = \lceil \log q \rceil = O(\log \mathfrak{K})$ and $m = O(\mathfrak{K}k)$. Let $\mathcal{D} = D_{\mathbb{Z}^{\bar{m} \times \mathfrak{K}k}, \omega(\sqrt{\log \mathfrak{K}})}$ be the Gaussian distribution over $\mathbb{Z}^{\bar{m} \times \mathfrak{K}k}$ with parameter $\omega(\sqrt{\log \mathfrak{K}})$ and let $s = O(\sqrt{\mathfrak{K}k})$ be a Gaussian parameter. Let the randomness space be defined as $\mathcal{R} = D_{\mathbb{Z}^m, s \cdot \omega(\sqrt{\log \mathfrak{K}})}$. Then, the Chameleon Hash is defined as follows:

– $\mathsf{KeyGen}(\mathfrak{K})$: choose a random matrix $\mathbf{A}_0 \xleftarrow{\$} \mathbb{Z}_q^{\mathfrak{K} \times \ell}$.
  Sample $(\mathbf{A}_1, \mathbf{R}_1) \xleftarrow{\$} \mathsf{GenTrap}^{\mathcal{D}}(1^{\mathfrak{K}}, 1^m, q)$. Define $\mathsf{ck} = (\mathbf{A}_0, \mathbf{A}_1)$ and $\mathsf{tk} = \mathbf{R}_1$.
– $\mathsf{VKeyGen}(\mathsf{ck})$: Outputs $\mathsf{vk} = \perp$, $\mathsf{vtk} = \perp$
– $\mathsf{CH}(\mathsf{ck}, \mathsf{vk}, \mathbf{m}; \mathbf{r})$: choose a vector $\mathbf{r}$ from the Gaussian distribution $D_{\mathbb{Z}^m, s \cdot \omega(\sqrt{\log \mathfrak{K}})}$, $\mathbf{r} \leftarrow D_{\mathbb{Z}^m, s \cdot \omega(\sqrt{\log \mathfrak{K}})}$. Compute the chameleon hash value $\mathbf{c} = \mathbf{A}_0\mathbf{m} + \mathbf{A}_1\mathbf{r}$. Return the chameleon hash $\mathbf{c}$ and the opening information $\mathbf{r}$. (which we will later commit using the CCA2 scheme)

– $\mathsf{Coll}(\mathsf{tk}, (\mathbf{m}_0, \mathbf{r}_0), \mathbf{m}_1)$: compute $\mathbf{u} = (\mathbf{A}_0\mathbf{m}_0 + \mathbf{A}_1\mathbf{r}_0) - \mathbf{A}_0\mathbf{m}_1$ and sample $\mathbf{r}_1 \in \mathbb{Z}^m$ according to $D_{\Lambda^{\perp}_{\mathbf{u}}(\mathbf{A}_1), s\cdot\omega(\sqrt{\log \mathfrak{K}})}$, $\mathbf{r}_1 \xleftarrow{\$} \mathsf{SampleD}(\mathbf{R}_1, \mathbf{A}_1, \mathbf{u}, s)$.
– $\mathsf{Verify}(\mathsf{ck}, \mathsf{vtk}, \mathbf{m}, \mathbf{c}, \mathbf{r})$: accept if $\|\mathbf{r}\| \le s \cdot \omega(\sqrt{\log \mathfrak{K}}) \cdot \sqrt{m}$ and $\mathbf{c} = \mathbf{A}_0\mathbf{m} + \mathbf{A}_1\mathbf{r}$; otherwise, reject.

It should be noted, that the trapdoor allows to recover not only a collision, but also a preimage if need be.

Naive $2m$-labelled multi LWE-based Encryption Scheme
Katz and Vaikuntanathan proposed in [KV09] a labelled CCA-Encryption with an approximate SPHF. In order to achieve the $2m$-labelled, one simply has to use the same label in all the encryptions, and then add a one-time signature, built for example by using the previous chameleon hash.

Oblivious Transfer using an Approximate SPHF
The approximate SPHF presented in [KV09] is sufficient for our application with a small modification to our generic framework. Indeed, instead of obtaining two identical values for Hash and ProjHash, the correctness only guarantees that for a well-formed ciphertext, those two values have a small Hamming distance, hence xoring the two values together leads to a string with low Hamming weight. Assuming the line in the database is first encoded using an Error Correcting Code, and then masked by the server using the Hash value, the user can then use his projective hash value to recover a word near a valid encoding for the required entry, and then decoding using the Error Correcting Code as the remaining nose is small, he will recover the valid string. On invalid lines, the noise is seemingly random, hence beyond the decoding limit of any possible code.

6 Instantiation Based on Paillier Encryption (Composite Residuosity)

The solution is pretty straightforward on how to instantiate the previous scheme while relying on a DCR assumption. This simply requires the generic transformation from any native DDH scheme into a DCR based one presented in [HO09].

It is interesting to note that this boils down to using the Paillier-based CCA encryption presented in [CS02], in addition to a DCR-based Chameleon Hash encryption. (Operations are done modulo N^2 except if indicated otherwise)

For lack of space, we only present here the two needed building blocks and postpone the description of the commitment scheme and the associated smooth projective hash function to the full version.

DCR-based Chameleon Hash
We simply use a direct transposition of the Chameleon Hash described in Section 4 in a group of order Z_{N^2}. While this may be improved, the description remain simple.

$2m$-labelled multi DCR-based Encryption Scheme
We use the variant of the CCA-2 encryption introduced in [CS02]. The encryption key ek is now a tuple $(g, s, \tilde{s})$, where $g = N + 1, s = g^{k_0}$ and $\tilde{s}_i = g^{k_i}$ where $\boldsymbol{k} \xleftarrow{\$} [\![0, \lfloor N^2/2 \rfloor]\!]^{\beta+2}$, and the encryption process becomes:

$\mathsf{Encrypt}(\mathsf{pk}, M; w)$: pick $w \stackrel{\$}{\leftarrow} [\![0, N/2]\!]$ and compute $\gamma = \mathcal{H}(\ell', g^w, Ms^w, \tilde{s}_1^w)$, and $\boldsymbol{b} = (g^w, Ms^w, \tilde{s}_1^w \prod_{j=2}^{\beta+1} s_j^{w\gamma_j})$.

Once again, knowing the respective discrete logarithms in the encryption keys allows to decrypt the ciphertext.

Acknowledgements. This work was supported in part by the French ANR-14-CE28-0003 EnBiD Project.

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, December 2013.

[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, August 2009.

[BBC$^+$13a] Fabrice Ben Hamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Efficient UC-secure authenticated key-exchange for algebraic languages. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 272–291. Springer, February / March 2013.

[BBC$^+$13b] 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, August 2013.

[BC15] Olivier Blazy and Céline Chevalier. Generic Construction of UC-Secure Oblivious Transfer. Cryptology ePrint Archive, 2015. Full version of the present paper.

[BCPV13] Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. Analysis and improvement of Lindell's UC-secure commitment schemes. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 534–551. Springer, June 2013.

[BN05] 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, August 2005.

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

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, August 2001.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, May 2010.

[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, 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, April / May 2002.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 581–596. Springer, August 2002.

[EHK$^+$13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, August 2013.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, August 1984.

[FLM11] Marc Fischlin, Benoît Libert, and Mark Manulis. Non-interactive and re-usable universally composable string commitments with adaptive security. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 468–485. Springer, December 2011.

[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, May 2003. `http://eprint.iacr.org/2003/032.ps.gz`.

[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, August 2009.

[Har11] Kristiyan Haralambiev. *Efficient Cryptographic Primitives for Non-Interactive Zero-Knowledge Proofs and Applications*. PhD thesis, New York University, 2011.

[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, August 2007.

[HO09] Brett Hemenway and Rafail Ostrovsky. Lossy trapdoor functions from smooth homomorphic hash proof systems. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:127, 2009.

[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, May 2005.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society, February 2000.

[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, 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, March 2011.

[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, May 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, April 2012.

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

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, August 1991.

[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, August 2008.

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