Efficient Fully-Simulatable Oblivious Transfer*

Yehuda Lindell†

Department of Computer Science
Bar-Ilan University, ISRAEL.
lindell@cs.biu.ac.il

Abstract

Oblivious transfer, first introduced by Rabin, is one of the basic building blocks of cryptographic protocols. In an oblivious transfer (or more exactly, in its 1-out-of-2 variant), one party known as the sender has a pair of messages and the other party known as the receiver obtains one of them. Somewhat paradoxically, the receiver obtains exactly one of the messages (and learns nothing of the other), and the sender does not know which of the messages the receiver obtained. Due to its importance as a building block for secure protocols, the efficiency of oblivious transfer protocols has been extensively studied. However, to date, there are almost no known oblivious transfer protocols that are secure in the presence of *malicious adversaries* under the *real/ideal model simulation paradigm* (without using general zero-knowledge proofs). Thus, *efficient protocols* that reach this level of security are of great interest. In this paper we present efficient oblivious transfer protocols that are secure according to the ideal/real model simulation paradigm. We achieve constructions under the DDH, Nth residuosity and quadratic residuosity assumptions, as well as under the assumption that homomorphic encryption exists.

1 Introduction

In an oblivious transfer, a sender with a pair of strings m_0, m_1 interacts with a receiver so that at the end the receiver learns exactly one of the strings, and the sender learns nothing [24, 11]. This is a somewhat paradoxical situation because the receiver can only learn one string (thus the sender cannot send both) whereas the sender cannot know which string the receiver learned (and so the receiver cannot tell the sender which string to send). Surprisingly, it is possible to achieve oblivious transfer under a wide variety of assumptions and adversary models [11, 15, 19, 23, 1, 17].

Oblivious transfer is one of the most basic and widely used protocol primitives in cryptography. It stands at the center of the fundamental results on secure two-party and multiparty computation showing that any efficient functionality can be securely computed [25, 15]. In fact, it has even been shown that oblivious transfer is *complete*, meaning that it is possible to securely compute any efficient function once given a box that computes oblivious transfer [18]. Thus, oblivious transfer has great importance to the theory of cryptography. In addition to this, oblivious transfer has been widely used to construct efficient protocols for problems of interest (e.g., it is central to almost all of the work on privacy-preserving data mining).

Due to its general importance, the task of constructing efficient oblivious transfer protocols has attracted much interest. In the semi-honest model (where adversaries follow the protocol specification but try to learn more than allowed by examining the protocol transcript), it is possible to

*An extended abstract of this work appeared at *CT-RSA 2008*.

†Most of this work was carried out for Aladdin Knowledge Systems.

construct efficient oblivious transfer from (enhanced) trapdoor permutations [11] and homomorphic encryption [19, 1]. However, the situation is significantly more problematic in the malicious model where adversaries may arbitrarily deviate from the protocol specification. One possibility is to use the protocol compiler of Goldreich, Micali and Wigderson [15] to transform oblivious transfer protocols for semi-honest adversaries into protocols that are also secure in the presence of malicious adversaries. However, the result would be a highly inefficient protocol. The difficulties in obtaining secure oblivious transfer in this model seem to be due to the strict security requirements of *simulation-based definitions* that follow the ideal/real model paradigm.[1] Thus, until recently, the only known oblivious transfer protocols that were secure under this definition, and thus were *fully simulatable*, were protocols that were obtained by applying the compiler of [15]. In contrast, highly-efficient oblivious transfer protocols that guarantee *privacy* (but not simulatability) in the presence of malicious adversaries have been constructed. These protocols guarantee that even a malicious sender cannot learn which string the receiver learned, and that a malicious receiver can learn only one of the sender's input strings. Highly efficient protocols have been constructed for this setting under the DDH and N-residuosity assumptions and using homomorphic encryption [19, 23, 1, 17].

This current state of affairs is highly unsatisfactory. The reason for this is that oblivious transfer is often used as a building block in other protocols. However, oblivious transfer protocols that only provide privacy are difficult – if not impossible – to use as building blocks. Thus, the vast number of protocols that assume (fully simulatable) oblivious transfer do not have truly efficient instantiations today. For just one example, this is true of the protocol of [20] that in turn is used in the protocol of [2] for securely computing the median. The result is that [2] has no efficient instantiation, even though it *is* efficient when ignoring the cost of the oblivious transfers. We conclude that the absence of efficient fully-simulatable oblivious transfer acts as a bottleneck in numerous other protocols.

Our results. In this paper, we construct oblivious transfer protocols that are secure (i.e., fully-simulatable) in the presence of malicious adversaries. Our constructions build on those of [23, 1, 17] and use cut-and-choose techniques. It is folklore that the protocols of [23, 1, 17] can be modified to yield full simulatability by adding proofs of knowledge. To some extent, this is what we do. However, a direct application of proofs of knowledge does not work. This is because the known efficient protocols are all information-theoretically secure in the presence of a malicious receiver. This means that only one of the sender's inputs is defined by the protocol transcript and thus a standard proof of knowledge cannot be applied. (Of course, it is possible to have the sender prove that it behaved honestly according to some committed input but this will already not be efficient.) Our protocols yield full simulatability and we provide a full proof of security.

As we show, our protocols are in the order of ℓ times the complexity of the protocols of [23, 1, 17], where ℓ is such the simulation fails with probability $2^{-\ell+2}$. Thus, ℓ can be taken to be relatively small (say, in the order of 30 or 40). This is a considerable overhead. However, our protocols are still by far the most efficient known without resorting to a random oracle.

Related work. There has been much work on efficient oblivious transfer in a wide range of settings. However, very little has been done regarding fully-simulatable oblivious transfer that is also efficient (without using random oracles). Despite this, recently there has been some progress in this area. In [6], fully simulatable constructions are presented. However, these rely on strong and relatively non-standard assumptions (q-power DDH and q-strong Diffie-Hellman). Following

[1]According to this paradigm, a real execution of a protocol is compared to an ideal execution in which a trusted third party receives the parties' inputs and sends them their outputs.

this, protocols were presented that rely on the Decisional Bilinear Diffie-Hellman assumption [16]. Our protocols differ from those of [6] and [16] in the following ways:

1. *Assumptions:* We present protocols that can be constructed assuming that DDH is hard, that there exist homomorphic encryption schemes, and more. Thus, we rely on far more standard and long-standing hardness assumptions.

2. *Complexity:* Regarding the number of exponentiations, it appears that our protocols are of a similar complexity to [6, 16]. However, as pointed out in [10], bilinear curves are considerably more expensive than regular Elliptic curves. Thus, the standard decisional Diffie-Hellman assumption is much more efficient to use (curves that provide pairing need keys that are similar in size to RSA, in contrast to regular curves that can be much smaller).

3. *The problem solved:* We solve the basic 1-out-of-2 oblivious transfer problem, although our protocols can easily be extended to solve the *static* k-out-of-n oblivious transfer problem (where static means that the receiver must choose which k elements it wishes to receive at the onset). In contrast, [6] and [16] both solve the considerably harder problem of *adaptive* k-out-of-n oblivious transfer where the receiver chooses the elements to receive one and a time, and can base its choice on the elements it has already received.

In conclusion, if adaptive k-out-of-n oblivious transfer is needed, then [6, 16] are the best solutions available. However, if (static) oblivious transfer suffices, then our protocols are considerably more efficient and are based on far more standard assumptions.

2 Definitions

In this section we present the definition of security for oblivious transfer, that is based on the general simulation-based definitions for secure computation; see [14, 21, 5, 7]. We refer the reader to [12, Chapter 7] for full definitions, and provide only a brief overview here. Since we only consider oblivious transfer in this paper, our definitions are tailored to the secure computation of this specific function only.

Preliminaries. We denote by $s \in_R S$ the process of randomly choosing an element s from a set S. A function $\mu(\cdot)$ is negligible in n, or just negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n's it holds that $\mu(n) < 1/p(n)$. A probability ensemble $X = \{X(n,a)\}_{a\in\{0,1\}^*;n\in\mathsf{N}}$ is an infinite sequence of random variables indexed by a and $n \in \mathsf{N}$. (The value a will represent the parties' inputs and n the security parameter.) Two distribution ensembles $X = \{X(n,a)\}_{n\in\mathsf{N}}$ and $Y = \{Y(n,a)\}_{n\in\mathsf{N}}$ are said to be computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(n,a),a) = 1] - \Pr[D(Y(n,a),a) = 1]| \leq \mu(n)$$

All parties are assumed to run in time that is polynomial in the security parameter. (Formally, each party has a security parameter tape upon which that value 1^n is written. Then the party is polynomial in the input on this tape.)

Oblivious transfer. The oblivious transfer functionality is formally defined as a function f with two inputs and one output. The first input is a pair (m_0, m_1) and the second input is a bit σ. The output is the string m_σ. Party P_1, also known as the sender, inputs (m_0, m_1) and receives no output. In contrast, party P_2, also known as the receiver, inputs the bit σ and receives m_σ for output. Formally, we write $f((m_0, m_1), \sigma) = (\lambda, m_\sigma)$ where λ denotes the empty string. Stated in words, in the oblivious transfer functionality party P_1 receives no output, whereas party P_2 receives m_σ (and learns nothing about $m_{1-\sigma}$).

Adversarial behavior. Loosely speaking, the aim of a secure two-party protocol is to protect an honest party against dishonest behavior by the other party. In this paper, we consider *malicious adversaries* who may arbitrarily deviate from the specified protocol. Furthermore, we consider the *static corruption model*, where one of the parties is adversarial and the other is honest, and this is fixed before the execution begins.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation.

Oblivious transfer in the ideal model. An ideal oblivious transfer execution proceeds as follows:

Inputs: Party P_1 obtains an input pair (m_0, m_1) with $|m_0| = |m_1|$, and party P_2 obtains an input bit σ.

Send inputs to trusted party: An honest party always sends its input unchanged to the trusted party. A malicious party may either abort, in which case it sends $\perp$ to the trusted party, or send some other input to the trusted party.

Trusted party computes output: If the trusted party receives $\perp$ from one of the parties, then it sends $\perp$ to both parties and halts. Otherwise, upon receiving some (m'_0, m'_1) from P_1 and a bit σ' from P_2, the trusted party sends $m'_{\sigma'}$ to party P_2 and halts.

Outputs: An honest party always outputs the message it has obtained from the trusted party ($\perp$ or nothing in the case of P_1, and $\perp$ or $m'_{\sigma'}$ in the case of P_2). A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Denote by f the oblivious transfer functionality and let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic *expected* polynomial-time machines (representing parties in the ideal model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the honest party instructions in the above-described ideal execution). Then, the joint execution of f under $\overline{M}$ in the ideal model (on input $((m_0, m_1), \sigma)$), denoted $\text{IDEAL}_{f,\overline{M}}((m_0, m_1), \sigma)$, is defined as the output pair of M_1 and M_2 from the above ideal execution.

Execution in the real model. We next consider the real model in which a real two-party protocol is executed and there exists no trusted third party. In this case, a malicious party may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform probabilistic polynomial-time machines. Let π be a two-party protocol. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the real model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the strategy specified by π). Then, the joint execution of π under $\overline{M}$ in the real model (on input $((m_0, m_1), \sigma)$), denoted $\text{REAL}_{\pi,\overline{M}}((m_0, m_1), \sigma)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

Definition 1 *Let f denote the oblivious transfer protocol and let π be a two-party protocol. Protocol π is said to be a* secure oblivious transfer protocol *if for every pair of admissible non-uniform probabilistic polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines $\overline{B} = (B_1, B_2)$ for the ideal model, such that for every $m_0, m_1 \in \{0, 1\}^*$ of the same length and every $\sigma \in \{0, 1\}$,*

$$\left\{\text{IDEAL}_{f,\overline{B}}(n, (m_0, m_1), \sigma)\right\} \stackrel{c}{\equiv} \left\{\text{REAL}_{\pi,\overline{A}}(n, (m_0, m_1), \sigma)\right\}$$

Note that we allow the ideal adversary/simulator to run in expected (rather than strict) polynomial-time. This is essential for achieving constant-round protocols; see [4].

3 Oblivious Transfer Under the DDH Assumption

In this section we present an oblivious transfer protocol that is secure in the presence of malicious adversaries, under the DDH assumption. The protocol is a variant of the two-round protocol of [23] with some important changes. Before proceeding, we recall the protocol of [23]. Basically, this protocol works by the receiver generating a tuple (g^a, g^b, g^c, g^d) with the following property: if the receiver's input equals 0 then $c = ab$ and d is random, and if the receiver's input equals 1 then $d = ab$ and c is random. The sender receives this tuple and carries out a manipulation that randomizes the tuple so that if $c = ab$ then the result of the manipulation on (g^a, g^b, g^c) is still a DDH tuple and the result of the manipulation on (g^a, g^b, g^d) yields a completely random tuple (if $d = ab$ then the same holds in reverse). The sender then derives a secret key from the manipulation of each of (g^a, g^b, g^c) and (g^a, g^b, g^d), and sends information that enables the receiver to derive the same secret key from the DDH tuple, whereas the key from the non-DDH tuple remains completely random. In addition, the sender encrypts its first message under the key derived from (g^a, g^b, g^c) and its second message under the key derived from (g^a, g^b, g^d). The receiver is able to decrypt the message derived from the DDH tuple but has no information about the other key and so cannot learn anything about the other message. We remark that the sender checks that $g^c \neq g^d$. This ensures that only one of (g^a, g^b, g^c) and (g^a, g^b, g^d) is a DDH tuple.

The secret key that is derived from the non-DDH tuple above is information-theoretically hidden from the receiver. This causes a problem when attempting to construct a simulator for the protocol because the simulator must learn *both* of the sender's inputs in order to send them to the trusted party (and for whatever first message the simulator sends, it can only learn one of the sender's inputs). We remark that if rewinding is used to obtain both messages then this causes a problem because the sender can make its input depend on the first message from the receiver. We therefore change the protocol of [23] so that instead of sending (g^a, g^b, g^c, g^d) where at most one of c or d equals $a \cdot b$, the receiver sends two tuples: one of the tuples is a DDH type and the other is *not*. The parties then interact to ensure that indeed only one of the tuples is of the DDH type. As we will see, this ensures that the receiver obtains only one message. The "interaction" used to prove this is of the simplest cut-and-choose type.

The protocol below uses two commitment schemes for the purpose of coin tossing: a perfectly hiding commitment scheme denoted Com_h, and a perfectly binding commitment scheme, denoted Com_b. We remark that such commitment schemes exist under the Discrete Log assumption, and thus also under the DDH assumption. We assume that the input values m_0, m_1 of the sender are in the group $\mathcal{G}$ that we are working with for the DDH assumption. If they cannot be mapped to $\mathcal{G}$ (e.g., they are too long), then the oblivious transfer can be used to exchange secret keys k_0 and k_1 that are used to encrypt m_0 and m_1, respectively.

Protocol 1

- **Auxiliary input:** *The parties have the description of a group $\mathcal{G}$ of order q, and a generator g for the group. In addition, they have a statistical error parameter ℓ.*
- **Input:** *The sender has a pair of group elements (m_0, m_1) and the receiver has a bit σ.*
- **The protocol:**
  1. *For $i = 1, \ldots, \ell$, the receiver P_2 chooses a random bit $\sigma_i \in_R \{0,1\}$ and random values $a_i^0, b_i^0, c_i^0, a_i^1, b_i^1, c_i^1 \in_R \{1, \ldots, q\}$ under the constraint that $c_i^{\sigma_i} = a_i^{\sigma_i} \cdot b_i^{\sigma_i}$ and $c_i^{1-\sigma_i} \neq a_i^{1-\sigma_i} \cdot b_i^{1-\sigma_i}$. Then, P_2 computes the tuples $\gamma_i^0 = (g^{a_i^0}, g^{b_i^0}, g^{c_i^0})$ and $\gamma_i^1 = (g^{a_i^1}, g^{b_i^1}, g^{c_i^1})$. Note that $\gamma_i^{\sigma_i}$ is a DDH tuple and $\gamma_i^{1-\sigma_i}$ is not.*
     P_2 sends all of the pairs $\langle(\gamma_1^0, \gamma_1^1), \ldots, (\gamma_\ell^0, \gamma_\ell^1)\rangle$ to the sender P_1.
  2. Coin tossing:
     (a) *P_1 chooses a random $s \in_R \{0,1\}^\ell$ and sends $\mathsf{Com}_h(s)$ to P_2.*
     (b) *P_2 chooses a random $s' \in_R \{0,1\}^\ell$ and sends $\mathsf{Com}_b(s')$ to P_1.*
     (c) *P_1 and P_2 send decommitments to $\mathsf{Com}_h(s)$ and $\mathsf{Com}_b(s')$, respectively, and set $r = s \oplus s'$. Denote $r = r_1, \ldots, r_\ell$.*
  3. *For every i for which $r_i = 1$, party P_2 sends $a_i^0, b_i^0, c_i^0, a_i^1, b_i^1, c_i^1$ to P_1.*
     In addition, for every j for which $r_j = 0$, party P_2 sends a "reordering" of γ_j^0 and γ_j^1 so that all of the γ_j^σ tuples are DDH tuples and all of the $\gamma_j^{1-\sigma}$ tuples are not. This reordering is a bit such that if it equals 0 then the tuples are left as is, and if it equals 1 then γ_j^0 and γ_j^1 are interchanged.
  4. *P_1 checks that for every i for which $r_i = 1$ it received the appropriate values and that they define γ_i^0 and γ_i^1. Furthermore, it checks that exactly one of γ_i^0 and γ_i^1 is a DDH tuple as defined above and the other is not. If any of the checks fail, P_1 halts and outputs $\perp$. Otherwise it continues as follows:*
     (a) *Denote $\gamma_j^0 = (x_j^0, y_j^0, z_j^0)$ and $\gamma_j^1 = (x_j^1, y_j^1, z_j^1)$. Then, for every j for which $r_j = 0$, party P_1 chooses random $u_i^0, u_i^1, v_i^0, v_i^1 \in_R \{1, \ldots, q\}$ and computes the following four*

values:

$$w_j^0 = \left(x_j^0\right)^{u_i^0} \cdot g^{v_i^0} \qquad\qquad k_j^0 = \left(z_j^0\right)^{u_i^0} \cdot \left(y_j^0\right)^{v_i^0}$$
$$w_j^1 = \left(x_j^1\right)^{u_i^1} \cdot g^{v_i^1} \qquad\qquad k_j^1 = \left(z_j^1\right)^{u_i^1} \cdot \left(y_j^1\right)^{v_i^1}$$

(b) *Let $j_1, \ldots, j_t$ be the indices j for which $r_j = 0$. Then, P_1 "encrypts" m_0 under all of the keys k_j^0, and m_1 under all of the keys k_j^1, as follows:*

$$c_0 = \left(\prod_{i=1}^{t} k_{j_i}^0\right) \cdot m_0 \qquad\qquad c_1 = \left(\prod_{i=1}^{t} k_{j_i}^1\right) \cdot m_1$$

P_1 sends P_2 all of the w_j^0, w_j^1 values, as well as the pair (c_0, c_1).

5. *For every j for which $r_j = 0$, party P_2 computes $k_j^\sigma = (w_j^\sigma)^{b_j^0}$. Then, P_2 outputs $m_\sigma = c_\sigma \cdot \left(\prod_{i=1}^{t} k_{j_i}^\sigma\right)^{-1}$.*

Before proceeding to the proof, we show that the protocol "works", meaning that when P_1 and P_2 are honest, the output is correctly obtained. We present this to "explain" the computations that take place in the protocol, although these are exactly as in the protocol of [23]. First, notice that

$$\left(w_j^\sigma\right)^{b_j^\sigma} = \left(x_j^\sigma\right)^{u_j^\sigma \cdot b_j^\sigma} \cdot \left(g^{v_j^\sigma}\right)^{b_j^\sigma} = \left(g^{a_j^\sigma \cdot b_j^\sigma}\right)^{u_j^\sigma} \cdot \left(g^{b_j^\sigma}\right)^{v_j^\sigma}$$

By the fact that γ_j^σ is a DDH tuple we have that $g^{a_j^\sigma \cdot b_j^\sigma} = z_j^\sigma$ and so

$$\left(w_j^\sigma\right)^{b_j^\sigma} = \left(z_j^\sigma\right)^{u_j^\sigma} \cdot \left(y_j^\sigma\right)^{v_j^\sigma} = k_j^\sigma$$

Thus P_2 correctly computes each key k_j^σ for j such that $r_j = 0$. Given all of these keys, it immediately follows that P_2 can decrypt c_σ, obtaining m_σ. We now proceed to prove the security of the protocol.

Theorem 1 *Assume that the decisional Diffie-Hellman problem is hard in $\mathcal{G}$ with generator g, that Com_h is a perfectly-hiding commitment scheme, and that Com_b is a perfectly-binding commitment scheme. Then, Protocol 1 securely computes the oblivious transfer functionality in the presence of malicious adversaries.*

Proof: We separately prove the security of the protocol for the case that no parties are corrupted, P_1 is corrupted, and P_2 is corrupted. In the case that both P_1 and P_2 are honest, we have already seen that P_2 obtains exactly m_σ. Thus, security holds. We now proceed to the other cases.

P_1 is corrupted. Let $\mathcal{A}_1$ be a non-uniform probabilistic polynomial-time real adversary that controls P_1. We construct a non-uniform probabilistic expected polynomial-time ideal-model adversary/simulator $\mathcal{S}_1$. The basic idea behind how $\mathcal{S}_1$ works is that it uses rewinding in order to ensure that all of the "checked" tuples are valid (i.e., one is a DDH tuple and the other is not), whereas all of the "unchecked" tuples have the property that they are *both* of the DDH type. Now, since the protocol is such that a receiver can obtain a key k_j^σ as long as γ_j^σ was a DDH tuple, it follows that $\mathcal{S}_1$ can obtain all of the k_j^0 and k_j^1 keys. This enables it to decrypt both c_0 and c_1 and obtain both messages input by $\mathcal{A}_1$ into the protocol. $\mathcal{S}_1$ then sends these inputs to the trusted party, and the honest party P_2 in the ideal model will receive the same message that it would have received in a real execution with $\mathcal{A}_1$ (or more accurately, a message that is computationally indistinguishable from that message).

We now describe $\mathcal{S}_1$ formally. Upon input 1^n and (m_0, m_1), the machine $\mathcal{S}_1$ invokes $\mathcal{A}_1$ upon the same input and works as follows:

1. $\mathcal{S}_1$ chooses a random $r \in_R \{0,1\}^\ell$ and generates tuples $\gamma_1^0, \gamma_1^1, \ldots, \gamma_\ell^0, \gamma_\ell^1$ with the following property:
   (a) For every i for which $r_i = 1$, $\mathcal{S}_1$ constructs γ_i^0 and γ_i^1 like an honest P_2 (i.e., one of them being a DDH tuple and the other not, in random order).
   (b) For every j for which $r_j = 0$, $\mathcal{S}_1$ constructs γ_j^0 and γ_j^1 to *both* be DDH tuples.

   $\mathcal{S}_1$ hands the tuples to $\mathcal{A}_1$.

2. *Simulation of the coin tossing:* $\mathcal{S}_1$ simulates the coin tossing so that the result is r, as follows:
   (a) $\mathcal{S}_1$ receives a commitment c_h from $\mathcal{A}_1$.
   (b) $\mathcal{S}_1$ chooses a random $s' \in_R \{0,1\}^\ell$ and hands $c_b = \mathsf{Com}_b(s')$ to $\mathcal{A}_1$.
   (c) If $\mathcal{A}_1$ does not send a valid decommitment to c_h, then $\mathcal{S}_1$ simulates P_2 aborting and sends $\perp$ to the trusted party. Then $\mathcal{S}_1$ outputs whatever $\mathcal{A}_1$ outputs and halts.

      Otherwise, let s be the decommitted value. $\mathcal{S}_1$ proceeds as follows:
      i. $\mathcal{S}_1$ sets $s' = r \oplus s$, rewinds $\mathcal{A}_1$, and hands it $\mathsf{Com}_b(s')$.
      ii. If $\mathcal{A}_1$ decommits to s, then $\mathcal{S}_1$ proceeds to the next step. If $\mathcal{A}_1$ decommits to a value $\tilde{s} \neq s$, then $\mathcal{S}_1$ outputs fail. Otherwise, if it does not decommit to any value, $\mathcal{S}_1$ returns to the previous step and tries again until $\mathcal{A}_1$ does decommit to s. (We stress that in every attempt, $\mathcal{S}_1$ hands $\mathcal{A}_1$ a commitment to the same value s'. However, the randomness used to generate the commitment $\mathsf{Com}_b(s')$ is independent each time.)[2]

3. Upon receiving a valid decommitment to s from $\mathcal{A}_1$, simulator $\mathcal{S}_1$ decommits to $\mathcal{A}_1$, revealing s'. (Note that $r = s \oplus s'$.)

4. For every i for which $r_i = 1$, simulator $\mathcal{S}_1$ hands $\mathcal{A}_1$ the values $a_i^0, b_i^0, c_i^0, a_i^1, b_i^1, c_i^1$ used to generate γ_i^0 and γ_i^1. In addition, $\mathcal{S}_1$ hands $\mathcal{A}_1$ a random reordering of the pairs.

5. If $\mathcal{A}_1$ does not reply with a valid message, then $\mathcal{S}_1$ sends $\perp$ to the trusted party, outputs whatever $\mathcal{A}_1$ outputs and halts. Otherwise, it receives a series of pairs (w_j^0, w_j^1) for every j for which $r_j = 0$, as well as ciphertexts c_0 and c_1. $\mathcal{S}_1$ then follows the instructions of P_2 for deriving the keys. However, unlike an honest P_2, it computes $k_j^0 = (w_j^0)^{b_j^0}$ and $k_j^1 = (w_j^1)^{b_j^1}$ and uses the keys it obtains to decrypt *both* c_0 and c_1. (Note that for each such j, both γ_j^0 and γ_j^1 are DDH tuples; thus this makes sense.)

   Let m_0 and m_1 be the messages obtained by decrypting. $\mathcal{S}_1$ sends the pair to the trusted party as the first party's input, outputs whatever $\mathcal{A}_1$ outputs and halts.

[2]This strategy by $\mathcal{S}_1$ is actually over-simplified and does not guarantee that it runs in expected polynomial-time. This technicality will be discussed below, and we will show how $\mathcal{S}_1$ can be "fixed" so that its expected running-time is polynomial.

We now prove that the joint output distribution of $\mathcal{S}_1$ and an honest P_2 in an ideal execution is computationally indistinguishable from the output distribution of $\mathcal{A}_1$ and an honest P_2 in a real execution. First, note that the view of $\mathcal{A}_1$ in the simulation with $\mathcal{S}_1$ is indistinguishable from its view in a real execution. The only difference in its view is due to the fact that the tuples γ_j^0 and γ_j^1 for which $r_j = 0$ are both of the DDH type. The only other difference is due to the coin tossing (and the rewinding). However, by the binding property of the commitment sent by $\mathcal{A}_1$ and the fact that P_2 generates its commitment after receiving $\mathcal{A}_1$'s, we have that the outcome of the coin tossing in a real execution is statistically close to uniform (where the only difference is due to the negligible probability that $\mathcal{A}_1$ will break the computational binding property of the commitment scheme.) In the simulation by $\mathcal{S}_1$, the outcome is always uniformly distributed, assuming that $\mathcal{S}_1$ does not output fail. Since $\mathcal{S}_1$ outputs fail when $\mathcal{A}_1$ breaks the computational binding of the commitment scheme, this occurs with at most negligible probability (a rigorous analysis of this is given in [13]). We therefore have that, apart from the negligible difference due to the coin tossing, the only difference is due to the generation of the tuples. Intuitively, indistinguishability therefore follows from the DDH assumption. More formally, this is proven by constructing a machine D that distinguishes many copies of DDH tuples from many copies of non-DDH tuples. D receives a series of tuples and runs in exactly the same way as $\mathcal{S}_1$ except that it constructs the γ_j^0 and γ_j^1 tuples (for $r_j = 0$) so that one is a DDH tuple and the other is from its input, in random order. Furthermore, it provides the reordering so that all of the DDH tuples it generates are associated with σ and all of the ones it receives externally are associated with $1 - \sigma$. (For the sake of this mental experiment, we assume that D is given the input σ of P_2.) It follows that if D receives a series of DDH tuples, then the view of $\mathcal{A}_1$ is exactly the same as in the simulation with $\mathcal{S}_1$ (because all the tuples are of the Diffie-Hellman type). In contrast, if D receives a series of non-DDH tuples, then the view of $\mathcal{A}_1$ is exactly the same as in a real execution (because only the tuples associated with σ are of the Diffie-Hellman type). This suffices for showing that the output of $\mathcal{A}_1$ in a real execution is indistinguishable from the output of $\mathcal{S}_1$ in an ideal execution (recall that $\mathcal{S}_1$ outputs whatever $\mathcal{A}_1$ outputs). However, we have to show this for the joint distribution of the output of $\mathcal{A}_1$ (or $\mathcal{S}_1$) and the honest P_2. In order to see this, recall that the output of P_2 is m_σ where σ is the honest P_2's input. Now, assume that there exists a polynomial-time distinguisher D' that distinguishes between the REAL and IDEAL distributions with non-negligible probability. We construct a distinguisher D as above that distinguishes DDH from non-DDH tuples. The machine D receives the input σ of P_2 and a series of tuples that are either DDH or non-DDH tuples. D then works exactly as above (i.e., constructing the γ_j^0 and γ_j^1 tuples so that in the reordering step, all the γ_j^σ tuples are those it generated itself and all the $\gamma_j^{1-\sigma}$ tuples are those it received as input). Since D generated all of the γ_j^σ tuples, it is able to "decrypt" c_σ and obtain m_σ. Machine D therefore does this, and invokes D' on the output of $\mathcal{A}_1$ *and* the message m_σ (which is the output that an honest P_2 would receive). Finally D outputs whatever D' does. It is clear that if D receives non-DDH tuples, then the output distribution generated is exactly like that of a real execution between $\mathcal{A}_1$ and P_2. In contrast, if it receives DDH tuples, then the output distribution is exactly like of an ideal execution with $\mathcal{S}_1$. (A subtle point here is that the distribution over the γ tuples generated by D who knows σ is identical to the distribution generated by $\mathcal{S}_1$ who does not know σ. The reason for this is that when all the tuples are of the DDH type, their ordering makes no difference.) We conclude that D solves the DDH problem with non-negligible probability, in contradiction to the DDH assumption. Thus, the REAL and IDEAL output distributions must be computationally indistinguishable, as required.

It remains to prove that $\mathcal{S}_1$ runs in expected polynomial-time. Unfortunately, this is not true! In order to see this, denote by p the probability that $\mathcal{A}_1$ decommits correctly to s when it receives a commitment to a random s'. Next, denote by q the probability that $\mathcal{A}_1$ decommits correctly when it receives a commitment to $s' = s \oplus r$. (Note that this is not random because r is implicit in the way that $\mathcal{S}_1$ generated the tuples. That is, if $r_i = 1$ then γ_i^0 and γ_i^1 are honestly generated, and otherwise they are both of the DDH type.) Now, by the hiding property of the commitment scheme Com_b, the difference between p and q can be at most negligible. Furthermore, the expected running-time of $\mathcal{S}_1$ in the rewinding stage equals p/q times some fixed polynomial factor. In order to see this, observe that $\mathcal{S}_1$ enters the rewinding stage with probability p, and concludes after an expected $1/q$ number of rewindings. It thus remains to bound p/q. (We remark that $\mathcal{S}_1$'s running time in the rest of the simulation is a fixed polynomial and so we ignore this from now on). Unfortunately, even though p and q are at most negligibly far from each other, as we have discussed, the value p/q may not necessarily be polynomial. For example, if $p = 2^{-n}$ and $q = 2^{-n} + 2^{-n/2}$ then $p/q \approx 2^{n/2}$. Thus, the expected running-time of $\mathcal{S}_1$ is not necessarily polynomial. Fortunately, this can be solved using the techniques of [13] who solved an identical problem. Loosely speaking, the technique of [13] works by first estimating p and then ensuring that the number of rewinding attempts does not exceed a fixed polynomial times the estimation of p. It is shown that this yields a simulator that is guaranteed to run in expected polynomial time. Furthermore, the output of the simulator is only negligibly far from the original (simplified) strategy described above. Thus, these techniques can be applied here and the simulator appropriately changed, with the result being that the output is only negligibly different from before, as required.

***P_2* is corrupted.** As before, we let $\mathcal{A}_2$ be any non-uniform probabilistic polynomial-time adversary controlling P_2 and we construct a non-uniform probabilistic expected polynomial-time simulator $\mathcal{S}_2$. The simulator $\mathcal{S}_2$ extracts the bit σ used by $\mathcal{A}_2$ by rewinding it and obtaining the reordering of tuples that it had previously opened. Formally, upon input 1^n and σ, the simulator $\mathcal{S}_2$ invokes $\mathcal{A}_2$ upon the same input and works as follows:

1. $\mathcal{S}_2$ receives a series of tuples $\gamma_1^0, \gamma_1^1, \ldots, \gamma_\ell^0, \gamma_\ell^1$ from $\mathcal{A}_2$.

2. $\mathcal{S}_2$ hands $\mathcal{A}_2$ a commitment $c_h = \mathsf{Com}_h(s)$ to a random $s \in_R \{0,1\}^\ell$, receives back c_b, decommits to c_h and receives $\mathcal{A}_2$'s decommitment to c_b. $\mathcal{S}_2$ then receives all of the $a_i^0, b_i^0, c_i^0, a_i^1, b_i^1, c_i^1$ values from $\mathcal{A}_2$, for i where $r_i = 1$, and the reorderings for j where $r_j = 0$. If the values sent by $\mathcal{A}_2$ are not valid (as checked by P_1 in the protocol) or $\mathcal{A}_2$ did not send valid decommitments, $\mathcal{S}_2$ sends $\perp$ to the trusted party, outputs whatever $\mathcal{A}_2$ outputs, and halts. Otherwise, it continues to the next step.

3. $\mathcal{S}_2$ rewinds $\mathcal{A}_2$ back to the beginning of the coin-tossing, hands $\mathcal{A}_2$ a commitment $\tilde{c}_h = \mathsf{Com}_h(\tilde{s})$ to a fresh random $\tilde{s} \in_R \{0,1\}^\ell$, receives back some $\tilde{c}_b$, decommits to $\tilde{c}_h$ and receives $\mathcal{A}_2$'s decommitment to $\tilde{c}_b$. In addition, $\mathcal{S}_2$ receives the $a_i^0, b_i^0, c_i^0, a_i^1, b_i^1, c_i^1$ values and reorderings.

   If any of the values are not valid, $\mathcal{S}_2$ repeats this step using fresh randomness each time, until all values are valid.

4. Following this, $\mathcal{S}_2$ rewinds $\mathcal{A}_2$ to the beginning and resends the exact messages of the first coin tossing (resulting in exactly the same transcript as before).

5. Denote by r the result of the first coin tossing (Step 2 above), and $\tilde{r}$ the result of the second coin tossing (Step 3 above). If $r = \tilde{r}$ then $\mathcal{S}_2$ outputs fail and halts. Otherwise, $\mathcal{S}_2$ searches for a value t such that $r_t = 0$ and $\tilde{r}_t = 1$. (Note that by the definition of the simulation, exactly one of γ_t^0 and γ_t^1 is a DDH tuple. Otherwise, the values would not be considered valid.) If no such t exists (i.e., for every t such that $r_t \neq \tilde{r}_t$ it holds that $r_t = 1$ and $\tilde{r}_t = 0$), then $\mathcal{S}_2$

begins the simulation from scratch with the exception that it must find r and $\tilde{r}$ for which all values are valid (i.e., if for r the values sent by $\mathcal{A}_2$ are not valid it does not terminate the simulation but rather rewinds until it finds an r for which the responses of $\mathcal{A}_2$ are all valid).

If $\mathcal{S}_2$ does not start again, we have that it has $a_t^0, b_t^0, c_t^0, a_t^1, b_t^1, c_t^1$ and can determine which of γ_t^0 and γ_t^1 is a DDH tuple. Furthermore, since $\tilde{r}_t = 1$, the reordering that $\mathcal{S}_2$ receives from $\mathcal{A}_2$ after the coin tossing indicates whether the DDH tuple is associated with 0 or with 1. $\mathcal{S}_2$ sets $\sigma = 0$ if after the reordering γ_t^0 is of the DDH type, and sets $\sigma = 1$ if after the reordering γ_t^1 is of the DDH type. (Note that exactly one of the tuples is of the DDH type because this is checked in the second coin tossing.)

6. $\mathcal{S}_2$ sends σ to the trusted party and receives back a string $m = m_\sigma$. Simulator $\mathcal{S}_2$ then computes the last message from P_1 to P_2 honestly, while encrypting m_σ under the keys k_j^σ (and encrypting any arbitrary string of the same length under the keys $k_{1-\sigma}^j$). $\mathcal{S}_2$ hands $\mathcal{A}_2$ these messages and outputs whatever $\mathcal{A}_2$ outputs and halts.

We now prove that the output distribution of $\mathcal{A}_2$ in a real execution with an honest P_1 (with input (m_0, m_1)) is computationally indistinguishable from the output distribution of $\mathcal{S}_2$ in an ideal execution with an honest P_1 (with the same input (m_0, m_1)). We begin by showing that $\mathcal{S}_2$ outputs fail with probability at most $2^{-\ell}$, ignoring for now the probability that $r = \tilde{r}$ in later rewindings (which may occur if $\mathcal{S}_2$ has to start again from scratch). Recall that this event occurs if everything is "valid" after the first coin tossing (where the result is r), and the result of the second coin-tossing after which everything is valid is $\tilde{r} = r$.[3] First, observe that the *distributions* of the strings r and $\tilde{r}$ are identical. This is because $\mathcal{S}_2$ runs the coin tossing in the same way each time (using fresh random coins), and accepts $\tilde{r}$ when all is valid, exactly as what happened with r. Next, note that the distribution over the result of the coin tossing – without conditioning over $\mathcal{A}_2$ sending valid decommitments – is uniform. This holds because the commitment that $\mathcal{S}_2$ hands to $\mathcal{A}_2$ is perfectly hiding and the commitment returned by $\mathcal{A}_2$ to $\mathcal{S}_2$ is perfectly binding. Let R be a random variable that denotes the result of the first coin tossing between $\mathcal{A}_2$ and $\mathcal{S}_2$ in the simulation, and let valid be the event that $\mathcal{A}_2$ replies with valid decommitments and values after the first coin tossing. Finally, for a given $r \in \{0,1\}^\ell$, let obtain_r denote the event that the result of one of the coin tossing attempts in the second stage equals r. (Note that this does not mean that $\tilde{r} = r$ because $\tilde{r}$ is the result that is finally accepted after $\mathcal{A}_2$ sends valid values. However, the decision of $\mathcal{A}_2$ to send valid values may also depend on the randomness used to generate $\mathsf{Com}_h(s)$. Thus, $\tilde{r}$ may not equal r, even though r is obtained in one of the coin tossing attempts in the second stage.) Clearly, fail can only occur if r is obtained at least once as the result of a coin tossing attempt in the second stage (because fail can only occur if $\tilde{r} = r$). We therefore have the following:

$$\Pr[\mathsf{fail}] \leq \sum_{r \in \{0,1\}^\ell} \Pr[R = r \;\&\; \mathsf{valid}] \cdot \Pr[\mathsf{obtain}_r] \tag{1}$$

Before analyzing this probability, we compute $\Pr[\mathsf{obtain}_r]$ for a fixed r. Let p denote the probability (over $\mathcal{A}_2$ and $\mathcal{S}_2$'s coin tosses) that $\mathcal{A}_2$ sends valid values after the coin tossing. It follows that the expected number of trials by $\mathcal{S}_2$ in the second coin tossing is $1/p$. Letting X_r be a Boolean random variable that equals 1 if and only if the result of the second coin tossing attempt equals the fixed r, we have that $E[X_r] = 2^{-\ell}$. By Wald's equation (e.g., see [22, Page 300]), it follows that the expected number of times that r is obtained as the result of a coin tossing attempt in the second

[3] It is very easy to prove that the probability that $\mathcal{S}_2$ outputs fail is at most $2^{-\ell/2}$. However, in order to keep ℓ to a low value, we present a more subtle analysis that demonstrates that $\mathcal{S}_2$ outputs fail with probability at most $2^{-\ell}$.

stage by $\mathcal{S}_2$ is $1/p \cdot 2^{-\ell}$. Using Markov's inequality, we have that the probability that r is obtained at least once as the result of a coin tossing attempt in the second stage is at most $1/p \cdot 2^{-\ell}$. That is:

$$\Pr[\mathsf{obtain}_r] \leq \frac{1}{p \cdot 2^\ell}$$

We are now ready to return to Eq. (1). Denote by p_r the probability that $\mathcal{A}_2$ sends valid values conditioned on the outcome of the coin tossing being r. It follows that

$$p = \sum_{r \in \{0,1\}^\ell} \Pr[R = r] \cdot p_r = \sum_{r \in \{0,1\}^\ell} \frac{p_r}{2^\ell}$$

Furthermore,

$$\Pr[R = r \ \& \ \mathsf{valid}] = \Pr[\mathsf{valid} \mid R = r] \cdot \Pr[R = r] = p_r \cdot \frac{1}{2^\ell}$$

Combining the above, we have:

$$\begin{aligned}
\Pr[\mathsf{fail}] &\leq \sum_{r \in \{0,1\}^\ell} \Pr[R = r \ \& \ \mathsf{valid}] \cdot \Pr[\mathsf{obtain}_r] \\
&\leq \sum_{r \in \{0,1\}^\ell} \frac{p_r}{2^\ell} \cdot \frac{1}{p \cdot 2^\ell} \\
&= \frac{1}{p \cdot 2^\ell} \cdot \sum_{r \in \{0,1\}^\ell} \frac{p_r}{2^\ell} \\
&= \frac{1}{p \cdot 2^\ell} \cdot p = \frac{1}{2^\ell}
\end{aligned}$$

We conclude that $\mathcal{S}_2$ outputs fail with probability at most $2^{-\ell}$, as required. Recall that this analysis doesn't take into account the probability that $\mathcal{S}_2$ starts the simulation from scratch. Rather, it just shows that $\mathcal{S}_2$ outputs fail in any simulation attempt (between starts from scratch) with probability at most $2^{-\ell}$. Below, we will show that the probability that $\mathcal{S}_2$ starts from scratch is at most $1/2$. Denote by fail_i the probability that $\mathcal{S}_2$ outputs fail in the ith attempt, given that there is such an attempt. Likewise, denote by repeat_i the probability that $\mathcal{S}_2$ has an ith attempt. We have shown that for every i, $\Pr[\mathsf{fail}_i] = 2^{-\ell}$, and below we show that every repeat happens with probability $1/2$ and so for every i, $\Pr[\mathsf{repeat}_i] = 2^{i-1}$ ($\mathsf{repeat}_1 = 1$ because we always have one attempt). We therefore have:

$$\Pr[\mathsf{fail}] = \sum_{i=1}^{\infty} \Pr[\mathsf{fail}_i] \cdot \Pr[\mathsf{repeat}_i] = \frac{1}{2^\ell} \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = \frac{1}{2^\ell} \cdot 2 = \frac{1}{2^{\ell-1}}$$

Given the above, we proceed to show indistinguishability of the ideal and real distributions. Notice that in the case that $\mathcal{S}$ does not output fail, the final transcript as viewed by $\mathcal{A}_2$ consists of the first coin tossing (that is distributed exactly as in a real execution) and the last message from $\mathcal{S}_2$ to $\mathcal{A}_2$. This last message is not generated honestly, in that c_σ is indeed an encryption of m_σ, but $c_{1-\sigma}$ is an encryption of an arbitrary value (and not necessarily of $m_{1-\sigma}$). However, as shown in [23], for any tuple $\gamma_j^{1-\sigma}$ that is *not* a DDH tuple, the value $k_j^{1-\sigma}$ is uniformly distributed in $\mathcal{G}$ (even given $w_j^{1-\sigma}$ as received by $\mathcal{A}_2$). This implies that $c_{1-\sigma}$ is uniformly distributed, independent of the value $m_{1-\sigma}$. Thus, $\mathcal{A}_2$'s view in the execution with $\mathcal{S}_2$ is statistically close to its view in a real execution with P_1 (the only difference being if $\mathcal{S}_2$ outputs fail). This completes the proof regarding indistinguishability.

It remains to prove that $\mathcal{S}_2$ runs in expected polynomial-time. We begin by analyzing the rewinding by $\mathcal{S}_2$ in the coin tossing phase (clearly, the running-time of $\mathcal{S}_2$ outside of the rewinding is strictly polynomial, and so it suffices to bound the expected number of rewinding attempts). Denote by p the probability that $\mathcal{A}_2$ completes the coin tossing phase and provides valid values to $\mathcal{S}_2$. The important point to note here is that each rewinding attempt is successful with probability exactly p (there is no difference between the distribution over the first and second coin tossing attempts, in contrast to the simulation where P_1 is corrupted). Thus, with probability p there are rewinding attempts, and in such a case there are an expected $1/p$ such attempts. This yields an expected number of rewindings of 1. We now analyze the number of times that $\mathcal{S}_2$ is expected to have to begin from scratch (due to there being no t for which $r_t = 0$ and $\tilde{r}_t = 1$). The main observation here is that for any pair r and $\tilde{r}$ which forces $\mathcal{S}_2$ to begin from scratch, interchanging r and $\tilde{r}$ would result in a pair for which $\mathcal{S}_2$ would be able to continue. Now, since r and $\tilde{r}$ are derived through independent executions of the coin tossing phase, the probability that they are in one order *equals* the probability that they are in the opposite order. Thus, the probability that $\mathcal{S}_2$ needs to start from scratch equals at most $1/2$. This implies that the expected number of times that $\mathcal{S}_2$ needs to start from scratch is at most two. We remark that when $\mathcal{S}_2$ starts from scratch, the expected number of times it needs to rewind in order to obtain each of r and $\tilde{r}$ is $1/p$. Thus, overall the expected number of rewinding attempts is $p \cdot \mathcal{O}(1)/p = \mathcal{O}(1)$. We conclude that the overall expected running time of $\mathcal{S}_2$ is polynomial, as required. ■

Efficiency. The complexity of the protocol is in the order of ℓ times the basic protocol of [23]. Thus, the efficiency depends strongly on the value of ℓ that is taken. It is important to notice that the simulation succeeds except with probability $\approx 2^{-\ell+1}$ (as long as the cryptographic primitives are not "broken"). To be more exact, one should take ℓ and n so that the probability of "breaking" the cryptographic primitives (the commitments for the coin tossing or the security of encryption) is at most $2^{-\ell+1}$. In such a case, our analysis in the proof shows that the ideal and real executions can be distinguished with probability at most $2^{-\ell+2}$. This means that ℓ can be chosen to be relatively small, depending on the level of security desired. Specifically, with $\ell = 30$ the probability of successful undetected cheating is $2^{-28} \approx 3.7 \times 10^{-9}$ which is already very very small. Thus, it is reasonable to say that the complexity of the protocol is between 30 and 40 times of that of [23]. This is a non-trivial price; however, this is far more efficient than known solutions. We also remark that a similar idea can be used to achieve security in the model of covert adversaries of [3]. For deterrent factor $\epsilon = 1/2$ one can use $\ell = 2$ and have the sender choose r singlehandedly with one bit of r equalling 0 and the other equalling 1. This yields very high efficiency, together with simulatability (albeit in the weaker model of covert adversaries).

4 Oblivious Transfer using Smooth Hashing

The protocol of [23] was generalized by [17] via the notion of smooth projective hashing of [8]. This enables the construction of oblivious transfer protocols that are analogous to [23] under the Nth residuosity and quadratic residuosity assumptions. Protocol 1 can be extended directly in the same way, yielding oblivious transfer protocols that are secure against malicious adversaries, under the Nth residuosity and quadratic residuosity assumptions. We remark that as in the protocol of [17], the instantiation of the protocol under the Nth residuosity assumption is highly efficient, whereas the instantiation under the quadratic residuosity assumption enables the exchange of a single bit only (but is based on a longer-standing hardness assumption). We remark, however, that using Elliptic curves, the solution based on the DDH assumption is by far the most efficient.

5 Oblivious Transfer from Homomorphic Encryption

In this section, we present a protocol based on the protocol of [1] that uses homomorphic encryption. We assume an additive homomorphic encryption scheme (G, E, D), where $G(1^n)$ outputs a key-pair of length n, E is the encryption algorithm and D the decryption algorithm. Note that additive homomorphic operations imply multiplication by a scalar as well. The ideas behind this protocol are similar to above, and our presentation is therefore rather brief.

Protocol 2

- **Input:** *The sender has a pair of strings* (m_0, m_1) *of known length and the receiver has a bit* σ. *Both parties have a security parameter* n *determining the length of the keys for the encryption scheme, and a separate statistical security parameter* ℓ.
- **The protocol:**
  1. Receiver's message:
     (a) *The receiver* P_2 *chooses a key-pair* $(pk, sk) \leftarrow G(1^n)$ *from a homomorphic encryption scheme* (G, E, D).[4]
     (b) *For* $i = 1, \ldots, \ell$, *party* P_2 *chooses a random bit* $b_i \in_R \{0, 1\}$ *and defines*
     $$c_i^{b_i} = E_{pk}(0; r_i^{b_i}) \quad \text{and} \quad c_i^{1-b_i} = E_{pk}(1; r_i^{1-b_i}) \ .$$
     where r_i^0 *and* r_i^1 *are random strings, and* $E_{pk}(x; r)$ *denotes an encryption of message* x *using random coins* r.
     (c) P_2 *sends* $pk, \langle c_1^0, c_1^1, \ldots, c_\ell^0, c_\ell^1 \rangle$ *to* P_1.
  2. Coin tossing:
     (a) P_1 *chooses a random* $\tilde{s} \in_R \{0, 1\}^\ell$ *and sends* $\mathsf{Com}_h(\tilde{s})$ *to* P_2.
     (b) P_2 *chooses a random* $\hat{s} \in_R \{0, 1\}^\ell$ *and sends* $\mathsf{Com}_b(\hat{s})$ *to* P_1.
     (c) P_1 *and* P_2 *send decommitments to* $\mathsf{Com}_h(\tilde{s})$ *and* $\mathsf{Com}_b(\hat{s})$, *respectively, and set* $s = \tilde{s} \oplus \hat{s}$. *Denote* $s = s_1, \ldots, s_\ell$. *Furthermore let* S_1 *be the set of all* i *for which* $s_i = 1$, *and let* S_0 *be the set of all* j *for which* $s_j = 0$. *(Note that* S_1, S_0 *are a partition of* $\{1, \ldots, \ell\}$.*)*
  3. Receiver's message:
     (a) *For every* $i \in S_1$, *party* P_2 *sends the randomness* r_i^0, r_i^1 *used to encrypt* c_i^0 *and* c_i^1.
     (b) *In addition, for every* $j \in S_0$, *party* P_2 *sends a bit* β_j *so that if* $\sigma = 0$ *then* $\beta_j = b_j$, *and if* $\sigma = 1$ *then* $\beta_j = 1 - b_j$.
  4. Sender's message:
     (a) *For every* $i \in S_1$, *party* P_1 *verifies that either* $c_i^0 = E_{pk}(0; r_i^0)$ *and* $c_i^1 = E_{pk}(1; r_i^1)$, *or* $c_i^0 = E_{pk}(1; r_i^0)$ *and* $c_i^1 = E_{pk}(0; r_i^1)$. *That is,* P_1 *verifies that in every pair, one ciphertext is an encryption of* 0 *and the other is an encryption of* 1. *If this does not hold for every such* i, *party* P_1 *halts. If it does hold, it proceeds to the next step.*
     (b) *For every* $j \in S_0$, *party* P_1 *defines* c_j *and* c_j' *as follows:*
        i. *If* $\beta_i = 0$ *then* $c_j = c_j^0$ *and* $c_j' = c_j^1$.
        ii. *If* $\beta_i = 1$ *then* $c_j = c_j^1$ *and* $c_j' = c_j^0$.

[4]We assume that it is possible to verify that a public-key pk is in the range of the key generation algorithm G. If this is not the case, then a zero-knowledge proof of this fact must be added.

This implies that if $\sigma = 0$ then $c_j = E_{pk}(0)$ and $c'_j = E_{pk}(1)$, and if $\sigma = 1$ then $c_j = E_{pk}(1)$ and $c'_j = E_{pk}(0)$.[5]

(c) *For every $j \in S_0$, party P_1 chooses random ρ_j, ρ'_j, uniformly distributed in the group defined by the encryption scheme. Then, P_1 uses the homomorphic properties of the encryption scheme to compute:*

$$c_0 = \left(\sum_{j \in S_1} \rho_j \cdot c_j\right) + E_{pk}(m_0) \quad \text{and} \quad c_1 = \left(\sum_{j \in S_1} \rho'_j \cdot c'_j\right) + E_{pk}(m_1)$$

where addition above denotes the homomorphic addition of ciphertexts and multiplication denotes multiplication by a scalar (again using the homomorphic properties).

(d) *P_1 sends (c_0, c_1) to P_2.*

5. Receiver computes output: *P_2 outputs $D_{sk}(c_\sigma)$ and halts.*

Before discussing security, we demonstrate correctness:

1. Case $\sigma = 0$: In this case, as described in Footnote 5, it holds that for every j, $c_j = E_{pk}(0)$ and $c'_j = E_{pk}(1)$. Noting that the multiplication of 0 by a scalar equals 0, we have:

$$c_0 = \left(\sum_{j \in S_1} \rho_j \cdot c_j\right) + E_{pk}(m_0) = E_{pk}(0) + E_{pk}(m_0) = E_{pk}(m_0).$$

   Thus, when P_2 decrypts c_0 it receives m_0, as required.

2. Case $\sigma = 1$: In this case, it holds that for every j, $c_j = E_{pk}(1)$ and $c'_j = E_{pk}(0)$. Thus, similarly to before,

$$c_1 = \cdot \left(\sum_{j} \rho'_j \cdot c'_j\right) + E_{pk}(m_1) = E_{pk}(0) + E_{pk}(m_1) = E_{pk}(m_1),$$

   and so when P_2 decrypts c_1, it receives m_1, as required.

We have the following theorem:

Theorem 2 *Assume that (G, E, D) is a secure homomorphic encryption scheme, Com_h is a perfectly-hiding commitment scheme and Com_b is a perfectly-biding commitment scheme. Then, Protocol 2 securely computes the oblivious transfer functionality in the presence of malicious adversaries.*

Proof (sketch): In the case that P_2 is corrupted, the simulator works by rewinding the corrupted P_2 over the coin tossing phase in order to obtain two different openings and reorderings. In this way, the simulator can easily derive the value of P_2's input σ (σ is taken to be 0 if all the c_j ciphertexts for which it obtained both reorderings and openings are encryptions of 0, and is taken to be 1 otherwise). It sends σ to the trusted party and receives back $m = m_\sigma$. Finally, the simulator generates c_σ as the honest party P_1 would (using m), and generates $c_{1-\sigma}$ as an encryption to a random string. Beyond a negligible fail probability in obtaining the two openings mentioned, the

[5] In order to see this, note that if $\sigma = 0$ then $\beta_j = b_j$. Thus, if $\beta_j = b_j = 0$ we have that $c_j = c_j^0 = E_{pk}(0)$ and $c'_j = c_j^1 = E_{pk}(1)$. In contrast, if $\beta_j = b_j = 1$ then $c_j = c_j^1 = E_{pk}(0)$ and $c'_j = c_j^0 = E_{pk}(1)$. That is, in all cases of $\sigma = 0$ it holds that $c_j = E_{pk}(0)$ and $c'_j = E_{pk}(1)$. Analogously, if $\sigma = 1$ the reverse holds.

only difference with respect to a corrupted P_2's view is the way $c_{1-\sigma}$ is generated. However, notice that:

$$c_{1-\sigma} = \left(\sum_{j\in S_1} \hat{\rho}_j \cdot \hat{c}_j\right) + E_{pk}(m_{1-\sigma})$$

where $\hat{\rho}_j = \rho_j$ and $\hat{c}_j = c_j$, or $\hat{\rho}_j = \rho'_j$ and $\hat{c}_j = c'_j$, depending on the value of σ. Now, if at least one value $\hat{c}_j$ for $j \in S_1$ is an encryption of 1, then the ciphertext $c_{1-\sigma}$ is an encryption of a uniformly distributed value (in the group defined by the homomorphic encryption scheme). This is due to the fact that $\hat{c}_j$ is multiplied by $\hat{\rho}_j$ which is uniformly distributed. Now, by the cut-and-choose technique employed, the probability that for all $j \in S_1$ it holds that $\hat{c}_j \neq E_{pk}(1)$ is negligible. This is due to the fact that this can only hold if for *many* ciphertext pairs c_i^0, c_i^1 sent by P_2 in its first message, the pair is *not* correctly generated (i.e., it is not the case that one is an encryption of 0 and the other an encryption of 1). However, if this is the case, then P_1 will abort except with negligible probability, because S_0 will almost certainly contain one of these pairs (and the sets S_0 and S_1 are chosen as a random partition based on the value s output from the coin tossing).

In the case that P_1 is corrupted, the simulator manipulates the coin tossing so that in the unopened pairs of encryptions, all of the ciphertexts encrypt 0. This implies that both $\left(\sum_{j\in S_1} \rho_j \cdot c_j\right) = E_{pk}(0)$ and $\left(\sum_{j\in S_1} \rho'_j \cdot c'_j\right) = E_{pk}(0)$, in turn implying that $c_0 = E_{pk}(m_0)$ and $c_1 = E_{pk}(m_1)$. Thus, the simulator obtains both m_0 and m_1 and sends them to the trusted party. This completes the proof sketch. A full proof follows from the proof of security for Protocol 1. ■

6 Acknowledgements

We would like to thank Nigel Smart for helpful discussions and Benny Pinkas for pointing out an error in a previous version.

References

[1] W. Aiello, Y. Ishai and O. Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *EUROCRYPT 2001*, Springer-Verlag (LNCS 2045), pages 119–135, 2001.

[2] G. Aggarwal, N. Mishra and B. Pinkas. Secure Computation of the k th-Ranked Element. In *EUROCRYPT 2004*, Springer-Verlag (LNCS 3027), pages 40–55, 2004.

[3] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In the *4th TCC*, Springer-Verlag (LNCS 4392), pages 137–156, 2007.

[4] B. Barak and Y. Lindell. Strict Polynomial-Time in Simulation and Extraction. *SIAM Journal on Computing,* 33(4):783–818, 2004.

[5] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[6] J. Camenisch, G. Neven and A. Shelat. Simulatable Adaptive Oblivious Transfer. In *EUROCRYPT 2007*, Springer-Verlag (LNCS 4515), pages 573–590, 2007.

[7] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[8] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *EUROCRYPT 2002*, Springer-Verlag (LNCS 2332), pages 45–64, 2002.

[9] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk and T. Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In *CRYPTO 2004*, Springer-Verlag (LNCS 3152), pages 494–510, 2004.

[10] S.D. Galbraith, K.G. Paterson and N.P. Smart. Pairings for Cryptographers. *Cryptology ePrint Archive* Report 2006/165, 2006.

[11] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. In *Communications of the ACM,* 28(6):637–647, 1985.

[12] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

[13] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[14] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.

[15] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987. For details see [12].

[16] M. Green and S. Hohenberger. Blind Identity-Based Encryption and Simulatable Oblivious Transfer. In *Asiacrypt 2007*, Springer-Verlag (LNCS 4833), pages 265–282, 2007.

[17] Y.T. Kalai. Smooth Projective Hashing and Two-Message Oblivious Transfer. In *EUROCRYPT 2005*, Springer-Verlag (LNCS 3494), pages 78–95, 2005.

[18] J. Kilian. Founding Cryptograph on Oblivious Transfer. In 20*th STOC*, pages 20–31, 1988.

[19] E. Kushilevitz and R. Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In 38*th FOCS*, pages 364–373, 1997.

[20] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT 2007*, Springer-Verlag (LNCS 4515), pages 52–78, 2007.

[21] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[22] M. Mitzenmacher and E. Upfal. *Probability and Computing.* Cambridge University Press, 2005.

[23] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In 12*th SODA*, pages 448–457, 2001.

[24] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[25] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.