Efficient and Universally Composable Protocols for Oblivious Transfer from the CDH Assumption

Eduard Hauck [1] Julian Loss [1]

October 17, 2017

[1] Ruhr University Bochum
{eduard.hauck,julian.loss}@rub.de

Abstract

Oblivious Transfer (OT) is a simple, yet fundamental primitive which suffices to achieve almost every cryptographic application. In a recent work (Latincrypt '15), Chou and Orlandi (CO) present the most efficient, fully UC-secure OT protocol to date and argue its security under the CDH assumption. Unfortunately, a subsequent work by Genc et al. (Eprint '17) exposes a flaw in their proof which renders the CO protocol insecure. In this work, we make the following contributions: We first point out two additional, previously undiscovered flaws in the CO protocol and then show how to patch the proof with respect to static and malicious corruptions in the UC model under the stronger Gap Diffie-Hellman (GDH) assumption. With the proof failing for adaptive corruptions even under the GDH assumption, we then present a novel OT protocol which builds on ideas from the CO protocol and can be proven fully UC-secure *under the CDH assumption*. Interestingly, our new protocol is actually significantly more efficient (roughly by a factor of two) than the CO protocol. This improvement is made possible by avoiding costly redundancy in the symmetric encryption scheme used in the CO protocol. Our ideas can also be applied to the original CO protocol, which yields a similar gain in efficiency.

Keywords: Oblivious Transfer, Universally Composable Security

1 Introduction

In multi-party computation (MPC), a set of n mutually distrustful parties engages in a distributed protocol to evaluate a function f in such a way that every honest party learns the output of f, but no malicious party learns anything about the inputs of the honest parties. MPC is one of the oldest and most well-studied fields within in cryptography, as it allows to securely implement a wide array of cryptographic applications. Recently, there has been a particular interest in so-called two-party computation (2PC) protocols which consider the important case of $n = 2$, i.e., two parties jointly running an MPC protocol. Notable works on 2PC

include [Yao82, DN00, Nie07, Lin03, HK07, Lin09]. An underlying primitive of great importance to 2PC is *oblivious transfer* (OT). In OT, two parties, a sender and a receiver, engage in a protocol that shall satisfy the following properties: The sender holds n messages $(m_0, ..., m_{n-1})$ and wishes to transfer exactly one of them to the receiver. The receiver chooses the message m_i that it wants to learn by submitting the index i to the protocol. The protocol is deemed secure if the receiver learns nothing about the messages $m_{j\neq i}$ and the sender does not learn the value i. OT was introduced independently by Rabin [Rab81] and Wiesner [Wie83] and has since been extensively studied [Kil88, Cré87, NP01, Bea98]. Indeed, one can show that many cryptographic tasks can be reduced (via MPC) to the apparently simple primitive of OT [Kil88, IPS08].
Since most protocols for MPC and 2PC that call an OT protocol in a black-box fashion require many calls to this protocol, there has recently been a significant interest in the design of efficient OT extension protocols [Bea96a], where a large number of OT executions can be obtained from a smaller number of "base" OT executions and symmetric key operations. Examples are [Nie07, Lar14, ALSZ16, KOS15]. Making these base OTs as efficient as possible presents a strong motivation for efficient OT protools. Further, in any scenario where parties want to perform 2PC (or more generally, MPC) between a large number of *distinct* parties, each pair of parties has to run an expensive OT extension protocol (including between 128 and 180 base OTs). A second issue is that while some OT extension protocols offer security against malicious adversaries, in all known OT extension protocols, the receiver must commit to its vector of choice values at the beginning of the protocol. However, in many cases, one is actually interested in OT protocols that remain secure even when the receiver may choose what messages it wants to obtain from the sender in an *adaptive fashion*. Motivated by the importance of OT, we propose in this work the most efficient, fully UC-secure OT protocol to date.

1.1 Existing Protocols and Our Contribution

Existing Protocols. The most efficient protocol (OT_{co}) meeting the (now common) standard of full UC-security [Can01] was recently proposed by CO [CO15]. For m executions, it requires only $2 + m$ modular exponentiations for the sender and $2m$ modular exponentiations for the receiver. The communication complexity of the protocol is $32(m + 1)$ bytes for the group elements that need to be sent. Additionally, mn ciphertexts must be transmitted. The security of their protocol is based on the CDH assumption. [CO15] also give a highly optimized implementation of their protocol which can be found at http://orlandi.dk/simpleOT. Unfortunately, a subsequent work by Genc et al. [GIR17] uncovers a flaw in the proof of CO which renders the protocol insecure.

Our Contribution. The outline of our paper is as follows. We begin by stating two additional flaws in OT_{co} that were not noticed in [GIR17]. As a warm-up, we show how, using the Gap DH (GDH) assumption, we can patch OT_{co} to satisfy security against static and malicious corruptions in the UC model. However, even under the GDH assumption, we are not able to prove OT_{co} secure for fully adaptive corruptions that may occur at any point during the protocol execution. As our main contribution, we then present a novel OT protocol OT^* which builds on ideas from OT_{co}, but can be proven fully UC-secure under the CDH

assumption. Surprisingly, we find that OT^* avoids redundancy in the ciphertexts which is necessary for the original proof in [CO15]. This greatly improves the efficiency of our protocol over OT_{co} since its main bottleneck stems from the mn ciphertexts that need to be sent. We improve the (asymptotic) communication complexity over OT_{co} by a factor of at least two, depending on the message length ℓ and security parameter λ. Concretely (again, for m executions of our protocol), we improve the complexity from $256(m+1)+nm(\ell+\lambda)$ bits to $512m+nm\ell$ bits. For the interesting case of single bit messages, our protocol offers an asymptotic improvement by a factor of λ (say, $\lambda = 128$) in communication complexity over OT_{co}. With the proof in [CO15] being broken, our protocol is the most efficient fully UC-secure 1-out-of-n OT protocol, while requiring only the CDH assumption. As an independent contribution, we believe that our thorough and detailed security analysis in the UC model may serve as a roadmap to future endeavours in designing UC-secure OT protocols.

Our Techniques. At a high level, the problem in the proof for fully adaptive corruptions in OT_{co} stems from the fact that in an adaptive corruption scenario, the internal state of the corrupted party must be simulated and output to the environment. The structure of OT_{co} prohibits the efficient computation of the correct state in the special case where the sender is corrupted at the beginning of the protocol and the receiver is corrupted after the protocol has been completed. More precisely, in OT_{co}, a statically corrupted sender may choose its first round message as group element $\mathbf{S} = g^y$ where y is not known to the simulator. Unfortunately, the simulator only learns which message was chosen by the receiver once it becomes corrupted. On the other hand, it must be able to forge a state (of the receiver) which matches previously simulated messages, regardless of the receivers choice, i.e., it must be able to equivocate the choice of the receiver once it becomes corrupted. In OT_{co}, this requires knowledge of y. We further elaborate on these issues in Section 3. Our fix to this problem is to add a new random oracle G to OT_{co} which is called by both parties on the element $\mathbf{S}$, resulting in output $\mathbf{T} = g^t$, i.e., $\mathbf{T} = \mathsf{G}(\mathbf{S})$. We then carefully restructure OT_{co} so as to include $\mathbf{T}$, resulting in our new protocol OT^*. Since the simulator controls the random oracle, it knows the value of t and can now resolve the above issue of equivocation.

1.2 Related Work

Previous works examine OT protocols in various security models and setup assumptions. In the MPC setting, it is desirable to consider models which allow to reason about the security of protocols when composed with themselves or other protocols in arbitrary concurrent or sequential executions. Examples are [GH08, RKP09, Gar04, DNO09, PVW07]. Before the notion of UC security was established, various different approaches were followed. Beaver proves in [Bea96b, Bea98] the security of an oblivious transfer protocol in a full-simulation proof against adaptive adversaries, which resembles a proof in the UC model against adaptive adversaries. To achieve adaptive security, Beaver defines the requirement of the OT being "content-equivocable", which is closely related to the notion of "non-committing encryption" (NCE) we use in this work.

For OT_{co} and OT^*, we require NCE for a secret key encryption scheme. In [CFGN96, CLOS02], NCE and similar notions are required from public key

encryption schemes. Notably, in the public key setting, achieving NCE is much costlier than for the symmetric setting. Garay et al. [GWZ09] address this problem by introducing the new notion *somewhat non-committing encryption* (SNCE), where the opening of a ciphertext is limited to a set possible decryptions. They show how to use this notion to construct fully UC-secure OTs from statically UC-secure OTs.

Choi et al. give [CDMW09] a compiler from semi-honest UC-secure OT to fully UC-secure OT. Abdalla et al. [ABB+13] build UC-secure OT from Smooth projective hash function friendly non interactive-commitments. Blazy et al. [BC15] give an generic compiler for UC-secure OT from a collision-resistant chameleon hash scheme and a CCA encryption scheme accepting a smooth projective hash function. Blazy et al. [BC16] show how to construct UC-secure OT from Structure-Preserving Smooth Projective Hashing. Finally, Blazy et al. [BCG17] build UC-secure OT from Quasi-Adaptive Non-Interactive Zero-Knowledge Proofs.

Some OT protocols assume the secure erasure of data. In the secure erasure model it is allowed to delete subsets of the state of a party, such that in the case of adaptive corruptions, the adversary is not able to learn the full state of the corrupted party. Garay [Gar04] proposes an OT protocol which is secure assuming secure erasure in the UC Model under the strong RSA assumption or the DSA assumption. Choi et al. [CKWZ13] propose UC-secure OT under the DLIN or SXDH assumption under a single, global CRS in the erasure model. Prior to Abdalla et al.'s result [ABP17], the assumption of secure erasure was avoided, due to being hard to guarantee.

Figure 1.2 depicts a comparison of some known OT protocols with our new protocol OT^*. Protocols OT_{co}, OT^*, the combination of protocol [PVW08] with the compiler [GWZ09] and [BC15] achieve full UC security. [NP01] achieves one sided security against a static adversary (however, not in the UC model) and [PVW08] achieves security against static corruptions with maliciously behaving corrupted parties in the UC model. The combination of protocol [PVW08] with the compiler [GWZ09] and [BC15] are proven secure in the CRS model.

Protocol	model	assumption	computational cost	communication cost/(in bits)
m times $\binom{n}{1}$				
[NP01]	ROM	DDH	$(n+2m, nm, 2m, nm+m)$	$(n+m, nm\ell)/256(n+m)+nm\ell$
OT_{co}	ROM, AC	GDH	$(3m+2, nm+2m+n-4, 1, (n+1)m)$	$(m+1, nm(\ell+\lambda))/256(m+1)+nm(\ell+\lambda)$
OT^*	ROM, AC	CDH	$(5m, m^2+2mn-4m, m, (n+3)m)$	$(2m, nm\ell)/512m+nm\ell$
One time $\binom{n}{1}$				
[PVW08]+ [GWZ09]	CRS	DDH, DCR	$(\mathcal{O}(n), \mathcal{O}(n), \mathcal{O}(n), \mathcal{O}(n))$	$(\mathcal{O}(n), \mathcal{O}(n\lambda))$
[BC15]	CRS	DDH	$(\mathcal{O}(n), \mathcal{O}(n), \mathcal{O}(n), \mathcal{O}(n))$	$(n+9\log_2(n)+4, -)/256n+2304\log_2(n)+4$
OT_{co}	ROM, AC	GDH	$(5, 2n-2, 1, n+1)$	$(2, n\ell)/256(n+1)+n(\ell+\lambda)$
OT^*	ROM, AC	CDH	$(5, 3n-4, 1, n+3)$	$(2, n\ell)/512+n\ell$
One time $\binom{2}{1}$				
[PVW08]	CRS	DDH	$(14, 7, 1)$	$(8, -)/2048$
[NP01]	ROM	DDH	$(5, 2, 2, 3)$	$(2, 2\ell)/514+\log_2(n)\ell$
OT_{co}	ROM, AC	GDH	$(5, 3, 1, 3)$	$(2, 2(\ell+\lambda))/512+2(\ell+\lambda)$
OT^*	ROM, AC	CDH	$(5, 3, 1, 5)$	$(2, 2\ell)/512+2\ell$

Figure 1: Overview of OT protocols. The abbreviations in column **model** denote the following: AC = Authenticated Channels, CRS = Common Reference String , ROM=Random Oracle Model. In columns **computational cost** and **communication cost** the cost is grouped in tuples as follows: (#modular exponentiations, #multiplications, #modular inversions, #hashes) and (#group elements, #ciphertexts), respectively. The communication cost in bits is computed for $\lambda = 128$ and $\mathcal{M} = \{0,1\}^\ell$.

Concurrent Work. In work concurrent and independent to our own, Barreto et al. [BDD+17] also show an OT protocol that meets UC security under adaptive corruptions and under the CDH assumption. Interestingly, their protocol follows from a more general framework for UC-secure OT and therefore uses an approach completely different from our own. However, the protocol of [BDD+17] requires a much higher amount of group exponentiations than OT^* and also has a significantly higher communication complexity. More precisely, to perform a $\binom{n}{1}$-OT, they need to perform $O(n)$ exponentiations, whereas OT^* needs to perform only a total of 5 exponentiations and $O(n)$ *group multiplications* which are much more efficient. Furthermore, their protocol needs to communicate n additional group elements (one per message held by the sender) compared to our protocol, in which only 2 group elements need to be exchanged regardless of n. Finally, it appears that the construction in [BDD+17] requires an ElGamal-type encryption scheme which decrypts either to a unique value or rejects (except with negligible probability). [BDD+17] does not elaborate on how exactly to implement this feature of their encryption scheme. An easy way to achieve this would be to add a hash of the encrypted message to the ciphertext. However, this would add even further additional communication costs to the scheme of [BDD+17].

2 Preliminaries

2.1 Notation

Algorithms. Let λ denote the security parameter. We say that an algorithm is a PPT algorithm, if it runs in probabilistic polynomial time (in λ). We assume implicitly that any algorithm receives the unary representation 1^λ of the security parameter as input as its first argument. We denote by $s \xleftarrow{\$} S$ the uniform sampling of the variable s from the (finite) set S. All our algorithms are (unless stated otherwise) probabilistic and written in uppercase letters A, B. To indicate that algorithm A runs on some inputs $(x_1, x_2, ...)$ and returns y, we write $y \xleftarrow{\$} \mathsf{A}(x_1, x_2, ...)$. If additionally, A has access to an algorithm B (via oracle access) during its execution, we write $y \xleftarrow{\$} \mathsf{A}^{\mathsf{B}}(x_1, x_2, ...)$. We denote with $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{X})$ the hash query $\mathsf{H}(\mathbf{S}, \mathbf{R}_i, \mathbf{X})$. A group generating algorithm GGen is a PPT algorithm that outputs $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\lambda)$, where $\mathbb{G}$ is a cyclic group of prime order p and g is a generator of $\mathbb{G}$. All group elements, apart from the generator g, are written in uppercase letters $\mathbf{X}, \mathbf{Y}$.

Security Games. We use code-based *security games* [BR04]. In game $\mathbf{G}$, an adversary A interacts with a challenger that answers oracle queries issued by A. It has a main procedure and (possibly zero) oracle procedures which describe how oracle queries are answered. We denote the output of a game $\mathbf{G}$ between a challenger and an adversary A via $\mathbf{G}^{\mathsf{A}}$. A is said to *win* if $\mathbf{G}^{\mathsf{A}} = 1$. We define the *advantage* of A in $\mathbf{G}$ as $\mathbf{Adv}^{\mathbf{G}}_{\mathsf{A}} := \Pr[\mathbf{G}^{\mathsf{A}} = 1]$. We say that winning a game $\mathbf{G}$ is *hard* if for every PPT algorithm A, $\mathbf{Adv}^{\mathbf{G}}_{\mathsf{A}} = \mathsf{negl}(\lambda)$.

The Computational Diffie-Hellman Problem. We briefly recap the well-known Computational Diffie-Hellman (CDH) problem. The CDH Game **cdh** is depicted in Figure 2.

The Gap Diffie-Hellman Problem. Next, we introduce the so-called so called Gap Diffie-Hellman problem. In the Gap Diffie-Hellman Game **gap**

$\underline{\mathbf{cdh}}^{\mathsf{A}}$
00 $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(\lambda)$
01 $x, y \xleftarrow{\$} \mathbb{Z}_p$
02 $(\mathbf{X}, \mathbf{Y}) := (g^x, g^y)$
03 $\mathbf{Z} \xleftarrow{\$} \mathsf{A}(\mathbf{X}, \mathbf{Y})$
04 Return $\mathbf{Z} = g^{xy}$

Figure 2: Computational Diffie-Hellman Game **cdh**, relative to adversary A.

depicted in Figure 3, the adversary A must also solve a CDH challenge, but is additionally given access to an oracle DH that decides the DDH problem.

$\underline{\mathbf{gdh}}^{\mathsf{A}}$	$\mathsf{DH}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$:
00 $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(\lambda)$	05 Return $\mathbf{Z} = g^{xy}$
01 $x, y \xleftarrow{\$} \mathbb{Z}_p$	
02 $(\mathbf{X}, \mathbf{Y}) := (g^x, g^y)$	
03 $\mathbf{Z} \xleftarrow{\$} \mathsf{A}^{\mathsf{DH}(\cdot,\cdot)}(\mathbf{X}, \mathbf{Y})$	
04 Return $\mathbf{Z} = g^{xy}$	

Figure 3: Gap Diffie-Hellman Game **gdh**, relative to adversary A.

Properties of the Encryption Scheme. We briefly introduce the a relevant notion for the symmetric encryption scheme that we will be using in our proof.

Definition 2.1 (Symmetric Encryption Scheme). A symmetric encryption scheme is a triple $(\mathsf{E}, \mathsf{D}) = (\mathsf{KG}(\cdot), \mathsf{E}(\cdot,\cdot), \mathsf{D}(\cdot,\cdot))$. The *key generation algorithm* takes as input a security parameter 1^λ and outputs a key $k \xleftarrow{\$} \mathcal{K}$. The *encryption algorithm* takes as inputs a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$ and outputs a ciphertext $e \in \mathcal{C}$. The *decryption algorithm* $\mathsf{D}(\cdot,\cdot)$ takes as input a ciphertext $e \in \mathcal{C}$ and a key $k \in \mathcal{K}$ and outputs a message $M' \in \mathcal{M}$. For correctness we require that $\forall M \in \mathcal{M}, \forall k \in \mathcal{K} : \mathsf{D}(k, \mathsf{E}(k, M)) = M$. Furthermore, we require that any ciphertext $e \in \mathcal{C}$ can be decrypted with any key $k \in \mathcal{K}$, i.e., that $\mathsf{D}(e, k) \in \mathcal{M}$.

Definition 2.2 (Non-Committing). We say a *symmetric encryption scheme* (E, D) is *non-committing* if there exist PPT algorithms S_1, S_2, which are allowed to share a state, such that for all PPT algorithms A, $M \in \mathcal{M}$, and $k_0 \xleftarrow{\$} \mathcal{K}, e_0 \xleftarrow{\$} \mathsf{E}(k_0, M), e_1 \xleftarrow{\$} \mathsf{S}_1(1^\lambda), k_1 \xleftarrow{\$} \mathsf{S}_2(e_1, M)$, we have

$$\left| \Pr[b = b' \mid b' \xleftarrow{\$} \mathsf{A}(e_b, k_b)] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

Random Oracle Model. As the original work of [CO15], our results are stated in the random oracle model [BR93]. Concretely, we model the hash functions $\mathsf{H} : \mathbb{G}^3 \longrightarrow \mathcal{K}$ and $\mathsf{G} : \mathbb{G} \longrightarrow \mathbb{G}$ as random oracles.

The OT Protocol. Chou and Orlandi's (CO) protocol OT_{co} is depicted in Figure 1 (here, we write the protocol for an m-fold $\binom{n}{1}$-OT). From lines 1 to 6 random keys for each message the sender holds are generated. The receiver

learns a subset of size m keys and thus can decrypt a subset of size m of the ciphertexts sent in step 8.

	Sender Input: $(M_{0,0}, ..., M_{m-1,n-1})$ Output: none		Receiver Input: $(c_0, ..., c_{m-1})$ Output:$(M'_0, ..., M'_{m-1})$
1.	$y \xleftarrow{\$} \mathbb{Z}_p \quad \mathbf{S} = g^y \quad \mathbf{T} = g^{y^2}$		$x_i \xleftarrow{\$} \mathbb{Z}_p \quad \forall i \in [m]$
2.		$\xrightarrow{\mathbf{S}}$	if $\mathbf{S} \notin \mathbb{G}$: abort;
3.	if $\exists \mathbf{R}_i \notin \mathbb{G} : abort$	$\xleftarrow{\mathbf{R}_0,...,\mathbf{R}_{m-1}}$	$\forall i \in [m] : \mathbf{R}_i = \mathbf{S}^{c_i} g^{x_i}$
4.	$\forall i \in [m], \forall j \in [n]$:		$\forall i \in [m]$:
5.	$k_{i,j} = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\left(\mathbf{R}_i^y \mathbf{T}^{-j}\right)$		$k_{R_i} = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{S}^{x_i})$
6.	$= \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\left(g^{(c_i-j)y^2} g^{x_i y}\right)$		$= \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(g^{x_i y})$
7.	$e_{i,j} \leftarrow \mathsf{E}(k_{i,j}, M_{i,j})$		
8.		$\xrightarrow{(e_{0,0},...,e_{m-1,n-1})}$	
9.			$M'_i = \mathsf{D}(k_{R_i}, e_{i,c_i}) \quad \forall i \in [m]$

Table 1: Protocol OT_{co}

Efficiency. The computational cost of protocol OT_{co} for $m > 2, n > m$, is composed as follows. The sender performs 2 and m modular exponentiations in steps 1 and 5, respectively, $n - 2 + nm$ multiplications in step 5 and a single inversion in step 5. The receiver performs m modular exponentiations in steps 3 and 5, respectively and $2m - 2$ multiplications in step 3.
In total both the sender and receiver perform modular $3m + 2$ exponentiations, $nm + 2m + n - 4$ multiplications and one modular inversion.
The communication cost is composed of $m + 1$ group elements $(\mathbf{S}, \mathbf{R}_0, ..., \mathbf{R}_{m-1})$ and nm ciphertexts $(e_{0,0}, ..., e_{m-1,n-1})$.
Further in step 5 the sender performs nm queries to H and the receiver performs m queries to H. In contrast to COs proof of OT_{co} we do not require the encryption scheme to be robust (c.f. [CO15]). Therefore we cut down the bit complexity of each ciphertext by a factor of at least 2, depending on the size of the message space.

3 Shortcomings of COs Protocol

Protocol OT_{co} [CO15] contains three flaws which prohibit a successful proof in the Universal Composability Framework.

Flaw 1. The first flaw as pointed out in [GIR17] invalidates COs proof against a corrupted sender. Namely, in COs proof the simulator Sim does not correctly simulate the behavior of an honest receiver running the protocol in the real world against a malicious sender S^* that actively deviates from the protocol specification. Concretely, S^* can issue hash queries to a random oracle H programmed by the simulator Sim. The proof requires Sim to extract the secret messages $\vec{M} = (M_{0,0}, ..., M_{m-1,n-1})$ from corresponding ciphertexts $(e_{0,0}, ..., e_{m-1,n-1})$ obtained from S^*. In a real execution of the protocol, the ciphertext $e_{i,j}$ is the

result of an encryption under some key $k_{i,j}$, i.e., $e_{i,j} = \mathsf{E}(M_{i,j}, k_{i,j})$. Therefore, Sim has to extract the correct decryption key $k_{i,j}$ to decrypt $e_{i,j}$. Each key $k_{i,j}$ in the protocol specification is the result of a hash query to $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}), \mathbf{U} \in \mathbb{G}$. Using a special property called *robustness* (c.f. [CO15]), every ciphertext can be decrypted only under the correct key $k_{i,j}$ that was previously obtained from a call to H. By exhaustive testing, Sim can therefore determine the unique key $k_{i,j}$ that decrypts $e_{i,j}$ for all $i \in [n], j \in [m]$. Unfortunately, Sim can not distinguish queries of the form $k_{i,j} = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\big(\mathbf{R}_i^y \mathbf{T}^{-j}\big)$ from queries of the form $k' = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$, where $\forall i \in [n], j \in [m] : \mathbf{U} \neq \mathbf{R}_i^y \mathbf{T}^{-j}$. This allows a malicious adversary to encrypt $M_{i,j}$ with $k' = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$ instead of $k_{i,j}$ without Sim noticing. Sim then decrypts the ciphertext $e_{i,j}$ with the (unique) key k'. However, the honest receiver in the real world can not compute k' and thus is not able to decrypt the ciphertext $e_{i,j}$.

Resolution of Flaw 1. In order to resolve this issue, Sim must compute its decryption keys indistinguishable from the honest receiver. Therefore, Sim must be able to distinguish hash queries of the form $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\big(\mathbf{R}_i^y \mathbf{T}^{-j}\big)$ from queries of the form $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\big(\mathbf{U}\big)$, where $\forall i \in [n], j \in [m] : \mathbf{U} \neq \mathbf{R}_i^y \mathbf{T}^{-j}$. Orlandi [Orl17] suggests that the GDH assumption can be used to resolve this flaw. Indeed, we provide a revised proof under the GDH assumption in Section 4.

Flaw 2. The second flaw lies in the proof that no malicious receiver can learn more than m messages. CO argue that such an attacker can be used to break the CDH assumption. Unfortunately, their reduction proof is only sketched. They argue as follows.
Under the assumption that there exists a PPT adversary A engaging with a honest sender in the OT_{co} protocol and outputs at least $m+1$ correct messages at the end of the protocol, CO show how to construct a PPT algorithm B, which wins the **cdh** game. They argue that B can extract the solution to any given CDH challenge from A, by running A three times. In each run, A queries H, by assumption, on at least two queries of the form: $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_0)$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_1)$, where $\mathbf{U}_0 = \mathbf{R}_i^y \mathbf{T}^{-j_0}$ and $\mathbf{U}_1 = \mathbf{R}_i^y \mathbf{T}^{-j_1}$, for some $i \in [m], j_0 \neq j_1 \in [n]$. However, they do not provide an explanation of how algorithm B simulates an honest sender to A. Essential to simulating, B has to provide an efficient simulation of random oracle H to A. In particular, queries $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{R}_i^y \mathbf{T}^{-j})$ must be answered with keys $k_{i,j}$ such that $\mathsf{D}(k_{i,j}, e_{i,j}) = M_{i,j}$. Again, Sim can not distinguish queries of the form $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}\big(\mathbf{R}_i^y \mathbf{T}^{-j}\big)$ from queries of the form $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$, where $\forall i \in [n], j \in [m] : \mathbf{U} \neq \mathbf{R}_i^y \mathbf{T}^{-j}$. Therefore, B has to guess for each of the m queries $(\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_0), ..., \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_{m-1}))$, the value $j \in [n]$ such that $\mathbf{U}_l = \mathbf{R}_i^y \mathbf{T}^{-j}$ (it needs to do so for each $l \in [m]$). If m runs of the protocol are executed in parallel, this incurs a multiplicative loss of at least $\frac{1}{n^m}$.

Resolution of Flaw 2. We resolve this flaw in Lemma 5.2, by proving security under the stronger GDH assumption.

Flaw 3. The third flaw lies in the proof that the protocol remains secure even if the receiver is adaptively corrupted. In the case of an adaptively corrupted party, the simulator must forge, upon corruption of the party, a state of the corrupted party. This state must be forged, such that the environment machine can not distinguish whether the provided state comes from the honest receiver which has knowledge of the receivers input before the execution, or from the simulator which learns the input upon corruption. Further, the simulator must

be able to forge such a state, even if the sender is statically corrupted and the receiver is corrupted after the execution finished.
CO do not show how to create an internal state of the receiver for the latter case. More precisely, they do not show how to compute a value x_i such that the previously sent values $\mathbf{R}_i$ can be explained as $\mathbf{R}_i = \mathbf{S}^{c_i} g^{x_i}$. Here, $\mathbf{S} = g^y$ is the first round message sent by the statically corrupted sender and therefore, the simulator does not know y. The problem with the simulator of CO is that it samples $x_i' \xleftarrow{\$} \mathbb{Z}_p$ and then outputs $\mathbf{R}_i = g^{x_i'}$ as the second round message of the protocol before it learns c_i (recall that it only learns c_i upon corruption of the receiver, after the protocol is completed). Now, it must be able to explain $\mathbf{R}_i$ as $\mathbf{R}_i = \mathbf{S}^{c_i} g^{x_i}$, for some x_i. This means, the simulator must compute x_i such that $x_i' = yc_i + x_i$. However, from this equation, we can easily recover y as $y = (x_i' - x_i)/c_i$, which means that our adapted simulator could be used to break the discrete logarithm assumption.

Handling of Flaw 3. Unfortunately we do not see a possibility of resolving Flaw 3 while keeping OT_{co} unchanged. We provide a proof for a weaker security notion (static corruptions) in Theorem 4.3. However, as our main contribution, we mend the aforementioned problem by adding a random oracle G to the protocol description and thereby attaining greater flexibility in our security proof. This is achieved by having both parties compute group element $\mathbf{T} = g^t$ as $\mathbf{T} = \mathsf{G}(\mathbf{S})$. Our simulator now makes extensive use of the fact that by programming G at point $\mathbf{S}$, it knows the value t. We describe our approach in detail in Section 5.

4 Revised Proof of COs Protocol

In this section we review the security proof of OT_{co} and prove security of OT_{co} for static, malicious corruptions under the Gap Diffie-Hellman assumption.

Lemma 4.1 (Sender can not extract receivers choice values). *No (computationally unbounded)* S^* *on input* R_i*, can guess* c_i *with probability greater than* $1/n$*.*

Proof. Same proof as in COs paper [CO15]. ∎

Lemma 4.2 (Receiver can not extract more than m messages). *In the random oracle model, no PPT algorithm* A*, engaging in the role of the receiver in an execution of* OT_{co}*, can output* $m+1$ *or more messages if the Gap Diffie-Hellman Game* **gdh** *is hard to win. Moreover, if there exists such a PPT algorithm* A *with advantage* ϵ *in outputting at least* $m+1$ *messages then there exists an PPT algorithm* B *with advantage* ϵ^3 *in winning the* **gdh** *game.*

Proof. Let A be an adversary which engages with a honest sender in the OT_{co} protocol and outputs at least $m+1$ correct messages at the end of the protocol. We show how to construct an adversary B, which uses A as a subroutine, to break the **gdh** assumption. B receives as input a GDH challenge $(\mathbf{A} = g^a, \mathbf{B} = g^b) \xleftarrow{\$} \mathbb{G}$ and runs A three times.
To simulate the first step of the protocol, B sends in the first run $\mathbf{S} = \mathbf{A}$, in the second run $\mathbf{S} = \mathbf{B}$ and in the third run $\mathbf{S} = \mathbf{AB}$. We show below how B simulates the first run of A, the second and the third run follow in a similar fashion.

To answer queries to H, B does the following. Before B learns the values $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$, B returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$, where $\mathbf{V}, \mathbf{W}, \mathbf{U} \in \mathbb{G}$. Once A returns group elements $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$, B computes $\forall i \in [m], j \in [m] : e_{i,j} \leftarrow \mathsf{S}_1(1^\lambda)$. B aborts if $\mathbf{R}_i \notin \mathbb{G}$ for any group element $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$.
To make the previously answered hash queries consistent with the encryptions, B checks for each previous hash query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, whether $\mathbf{V} = \mathbf{S}$ and there exists an $i \in [m]$ such that $\mathbf{W} = \mathbf{R}_i$. If so, B tests whether $\mathsf{DH}(\mathbf{A}, \mathbf{R}_i\mathbf{S}^{-j}, \mathbf{U}) = 1$ for any $j \in [n]$. If B finds such a j then B overwrites $e_{i,j} = \mathsf{E}(\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}), M_{i,j})$. Then B sends $(e_{0,0}, ..., e_{n-1,m-1})$ to A.
To simulate all future hash queries $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$ correctly, B tests whether $\mathsf{DH}(\mathbf{A}, \mathbf{R}_i\mathbf{S}^{-j}, \mathbf{U}) = 1$ for any $j \in [n]$. If the latter is true for some index j then B returns $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}) = \mathsf{S}_2(e_{i,j}, M_{i,j})$, else B returns a random group element.
By assumption, A outputs $m+1$ messages. Therefore, A has to perform $m+1$ distinct queries $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$, with $\mathsf{DH}(\mathbf{A}, \mathbf{R}_i\mathbf{S}^{-j}, \mathbf{U}) = 1$. Since, there are only m values $\mathbf{R}_i$, there will be two hash queries $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_0)$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_1)$ such that $\mathsf{DH}(\mathbf{A}, \mathbf{R}_i\mathbf{S}^{-j_0}, \mathbf{U}_0) = 1$ and $\mathsf{DH}(\mathbf{A}, \mathbf{R}_i\mathbf{S}^{-j_1}, \mathbf{U}_1) = 1$. Once B observes two such queries, it saves the group elements $(\mathbf{U}_0,\ \mathbf{U}_1)$ and indices (j_0, j_1) and aborts the execution of A. From both group elements $(\mathbf{U}_0, \mathbf{U}_1)$, as well as the indices (j_0, j_1), g^{a^2} can be extracted as follows:

$$\left(\frac{\mathbf{U}_0}{\mathbf{U}_1}\right)^{\frac{1}{j_1 - j_0}} = \left(\frac{g^{-j_0 a^2 + x_i y}}{g^{-j_1 a^2 + x_i y}}\right)^{\frac{1}{j_1 - j_0}} = g^{a^2} \tag{1}$$

In the second run, B extracts g^{b^2}. In the third run B extracts $g^{(a+b)^2}$. With group elements $(g^{a^2}, g^{b^2}, g^{(a+b)^2})$, B computes g^{ab} as follows:

$$\left(\frac{g^{(a+b)^2}}{g^{a^2} g^{b^2}}\right)^{\frac{1}{2}} = g^{ab} \tag{2}$$

It is easy to see that if A has advantage ϵ in outputting at least $m+1$ messages then B has advantage ϵ^3 in winning the **gdh** game. Moreover, B's simulation is efficient. ■

4.1 UC Security

Ideal Functionality. OT_{co} aims to implement the ideal functionality $\mathsf{F}_{\mathsf{OT}}^{-}$ depicted in Figure 4.1. In the Universal Composability Framework security of a protocol is proven by showing that a protocol implements an ideal functionality. The minus in the name $\mathsf{F}_{\mathsf{OT}}^{-}$ results from a weakening of the standard ideal Functionality for OT, such that a receiver is allowed to input its choice values adaptively.

Functionality $\mathsf{F}_{\mathsf{OT}}^{-}(n, m, \ell)$

$\mathsf{F}_{\mathsf{OT}}^{-}$ interacts with a sender S and a receiver R.

- The functionality receives the messages $(M_{0,0}, ..., M_{m-1,n-1})$ from S, where for all $i, j : M_{i,j} \in \{0,1\}^\ell$.
- The functionality receives the choice values $(c_0, ..., c_{m-1}) \in \mathbb{Z}_n^m$ from R.

- The functionality returns the messages $M_{0,c_0}, ..., M_{m-1,c_{m-1}}$ to R.
- R may input the choice values in an adaptive fashion, i.e., it can input the choice values c_i one by one and learn the message M_{i,c_i} before choosing the next index.

As a warm-up, we present a security proof of the OT_{co} in the UC model for active (i.e., malicious), static corruptions under the Gap Diffie-Hellman assumption. We remark that it is also possible to show that security holds even if the receiver remains statically corrupted and the sender is corrupted in a completely adaptive fashion (also under the GDH assumption). However, since the arguments are lengthy and very similar to the ones presented in Section 5, we focus here only on the case of statically corrupted parties.

Theorem 4.3 (UC Security). OT_{co} *securely realizes* $\mathsf{F}_{\mathsf{OT}}^{-}(n, m, \ell)$ *under the following conditions.*

Corruption Model: *Any active, static corruption.*
Hybrid Functionalities: *We model* H *as a random oracle and we assume an authenticated channel (but not confidential) between the parties;*
Computational Assumptions: *We assume that the symmetric encryption scheme* (E, D) *satisfies Definition 2.2 and that the Gap Diffie-Hellman game* **gdh** *is hard to win.*

Statically Corrupted Sender and Uncorrupted Receiver. We show how to construct a simulator $\mathsf{Sim}_{\mathsf{S}^*}$ for the case of a statically corrupted sender S^* and an uncorrupted receiver. $\mathsf{Sim}_{\mathsf{S}^*}$ has access to $\mathsf{F}_{\mathsf{OT}}^{-}$, simulates an execution of protocol OT_{co} to S^* and has to simulate random oracle H.
$\mathsf{Sim}_{\mathsf{S}^*}$ proceeds as follows: For all $\mathbf{V}, \mathbf{W}, \mathbf{U} \in \mathbb{G}$, it initializes $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. To answer queries to H before learning the value $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ tests whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. If so, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{S}^*}$ returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$. Once S^* returns the group element $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ checks if $\mathbf{S} \in \mathbb{G}$ and aborts otherwise. To simulate the second step of the protocol, $\mathsf{Sim}_{\mathsf{S}^*}$ chooses random $x_i \xleftarrow{\$} Z_p$, computes $\mathbf{R}_i = g^{x_i}$, $\forall i \in [m]$ and sends the group elements $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$ to S^*. Further, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\forall i \in [m], \forall j \in [n]$: $k_{i,j} \xleftarrow{\$} \mathcal{K}$. To make previous queries to H consistent with keys $k_{i,j}$, $\mathsf{Sim}_{\mathsf{S}^*}$ checks for each previous query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ whether $\mathbf{V} = \mathbf{S}$ and there exists an $i \in [n], j \in [m]$ such that $\mathbf{W} = \mathbf{R}_i$ and $\mathsf{DH}(\mathbf{S}, \mathbf{R}_i\mathbf{S}^{-j}, \mathbf{U}) = 1$. If $\mathsf{Sim}_{\mathsf{S}^*}$ finds such i, j then $\mathsf{Sim}_{\mathsf{S}^*}$ overwrites $k_{i,j} = \mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U})$.
To simulate further queries of the form $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, $\mathsf{Sim}_{\mathsf{S}^*}$ again first checks whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$). $\mathsf{Sim}_{\mathsf{S}^*}$ checks whether both $\mathbf{V} = \mathbf{S}$ and there exists an $i \in [n]$ such that $\mathbf{W} = \mathbf{R}_i$. If this is the case, $\mathsf{Sim}_{\mathsf{S}^*}$ tests whether $\mathsf{DH}(\mathbf{S}, \mathbf{R}_i\mathbf{S}^{-j}, \mathbf{U}) = 1$ for any $j \in [n]$. If $\mathsf{Sim}_{\mathsf{S}^*}$ finds such a j then $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}) = k_{i,j}$. Otherwise, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{S}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.
Once $\mathsf{Sim}_{\mathsf{S}^*}$ receives ciphertexts $(e_{0,0}, ..., e_{n-1,m-1})$ from S^*, it computes $\forall i \in [m], \forall j \in [n]$: $M_{i,j} = \mathsf{D}(k_{i,j}, e_{i,j})$ and sends $(M_{0,0}, ..., M_{n-1,m-1})$ to $\mathsf{F}_{\mathsf{OT}}^{-}$.
To prove that no PPT distinguisher Env can tell the real world view apart from the simulated view, we argue as follows.

The distribution of the elements $\vec{R}$ sent by $\mathsf{Sim}_{\mathsf{S}^*}$ is indistinguishable from a real execution of the protocol. This follows from Lemma 4.1. Also, it is easy to see that $\mathsf{Sim}_{\mathsf{S}^*}$ perfectly simulates replies for queries to H, even if they are used as keys for encryption. Moreover, if $\mathsf{Sim}_{\mathsf{S}^*}$ aborts after learning the value $\mathbf{S}$ then $\mathsf{Sim}_{\mathsf{S}^*}$ behaves exactly like the honest receiver. It remains to argue that the messages $\vec{M}$ that $\mathsf{Sim}_{\mathsf{S}^*}$ extracts are identically distributed to the messages that an honest receiver computes in a real execution of the protocol. This follows directly from the fact that the keys derived by an honest receiver are identically distributed as in $\mathsf{Sim}_{\mathsf{S}^*}$'s simulation, since given a ciphertext $e \in \mathcal{C}$ and a key $k \in \mathcal{K}$, $M \leftarrow \mathsf{D}(e,k)$ is fully determined.

Statically Corrupted Receiver and Uncorrupted Sender. We show how to construct a simulator $\mathsf{Sim}_{\mathsf{R}^*}$ for the (statically) corrupted receiver R^* and a sender which is not corrupted. $\mathsf{Sim}_{\mathsf{R}^*}$ has access to $\mathsf{F}_{\mathsf{OT}}^-$, simulates an execution of protocol OT_{co} to R^*, has to simulate random oracle H and has to create a state for the sender which is consistent with values $(\vec{M}, \mathbf{S}, \vec{\mathbf{R}}, \vec{e})$.
$\mathsf{Sim}_{\mathsf{R}^*}$ works as follows: For all $\mathbf{V}, \mathbf{W}, \mathbf{U} \in \mathbb{G}, i \in [m]$, it initializes $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ and $c_i = \perp$.
To simulate the first step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ chooses random $y \in Z_p$, computes $\mathbf{S} = g^y$ and sends the group element $\mathbf{S}$ to R^*.
To answer queries to H, before learning the values $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$, $\mathsf{Sim}_{\mathsf{R}^*}$ tests whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. If so, $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{R}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.
Once R^* returns the group elements $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$, $\mathsf{Sim}_{\mathsf{R}^*}$ checks if for all $i \in [m]$, $\mathbf{R}_i \in \mathbb{G}$ and aborts otherwise.
To simulate the third step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ computes $\forall i \in [m], j \in [m]$: $e_{i,j} \leftarrow \mathsf{S}_1(1^\lambda)$.
To make the previously answered hash queries consistent with ciphertexts $e_{i,j}$, $\mathsf{Sim}_{\mathsf{R}^*}$ checks for each previous query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ whether $\mathbf{V} = \mathbf{S}$ and there exist $i \in [n], j \in [m]$ such that $\mathbf{W} = \mathbf{R}_i$ and $\mathbf{U} = \mathbf{R}_i^y \mathbf{S}^{-jy}$. If so, $\mathsf{Sim}_{\mathsf{R}^*}$ checks whether $c_i = \perp$. In this case, it sets $c_i = j$ and inputs c_i to $\mathsf{F}_{\mathsf{OT}}^-$, which returns $M_{i,j}$. Then $\mathsf{Sim}_{\mathsf{R}^*}$ overwrites $e_{i,j} = \mathsf{E}(\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}), M_{i,j})$. If $c_i \neq \perp$, $\mathsf{Sim}_{\mathsf{R}^*}$ aborts.
To finish simulating the third step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ sends $(e_{0,0}, ..., e_{n-1,m-1})$ to R^*.
To simulate further queries of the form $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, $\mathsf{Sim}_{\mathsf{R}^*}$ again first checks whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$). $\mathsf{Sim}_{\mathsf{R}^*}$ tests whether $\mathbf{V} = \mathbf{S}$ and there exists an $i \in [n], j \in [m]$ such that $\mathbf{W} = \mathbf{R}_i$ and $\mathbf{U} = \mathbf{R}_i^y \mathbf{S}^{-jy}$. In this case, if $c_i = \perp$, it sets $c_i = j$ and inputs c_i to $\mathsf{F}_{\mathsf{OT}}^-$, which returns $M_{i,j}$ (it aborts when $c_i \neq \perp$). $\mathsf{Sim}_{\mathsf{R}^*}$ sets $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}) = \mathsf{S}_2(e_{i,j}, M_{i,j})$. If $\mathbf{V} \neq \mathbf{S}$ or $\forall i : \mathbf{W} \neq \mathbf{R}_i$, $\mathsf{Sim}_{\mathsf{R}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{R}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.
Once for all $i \in [m]$, $c_i \neq \perp$, $\mathsf{Sim}_{\mathsf{R}^*}$ outputs all messages $M_{i,j}$.
To prove that no PPT distinguisher Env can tell the real world view apart from the simulated view, we argue as follows.
Clearly, the distribution of the group element $\mathbf{S}$ sent by $\mathsf{Sim}_{\mathsf{R}^*}$ is indistinguishable from a real execution of the protocol. The distribution of the ciphertexts $\vec{e}$ sent by $\mathsf{Sim}_{\mathsf{R}^*}$ and the replies for queries to H are indistinguishable from a real execution of the protocol, due to the *non-committing* property of the encryption scheme (E, D). Moreover, if $\mathsf{Sim}_{\mathsf{R}^*}$ aborts after learning the values $(\mathbf{R}_0, ..., \mathbf{R}_{m-1})$ then $\mathsf{Sim}_{\mathsf{R}^*}$ behaves exactly like the honest sender in the real world. If $\mathsf{Sim}_{\mathsf{R}^*}$ aborts at a later point in the simulation then there are two random oracle queries $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_0)$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R}_i)}(\mathbf{U}_1)$ such that $\mathbf{U}_0 = \mathbf{R}_i^y \mathbf{S}^{-j_0 y}$ and $\mathbf{U}_1 = \mathbf{R}_i^y \mathbf{S}^{-j_1 y}$,

Functionality $\mathsf{F}_{\mathsf{OT}}(n, \ell)$

F_{OT} interacts with a sender S and a receiver R.

- The functionality receives the messages $(M_0, ..., M_{n-1})$ from S, where for all $j : M_j \in \{0,1\}^\ell$.
- The functionality receives the choice value $c \in \mathbb{Z}_n$ from R.
- The functionality returns the message M_c to R.

Figure 4: Ideal Functionality F_{OT}.

where $j_0 \neq j_1 \in [n]$. In this case, R^* can be used to win the **gdh** game, which follows from a proof similar to the one of Lemma 4.2.

5 Our Protocol

In this section, we present our new OT protocol OT^*. In comparison to the protocol OT_{co}, running the protocol m times in parallel does not present any efficiency gain. Therefore, we define F_{OT} for only a single OT in Figure 4 accordingly.

Ideal Functionality. OT^* aims to implement the ideal functionality F_{OT} depicted in Figure 4.

The Protocol. OT^* is depicted in Figure 2. In comparison to COs protocol, we chose to change the computation of the individual encryption keys and added an additional random oracle G. This allows us to prove security under the Computational Diffie-Hellman assumption, in contrast to the stronger Gap Diffie-Hellman assumption.

	Sender		Receiver
	Input: $(M_0, ..., M_{n-1})$		Input: $c \in [n]$
	Output: none		Output:M'
1.	$y \xleftarrow{\$} \mathbb{Z}_p \quad \mathbf{S} = g^y$		$x \xleftarrow{\$} \mathbb{Z}_p$
2.		$\xrightarrow{\mathbf{S}}$	if $\mathbf{S} \notin \mathbb{G}$: abort;
3.	$\mathbf{T} = \mathsf{G}(\mathbf{S})$		$\mathbf{T} = \mathsf{G}(\mathbf{S})$
4.	if $\mathbf{R} \notin \mathbb{G}$: abort	$\xleftarrow{\mathbf{R}}$	$\mathbf{R} = \mathbf{T}^c g^x$
5.	$\forall j \in [n]$:		
6.	$k_j = \mathsf{H}_{(\mathbf{S},\mathbf{R})}\left(\mathbf{R}^y \mathbf{T}^{-jy}\right)$		$k_R = \mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{S}^x)$
7.	$= \mathsf{H}_{(\mathbf{S},\mathbf{R})}\left(g^{(c-j)ty} g^{xy}\right)$		$= \mathsf{H}_{(\mathbf{S},\mathbf{R})}(g^{xy})$
8.	$e_j \leftarrow \mathsf{E}(k_j, M_j)$		
9.		$\xrightarrow{(e_0,\ldots,e_{n-1})}$	
10.			$M' = \mathsf{D}(k_R, e_c)$

Table 2: Protocol OT^*. The value t is not known to any party. The depicted representation is for clarification only.

Efficiency. The computational cost of protocol OT^* for is composed as follows. The sender performs 1 and 2 modular exponentiations in steps 1 and 6, respectively, $2n-2$ multiplications in step 6 and 1 inversion in step 6. The receiver performs 1 and 1 modular exponentiation in steps 4 and 6, respectively and $n-2$ multiplications in step 4. In total both the sender and the receiver perform 5 modular exponentiations, $3n-4$ multiplications and one inversions. The communication cost is composed of 2 group elements $(\mathbf{S}, \mathbf{R})$ and n ciphertexts $\vec{e}$. Further, both the sender and the receiver query G each a single time in step 3. In step 6 the sender queries H n times and the receiver queries H one time.

5.1 Proof

Lemma 5.1 (Sender can not extract receivers choice values). *No (computationally unbounded)* S^* *on input* R *can guess* c *with probability greater than* $1/n$.

Proof. Same proof as in COs paper [CO15]. ∎

Lemma 5.2 (Receiver can not extract more than m messages). *In the random oracle model, no PPT algorithm* A*, engaging in the role of the receiver in an execution of* OT^* *and making at most* q_H *queries to* H *can output* 2 *or more messages if the Computational Diffie-Hellman game* **cdh** *is hard to win. Moreover, if* A *has advantage* ϵ *in outputting at least* 2 *messages then there exists an PPT algorithm* B *with advantage* $\frac{1}{q_H^2 n^2}\epsilon$ *in winning the* **gdh** *game.*

Proof. Let A be an adversary which engages with a honest sender in the OT^* protocol and outputs at least 2 correct messages at the end of the protocol. We show how to construct an adversary B, which uses A as a subroutine, to break the **cdh** game. B receives as input a CDH challenge $(\mathbf{A} = g^a, \mathbf{B} = g^b) \xleftarrow{\$} \mathbb{G}$. For all $\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{Z} \in \mathbb{G}$, B initializes both $\mathsf{G}(\mathbf{Z}) = \perp$, $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. To simulate the first step of the protocol, B sends the group element $\mathbf{S} = \mathbf{A}$ to A. To answer queries of the form $\mathsf{G}(\mathbf{U})$, where $\mathbf{U} \in \mathbb{G}$, B does the following. It first

checks whether $\mathbf{U} = \mathbf{S}$. If so, it sets $\mathsf{G}(\mathbf{U}) = \mathbf{B}$. Otherwise, it sets $\mathsf{G}(\mathbf{U}) \xleftarrow{\$} \mathbb{G}$. It then returns $\mathsf{G}(\mathbf{U})$.
To answer queries to H, B first draws four random indices $o_0, o_1 \xleftarrow{\$} [q_H]; j_0, j_1 \xleftarrow{\$} [n]$, where $o_0 \neq o_1, j_0 \neq j_1$. Before B learns the value $\mathbf{R}$, B checks for each incoming query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. If so, B sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. B then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$. Once A returns the group element $\mathbf{R}$, B checks if $\mathbf{R} \in \mathbb{G}$ and aborts otherwise.
To simulate the third step of the protocol, B computes $\forall j \in [m] : e_j \leftarrow \mathsf{S}_1(1^\lambda)$. To make the previously answered queries consistent with the ciphertexts e_j, B checks for each previous query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, whether both $\mathbf{V} = \mathbf{S}$ and $\mathbf{W} = \mathbf{R}$. If the query is of the above form, B checks whether the current query is the o_0th or o_1th query to H. If so, B overwrites $e_{j_0} = \mathsf{E}(\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}), M_{j_0})$ or $e_{j_1} = \mathsf{E}(\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}), M_{j_1})$, respectively.
To finish simulating the third step of the protocol, B sends $(e_0, ..., e_{n-1})$ to A.
To simulate future queries $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ correctly, B again first checks whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$). Then B checks whether both $\mathbf{V} = \mathbf{S}$ and $\mathbf{W} = \mathbf{R}$. If the query is of the above form, B checks whether the current query is the o_0th or o_1th query to H. If so, B returns $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}) = \mathsf{S}_2(e_{j_0}, M_{j_0})$ or $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}) = \mathsf{S}_2(e_{j_1}, M_{j_1})$, respectively. B aborts after the $\mathsf{max}(o_0, o_1)$th query to H.
By assumption, A outputs 2 messages. Therefore, A has to perform 2 distinct queries $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(g^{-j_0' ab} g^{ax})$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(g^{-j_1' ab} g^{ax})$, where $j_0', j_1' \in [n]$. If $j_0' = j_0$ and $j_1' = j_1$ then from both group elements $(\mathbf{U}_0, \mathbf{U}_1)$, as well as the indices j_0 and j_1, g^{ab} can be extracted as follows:

$$\left(\frac{\mathbf{U}_0}{\mathbf{U}_1}\right)^{\frac{1}{j_1 - j_0}} = \left(\frac{g^{-j_0' ab + ax}}{g^{-j_1' ab + ax}}\right)^{\frac{1}{j_1 - j_0}} = g^{ab} \tag{3}$$

If A has advantage ϵ in outputting at least 2 messages and given that A asks $\mathbf{U}_0, \mathbf{U}_1$ of the above form, the probability of B in winning the **cdh** game is made up of correctly guessing the hash queries $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}_0)$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}_1)$, i.e., correctly guessing the indices $o_0, o_1 \in [q_H]$ as well as correctly guessing the corresponding indices $j_0, j_1 \in [n]$. Combining terms, we obtain that

$$\mathbf{Adv}^{\mathbf{cdh}}_{\mathsf{B}} = \frac{1}{\binom{q_H}{2}\binom{n}{2}}\epsilon \geq \frac{1}{q_H^2 n^2}\epsilon.$$

Moreover, B's simulation is efficient. ∎

Theorem 5.3 (UC Security). OT^* *securely realizes* $\mathsf{F}_{\mathsf{OT}}(n, \ell)$ *under the following conditions.*

Corruption Model: *any active, adaptive corruption;*
Hybrid Functionalities: *we model* H *and* G *as random oracles and we assume an authenticated channel (but not confidential) between the parties;*
Computational Assumptions: *We assume that* (E, D) *satisfies Definition 2.2 and that the Computational Diffie-Hellman game* **cdh** *is hard to win.*

Depending on the corruption type, the tasks of a simulator differ. In the case of statically corrupted parties, the simulator must simulate an execution of the protocol to the corrupted parties, simulate both random oracles H, G, extract the input of the corrupted parties and forward this input to F_{OT}. In the case

of adaptive corruptions, the simulator learns upon corruption the input of the corrupted party. The simulator must come up with a transcript of the protocol and an internal state of this party such that the transcript and the state match the input according to the protocol specification. Adaptive corruptions can occur at any time. The later the corruption, the heavier the cost of the simulator to forge a valid state. Therefore, the hardest case to simulate are post-execution corruptions. For the honest sender, the internal state consists (of some subset of, depending on the point of corruption) of the randomness y used to generate $\mathbf{S}$ (i.e., $\mathbf{S} = g^y$), the vector of messages $\vec{M}$, and the vector of keys $\vec{k}$. For the honest receiver, the internal state consists of some subset of the internal randomness x used to generate $\mathbf{R}$, the choice value c, the key k_R, and the output message M'. We have to create a distinct simulator for each possible combination of corruptions. To this end, we incrementally describe simulators $\mathsf{Sim}_{\mathsf{S}^*}, \mathsf{Sim}_{\triangle}$, and $\mathsf{Sim}_{\triangledown}$ which build on each other and cover all the cases in which the sender is corrupted prior to or simultaneously as the receiver. Analogous (but simpler) arguments can be used to argue for the remaining cases in which the receiver is corrupted prior to the sender. We begin by describing $\mathsf{Sim}_{\mathsf{S}^*}$ for the case where the sender is statically corrupted and the receiver is not corrupted. Building on $\mathsf{Sim}_{\mathsf{S}^*}$, we describe simulator $\mathsf{Sim}_{\triangle}$ that handles the case where the sender is statically corrupted and the receiver is corrupted in a post execution fashion, i.e., after the protocol has been completed. Next, we transform $\mathsf{Sim}_{\triangle}$ into a simulator $\mathsf{Sim}_{\triangleright}$ that emulates $\mathsf{Sim}_{\triangle}$ internally and handles cases in which the sender can be corrupted at any point in the protocol, but the receiver is only corrupted once the protocol has been completed. Finally, we construct $\mathsf{Sim}_{\triangledown}$, which again emulates $\mathsf{Sim}_{\triangleright}$ internally and handles any case of corruption in which the sender is corrupted prior to or at the same time as the receiver. We remark that in our arguments, we do not call the simulators in a black-box fashion, as we sometimes require knowledge of the internal randomness that simulators generate during their runs.

Statically Corrupted Sender and Uncorrupted Receiver. We show how to construct a simulator $\mathsf{Sim}_{\mathsf{S}^*}$ for the case of a statically corrupted sender S^* and a receiver which is not corrupted. $\mathsf{Sim}_{\mathsf{S}^*}$ has access to F_{OT}, simulates an execution of protocol OT^* to S^*, and has to simulate both random oracles H and G.

$\mathsf{Sim}_{\mathsf{S}^*}$ proceeds as follows: For all $\mathbf{V}, \mathbf{W}, \mathbf{U}, \mathbf{Z} \in \mathbb{G}$, it initializes $\mathsf{G}(\mathbf{Z}) = \perp$, $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ and $r[\mathbf{U}] = \perp$.

To answer queries of the form $\mathsf{G}(\mathbf{U})$ before learning the value $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ tests whether $\mathsf{G}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{G}(\mathbf{U})$). If so, it samples $r[\mathbf{U}] \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{G}(\mathbf{U}) = g^{r[\mathbf{U}]}$. It then returns $\mathsf{G}(\mathbf{U})$.

To answer queries of the form $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ before seeing the value $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ tests whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. If so, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{S}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.

Once S^* returns the group element $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ checks if $\mathbf{S} \in \mathbb{G}$ and aborts otherwise. It then checks for each previous query $\mathsf{G}(\mathbf{U})$ whether $\mathbf{U} = \mathbf{S}$. If so, it sets $t := r[\mathbf{U}]$.

To answer queries of the form $\mathsf{G}(\mathbf{U})$, after learning the value $\mathbf{S}$, $\mathsf{Sim}_{\mathsf{S}^*}$ does as follows. It first checks whether $\mathsf{G}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{G}(\mathbf{U})$). If so, it tests whether $\mathbf{U} = \mathbf{S}$. If so, it samples $t \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{G}(\mathbf{U}) = g^t$. Otherwise, it sets $\mathsf{G}(\mathbf{U}) \xleftarrow{\$} \mathbb{G}$. It then returns $\mathsf{G}(\mathbf{U})$.

To simulate the second step of the protocol, $\mathsf{Sim}_{\mathsf{S}^*}$ chooses random $x \xleftarrow{\$} Z_p$, computes $\mathbf{R} = g^x$, and sends the group element $\mathbf{R}$ to S^*. Further, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\forall j \in [n]$: $k_j \xleftarrow{\$} \mathcal{K}$. To make previous queries to H consistent with keys k_j, $\mathsf{Sim}_{\mathsf{S}^*}$ checks for each previous query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ whether there exists a $j \in [m]$ such that $\mathbf{V} = \mathbf{S}$, $\mathbf{W} = \mathbf{R}$, and $\mathbf{U} = \mathbf{S}^{-jt}\mathbf{S}^x$. If $\mathsf{Sim}_{\mathsf{S}^*}$ finds such j then $\mathsf{Sim}_{\mathsf{S}^*}$ overwrites $k_j = \mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U})$.
To simulate further queries of the form $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, $\mathsf{Sim}_{\mathsf{S}^*}$ again first checks whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$). $\mathsf{Sim}_{\mathsf{S}^*}$ checks whether there exists a $j \in [m]$ such that $\mathbf{V} = \mathbf{S}$, $\mathbf{W} = \mathbf{R}$ and $\mathbf{U} = \mathbf{S}^{-jt}\mathbf{S}^x$. If $\mathsf{Sim}_{\mathsf{S}^*}$ finds such j then $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}) = k_j$. Otherwise, $\mathsf{Sim}_{\mathsf{S}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{S}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.
Once, S^* returns the values $(e_0, ..., e_{n-1})$, $\mathsf{Sim}_{\mathsf{S}^*}$ extracts the message vector $\vec{M}$ by computing $\forall j \in [n] : M_j = \mathsf{D}(k_j, e_j)$ and forwards the values $(M_0, ..., M_{n-1})$ to F_{OT}.
To prove that no PPT distinguisher Env can tell the real world view apart from the simulated view, we argue as follows.
The distribution of the group element $\mathbf{R}$ sent by $\mathsf{Sim}_{\mathsf{S}^*}$ is indistinguishable from a real execution of the protocol. This follows from Lemma 5.1. Also, it is easy to see that $\mathsf{Sim}_{\mathsf{S}^*}$ perfectly simulates replies for queries to H, even if they are used as keys for encryption. Moreover, if $\mathsf{Sim}_{\mathsf{S}^*}$ aborts after learning the value $\mathbf{S}$ then $\mathsf{Sim}_{\mathsf{S}^*}$ behaves exactly like the honest receiver. The messages $\vec{M}$ that $\mathsf{Sim}_{\mathsf{S}^*}$ extracts are identically distributed to the messages that an honest receiver computes in a real execution of the protocol. This follows directly from the fact that the keys derived by an honest receiver are identically distributed as in $\mathsf{Sim}_{\mathsf{S}^*}$'s simulation, since given a ciphertext $e \in \mathcal{C}$ and a key $k \in \mathcal{K}$, $M \leftarrow \mathsf{D}(e, k)$ is fully determined. The choice value c is in both worlds statistically hidden from the sender.

Statically Corrupted Sender and Post-Execution Corrupted Receiver. We show how to construct a simulator $\mathsf{Sim}_{\triangle}$ for the case of a statically corrupted sender S^* and a post-execution adaptively corrupted receiver. Simulator $\mathsf{Sim}_{\triangle}$ has access to F_{OT} and $\mathsf{Sim}_{\mathsf{S}^*}$, simulates an execution of protocol OT^* to S^*, has to simulate random oracles H and G, as well as the transcript to both parties, and has to create a state for the receiver which is consistent with values $(c, M', \mathbf{S}, \mathbf{R}, \vec{e})$.
$\mathsf{Sim}_{\triangle}$ works as follows: To simulate protocol OT^* to S^*, $\mathsf{Sim}_{\triangle}$ emulates internally simulator $\mathsf{Sim}_{\mathsf{S}^*}$. $\mathsf{Sim}_{\triangle}$ forwards all messages sent from S^* to $\mathsf{Sim}_{\mathsf{S}^*}$ and redirects the responses accordingly.
To simulate both random oracles H and G to the parties, $\mathsf{Sim}_{\triangle}$ uses simulator $\mathsf{Sim}_{\mathsf{S}^*}$, $\mathsf{Sim}_{\triangle}$ forwards all queries sent from the parties to $\mathsf{Sim}_{\mathsf{S}^*}$ and redirects the responses accordingly.
When the receiver gets corrupted after the protocol has been completed, the simulator learns c and M'. To create a state which is consistent with $(c, M', \mathbf{S}, \mathbf{R}, \vec{e})$, $\mathsf{Sim}_{\triangle}$ forwards the internal state $(x' := x - ct, k_R, c, M')$ to Env.
It is straight forward to verify that given $\mathsf{Sim}_{\mathsf{S}^*}$, $\mathsf{Sim}_{\triangle}$'s simulation is indistinguishable from a real execution of the protocol in the same corruption scenario.

Intermediate $\mathsf{Sim}_{\triangleright}$. We show how to construct a simulator $\mathsf{Sim}_{\triangleright}$ for the case of (adaptive) corruption of the sender which takes place at any point in the protocol. However, we still assume that the receiver's corruption takes place after the protocol is completed. Since we have already argued for the case of the statically

corrupted sender, we will now assume that the sender is corrupted only after the beginning of the protocol. $\mathsf{Sim}_{\triangleright}$ has access to F_{OT} and internally emulates the steps of $\mathsf{Sim}_{\triangle}$. At a high level, $\mathsf{Sim}_{\triangleright}$ works as follows: It engages with $\mathsf{Sim}_{\triangle}$ as a *statically corrupted sender* by providing it with a transcript of an honest sender who faithfully executes OT^*[1]. It forwards all of $\mathsf{Sim}_{\triangle}$'s queries to F_{OT} and returns F_{OT}'s respective answers. Similarly, it answers all random oracle queries by querying $\mathsf{Sim}_{\triangle}$. When the sender actually becomes corrupted, $\mathsf{Sim}_{\triangleright}$ outputs the state of the sender that it has been simulating thus far. It then bridges the transition to a corrupted sender from $\mathsf{Sim}_{\triangle}$'s view by making the corrupted senders queries from this point forward fit the transcript that it has generated so far. From this point on, $\mathsf{Sim}_{\triangleright}$ acts only as a relay between the corrupted sender and receiver, $\mathsf{Sim}_{\triangle}$, and F_{OT}. Since we have already shown that $\mathsf{Sim}_{\triangle}$ provides a perfect simulation in the case of a receiver that is corrupted after the protocol execution has terminated, it directly follows that $\mathsf{Sim}_{\triangleright}$'s simulation is also perfect. We show in the following how to smoothly implement the transition described above.

To simulate an honest sender, $\mathsf{Sim}_{\triangleright}$ first samples $y \xleftarrow{\$} \mathbb{Z}_p$, computes $\mathbf{S} = g^y$ and sends the group element $\mathbf{S}$ to $\mathsf{Sim}_{\triangle}$. $\mathsf{Sim}_{\triangle}$ now returns a group element $\mathbf{R}$. To simulate the last round of the protocol, $\mathsf{Sim}_{\triangleright}$ queries the random oracle H on values $k_j = \mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{R}^y\mathbf{T}^{-jy}), \forall j \in [n]$, and computes the values $e_j \leftarrow \mathsf{S}_1(1^\lambda), \forall j \in [n]$. It sends the values $\vec{e}$ to $\mathsf{Sim}_{\triangle}$.

When the sender becomes corrupted, $\mathsf{Sim}_{\triangleright}$ learns the input $(M_0, ..., M_{n-1})$ of the sender. We make the following case distinction:

- Corruption of the sender occurs before $\vec{e}$ was sent: $\mathsf{Sim}_{\triangleright}$ computes $\vec{k}$ as described above outputs as internal state $(y, \vec{k}, \vec{M})$ to the environment.
- Corruption occurs after $\vec{e}$ was sent: In this case, $\mathsf{Sim}_{\triangleright}$ sets $\forall j \in [n] : k_j = \mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{R}^y\mathbf{T}^{-jy}) = \mathsf{S}_2(e_{i,j}, M_{i,j})$ (thereby reprogramming the random oracle on these values). It then outputs $(y, \vec{k}, \vec{M})$ to the environment.

Given our arguments for $\mathsf{Sim}_{\triangle}$'s simulation from above as well as the non-committing property of (E, D), it is straight-forward to verify that $\mathsf{Sim}_{\triangleright}$ also provides a view to Env that cannot be distinguished from a real execution of the protocol.

Final Simulator $\mathsf{Sim}_{\triangledown}$. We show how to construct a simulator $\mathsf{Sim}_{\triangledown}$ for the case where the sender is adaptively corrupted and the receiver is adaptively corrupted. The corruption of the receiver must take place after the sender is corrupted (or at the same time). Simulator $\mathsf{Sim}_{\triangledown}$ has access to F_{OT} and internally emulates the steps of $\mathsf{Sim}_{\triangleright}$. The high level idea of $\mathsf{Sim}_{\triangledown}$ is the same as for $\mathsf{Sim}_{\triangleright}$: $\mathsf{Sim}_{\triangledown}$ acts as a relay for all communication between an adaptively corrupted sender and $\mathsf{Sim}_{\triangleright}$ and in this way uses $\mathsf{Sim}_{\triangleright}$ to provide a perfect simulation of an honest receiver until the receiver becomes corrupted. When the receiver becomes corrupted, $\mathsf{Sim}_{\triangledown}$ internally completes a run of the protocol for $\mathsf{Sim}_{\triangleright}$ in which the receiver is post execution corrupted (note that in reality, at this point, both parties are corrupted). To do so, it provides $\mathsf{Sim}_{\triangleright}$ with dummy values for the sender side and simply emulates the remaining steps at this point in the protocol of the simulator $\mathsf{Sim}_{\triangleright}$, given these dummy values from the

[1]Note that executing the protocol honestly is a valid strategy for a statically corrupted sender

sender. Note that this is possible, because Sim_∇ internally emulates $\mathsf{Sim}_\triangleright$ and therefore knows its internal randomness. It completes the internal run of $\mathsf{Sim}_\triangleright$ by simulating a corruption of the receiver by the environment after the protocol has been completed. When $\mathsf{Sim}_\triangleright$ outputs its internal state (x, k_R, c, M'), Sim_∇ extracts from it the necessary values and outputs them to Env. Correctness and indistinguishability of Sim_∇'s simulation follow directly from $\mathsf{Sim}_\triangleright$'s properties. We now argue for the cases of adaptive receiver corruption which occurs prior to sender corruption. The structure of our proof for this case is analogous as for the inverse case, but we only show the argument up the case in which the receiver is statically corrupted and the sender is corrupted in a post execution fashion as the remaining cases follow in a rather straight-forward fashion.

Statically Corrupted Receiver and Uncorrupted Sender. We show how to construct a simulator $\mathsf{Sim}_{\mathsf{R}^*}$ for the case of a statically corrupted receiver R^* and a sender which is corrupted after the execution of the protocol finished. $\mathsf{Sim}_{\mathsf{R}^*}$ has access to F_{OT}, simulates an execution of protocol OT^* to R^*, has to simulate random oracles H and G and has to create a state for the sender which is consistent with values $(\vec{M}, \mathbf{S}, \mathbf{R}, \vec{e})$.
$\mathsf{Sim}_{\mathsf{R}^*}$ works as follows: For all $\mathbf{V}, \mathbf{W}, \mathbf{U}, \mathbf{Z} \in \mathbb{G}$, it initializes $\mathsf{G}(\mathbf{Z}) = \perp$, $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ and $c = \perp$.
To answer queries of the form $\mathsf{G}(\mathbf{U}), \mathbf{U} \in \mathbb{G}$, $\mathsf{Sim}_{\mathsf{R}^*}$ does as follows. It first checks whether $\mathsf{G}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{G}(\mathbf{U})$). If so, it sets $\mathsf{G}(\mathbf{U}) \xleftarrow{\$} \mathbb{G}$. It then returns $\mathsf{G}(\mathbf{U})$.
To answer queries to H, before learning the value $\mathbf{R}$, $\mathsf{Sim}_{\mathsf{R}^*}$ tests whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$. If so, $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{R}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$. Once R^* returns the group element $\mathbf{R}$, $\mathsf{Sim}_{\mathsf{R}^*}$ checks whether $\mathbf{R} \in \mathbb{G}$ and aborts otherwise.
To simulate the first step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ samples $y \xleftarrow{\$} Z_p$, computes $\mathbf{S} = g^y$ and sends the group element $\mathbf{S}$ to R^*.
To simulate the third step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ computes $\forall j \in [n] : e_j \leftarrow \mathsf{S}_1(1^\lambda)$.
To make the previously answered hash queries consistent with ciphertexts e_j, $\mathsf{Sim}_{\mathsf{R}^*}$ checks for each previous query $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$ whether there exist $j \in [n]$, such that $\mathbf{V} = \mathbf{S}, \mathbf{T} = \mathsf{G}(\mathbf{S}), \mathbf{W} = \mathbf{R}$, and $\mathbf{U} = \mathbf{R}^y\mathbf{T}^{-jy}$. If so, $\mathsf{Sim}_{\mathsf{R}^*}$ checks whether $c = \perp$. In this case, it sets $c = j$ and inputs c to F_{OT}, which returns M_j. Then $\mathsf{Sim}_{\mathsf{R}^*}$ overwrites $e_j = \mathsf{E}(\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}), M_j)$. If $c \neq \perp$, $\mathsf{Sim}_{\mathsf{R}^*}$ aborts.
To finish simulating the third step of the protocol, $\mathsf{Sim}_{\mathsf{R}^*}$ sends $(e_0, ..., e_{n-1})$ to R^*.
To simulate further queries of the form $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$, $\mathsf{Sim}_{\mathsf{R}^*}$ again first checks whether $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) = \perp$ (otherwise, it returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$). $\mathsf{Sim}_{\mathsf{R}^*}$ tests whether there exist $j \in [n]$, $\mathbf{T} = \mathsf{G}(\mathbf{S})$ such that $\mathbf{V} = \mathbf{S}$, $\mathbf{W} = \mathbf{R}$ and $\mathbf{U} = \mathbf{R}^y\mathbf{T}^{-jy}$. In this case, $\mathsf{Sim}_{\mathsf{R}^*}$ checks whether $c = \perp$(otherwise it aborts). If so, it sets $c = j$ and inputs c to F_{OT}, which returns M_c. $\mathsf{Sim}_{\mathsf{R}^*}$ sets $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}) = \mathsf{S}_2(e_{i,j}, M_{i,j})$. If both $\mathbf{W} \neq \mathbf{R}$ and $\mathbf{V} \neq \mathbf{S}$, $\mathsf{Sim}_{\mathsf{R}^*}$ sets $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U}) \xleftarrow{\$} \mathcal{K}$. $\mathsf{Sim}_{\mathsf{R}^*}$ then returns $\mathsf{H}_{(\mathbf{V},\mathbf{W})}(\mathbf{U})$.
Once $c \neq \perp$, $\mathsf{Sim}_{\mathsf{R}^*}$ outputs message M_c.
To prove that no PPT distinguisher Env can tell the real world view apart from the simulated view, we argue as follows.
Clearly, the distribution of the elements $(\mathbf{S}, \vec{e})$ sent by $\mathsf{Sim}_{\mathsf{R}^*}$ is indistinguishable from a real execution of the protocol. Also, it is easy to see that $\mathsf{Sim}_{\mathsf{R}^*}$ perfectly simulates replies for queries to H, even if they are used as keys for encryption.

Moreover, if $\mathsf{Sim}_{\mathsf{R}^*}$ aborts after learning the value $\mathbf{R}$ then $\mathsf{Sim}_{\mathsf{R}^*}$ behaves exactly like the honest sender. The value c that $\mathsf{Sim}_{\mathsf{R}^*}$ extracts is identical to the choice value of an honest receiver. If $\mathsf{Sim}_{\mathsf{R}^*}$ aborts at a later point in the simulation then there are two random oracle queries $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}_0)$ and $\mathsf{H}_{(\mathbf{S},\mathbf{R})}(\mathbf{U}_1)$ such that $\mathbf{U}_0 = \mathbf{R}^y\mathbf{T}^{-j_0 y}$ and $\mathbf{U}_1 = \mathbf{R}^y\mathbf{T}^{-j_1 y}$, where $j_0 \neq j_1 \in [n]$. In this case, R^* can be used to win the **cdh** game, which follows from a proof similar to the one of Lemma 5.2.

Statically Corrupted Receiver and Post-Ex. Corrupted Sender. We show how to construct a simulator $\mathsf{Sim}_\diamond$ for the case of a statically corrupted receiver R^* and a sender which is corrupted after the execution of the protocol finished. $\mathsf{Sim}_\diamond$ has access to F_{OT} and $\mathsf{Sim}_{\mathsf{R}^*}$, simulates an execution of protocol OT^* to R^*, has to simulate random oracles H and G and has to create a state for the sender which is consistent with values $(\vec{M}, \mathbf{S}, \mathbf{R}, \vec{e})$.
To simulate protocol OT^* and both random oracles H and G to R^*, $\mathsf{Sim}_\diamond$ uses simulator $\mathsf{Sim}_{\mathsf{S}^*}$ as follows. $\mathsf{Sim}_\diamond$ forwards all messages sent from R^* to $\mathsf{Sim}_{\mathsf{S}^*}$ and vice versa.
When the sender gets corrupted after $\mathsf{Sim}_\diamond$ sends messages $(e_0, ..., e_{n-1})$ then, to create a state which is consistent with messages $(\mathbf{S}, \mathbf{R}, \vec{M}, e_0, ..., e_{n-1})$, $\mathsf{Sim}_\diamond$ forwards the state $(y, \vec{k}, \vec{M})$ to the environment machine.
The state produced by $\mathsf{Sim}_\diamond$ is indistinguishable from the state created by the honest sender.

References

[ABB+13] Michel Abdalla, Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, and David Pointcheval. SPHF-friendly non-interactive commitments. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 214–234. Springer, Heidelberg, December 2013. (Cited on page 4.)

[ABP17] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Removing erasures with explainable hash proof systems. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 151–174. Springer, Heidelberg, March 2017. (Cited on page 4.)

[ALSZ16] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. Cryptology ePrint Archive, Report 2016/602, 2016. http://eprint.iacr.org/2016/602. (Cited on page 2.)

[BC15] Olivier Blazy and Céline Chevalier. Generic construction of UC-secure oblivious transfer. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 65–86. Springer, Heidelberg, June 2015. (Cited on page 4.)

[BC16] Olivier Blazy and Céline Chevalier. Structure-preserving smooth projective hashing. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 339–369. Springer, Heidelberg, December 2016. (Cited on page 4.)

[BCG17] Olivier Blazy, Céline Chevalier, and Paul Germouty. Almost optimal oblivious transfer from QA-NIZK. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 579–598. Springer, Heidelberg, July 2017. (Cited on page 4.)

[BDD+17] Paulo S.L.M. Barreto, Bernardo David, Rafael Dowsley, Kirill Morozov, and Anderson C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the rom. *EPRINT*, 2017. (Cited on page 5.)

[Bea96a] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996. (Cited on page 2.)

[Bea96b] Donald Beaver. Equivocable oblivious transfer. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 119–130. Springer, Heidelberg, May 1996. (Cited on page 3.)

[Bea98] Donald Beaver. Adaptively secure oblivious transfer. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 300–314. Springer, Heidelberg, October 1998. (Cited on page 2, 3.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. (Cited on page 6.)

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. http://eprint.iacr.org/2004/331. (Cited on page 5.)

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. (Cited on page 2.)

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 387–402. Springer, Heidelberg, March 2009. (Cited on page 4.)

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996. (Cited on page 3.)

[CKWZ13] Seung Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. Efficient, adaptively secure, and composable oblivious transfer with a single, global CRS. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 73–88. Springer, Heidelberg, February / March 2013. (Cited on page 4.)

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140, 2002. http://eprint.iacr.org/2002/140. (Cited on page 3.)

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015. (Cited on page 2, 3, 6, 7, 8, 9, 14.)

[Cré87] Claude Crépeau. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 239–247. Springer, Heidelberg, August 1987. (Cited on page 2.)

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Heidelberg, August 2000. (Cited on page 2.)

[DNO09] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 318–335. Springer, Heidelberg, December 2009. (Cited on page 3.)

[Gar04] Juan A. Garay. Efficient and universally composable committed oblivious transfer and applications. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 297–316. Springer, Heidelberg, February 2004. (Cited on page 3, 4.)

[GH08] Matthew Green and Susan Hohenberger. Universally composable adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/163, 2008. http://eprint.iacr.org/2008/163. (Cited on page 3.)

[GIR17] Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial. Too simple to be uc-secure: On the uc-insecurity of the "simplest protocol for oblivious transfer" of chou and orlandi. Cryptology ePrint Archive, Report 2017/370, 2017. (Cited on page 2, 7.)

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 505–523. Springer, Heidelberg, August 2009. (Cited on page 4.)

[HK07] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007. (Cited on page 2.)

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008. (Cited on page 2.)

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. (Cited on page 2.)

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. Cryptology ePrint Archive, Report 2015/546, 2015. http://eprint.iacr.org/2015/546. (Cited on page 2.)

[Lar14] Enrique Larraia. Extending oblivious transfer efficiently, or - how to get active security with constant cryptographic overhead. Cryptology ePrint Archive, Report 2014/692, 2014. http://eprint.iacr.org/2014/692. (Cited on page 2.)

[Lin03] Yehuda Lindell. General composition and universal composability in secure multiparty computation. Cryptology ePrint Archive, Report 2003/141, 2003. http://eprint.iacr.org/2003/141. (Cited on page 2.)

[Lin09] Andrew Y. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 117–132. Springer, Heidelberg, April 2009. (Cited on page 2.)

[Nie07] Jesper Buus Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. http://eprint.iacr.org/2007/215. (Cited on page 2.)

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001. (Cited on page 2, 4.)

[Orl17] Claudio Orlandi. discussion on "too simple to be uc-secure: On the uc-insecurity of the “simplest protocol for oblivious transfer” of chou and orlandi", 2017. (Cited on page 8.)

[PVW07] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. Cryptology ePrint Archive, Report 2007/348, 2007. http://eprint.iacr.org/2007/348. (Cited on page 3.)

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008. (Cited on page 4.)

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Harvard University, 1981. (Cited on page 2.)

[RKP09] Alfredo Rial, Markulf Kohlweiss, and Bart Preneel. Universally composable adaptive priced oblivious transfer. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 231–247. Springer, Heidelberg, August 2009. (Cited on page 3.)

[Wie83] Stephen Wiesner. Conjugate coding. *ACM Sigact News*, 15(1):78–88, 1983. (Cited on page 2.)

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. (Cited on page 2.)

Acknowledgments

We would like to thank Claudio Orlandi for many insightful comments about our protocol and for pointing out an unnoticed issue in the proof for adaptive security. We would also like to thank Jesper Buus Nielsen and Eike Kiltz for helpful discussions.