Universally Composable Oblivious Transfer in the Multi-Party Setting

Marc Fischlin*

Institute for Theoretical Computer Science, ETH Zurich, Switzerland

marc.fischlin@inf.ethz.ch
http://www.fischlin.de/

Abstract. We construct efficient universally composable oblivious transfer protocols in the multi-party setting for honest majorities. Unlike previous proposals our protocols are designed in the plain model (i.e., without a common reference string), are secure against malicious adversaries from scratch (i.e., without requiring an expensive compiler), and are based on weaker cryptographic assumptions than comparable two-party protocols. Hence, the active participation of auxiliary parties pays off in terms of complexity. This is particularly true for the construction of one of our building blocks, an efficient universally composable *homomorphic* commitment scheme. Efficient solutions for this problem in the two-party setting are not known, not even in the common reference string model.

1 Introduction

Oblivious transfer (OT), originally defined by Rabin [Rab81], is a two-party protocol between a sender and a receiver. In this protocol the receiver either obtains a message initially held by the sender, or gets the undefined symbol $\perp$ instead. Each event occurs with probability 1/2, yet the sender remains unaware of the success of the transfer.

Oblivious transfer is a very important cryptographic primitive, for secure multi-party computations [Kil88,GMW87] as well as a tool for more practical applications like anonymous buying over the Internet [BCR87,NPS99,AIR01]. Often, these cases rely on a variant called chosen one-out-of-two oblivious transfer [EGL85]. There, the sender holds two messages and the receiver gets to choose one (and only one), but the choice is hidden from the sender. Both versions of OT have been shown to be equally powerful [Cre87].

Previous proposals for secure OT [EGL85,BCR87,BM90,NP01] have often been investigated in an isolated setting where issues like concurrency of executions or side effects caused by other cryptographic protocols are not considered. As a noteworthy exception, Garay and MacKenzie [GM00] gave the first oblivious transfer protocol which is provably secure if run concurrently. Yet, even this protocol is not known to remain secure under more advanced attack models. For example, the adversary could have some auxiliary information about the sender's messages (e.g., if the messages are used in other subprotocols), or the protocol may be executed in parallel with other cryptographic protocols.

Ideally, one would like to have an OT protocol which can be safely used as a building block within larger protocols, independently how the execution is interleaved with other steps. Such a security guarantee is provided by Canetti's universal composition (UC) framework [Can01].

* This work was supported by the Emmy Noether Programme Fi 940/1-1 of the German Research Foundation (DFG).

In this framework one defines an idealized version of the primitive in question, capturing the desired security properties in an abstract way and ensuring that the functionality is secure in interdependent settings. For example, an idealized functionality for oblivious transfer is a trustworthy interface which waits to receive the two messages from the sender as well as the receiver's choice and then delivers the corresponding message to the receiver; no further information about one party's input is given to the other party.

Given an appropriate formalization of some functionality in the UC framework, one next shows that this functionality can be securely realized by an interactive protocol between the parties (without the trusted interface). Here, securely realizing means that, in any environment in which the protocol may be run, for this environment executions of the interactive protocol are indistinguishable from executions in the ideal model with the trustworthy functionality. The UC framework, notably the composition theorem, then guarantees that the protocol can indeed be securely deployed as a subroutine in more complex protocols and environments.

1.1 Previous Results

While some important stand-alone primitives like encryption and signatures basically preserve security in the UC framework [Can01,Can04], other functionalities for commitment and oblivious transfer cannot be implemented by *any* protocol between two parties of which one can be dishonest [Can01,CF01]. In particular, previously proposed OT protocols in a stand-alone setting (even if geared to be secure for concurrent executions like [GM00]) demonstrably fail to realize the aforementioned ideal OT functionality. In fact, Lindell [Lin04], using a weaker security notion than universal composition called concurrent self-composition, shows that oblivious transfer (and other functionalities) cannot even be accomplished in this setting.

We stress that the impossibility results of [Can01,CF01] refer to protocols between two parties in the plain model, i.e., without any auxiliary parties or setup assumptions. Indeed, Canetti [Can01] shows that any functionality can be realized in the UC framework by two or more parties if a majority of the players is honest (which for two parties implies that both parties cannot be corrupted). Although based on general cryptographic assumptions, this feasibility construction is computationally expensive. It requires to evaluate the circuit computing the functionality in a gate-wise manner, and also involves a general compiler lifting security in the presence of honest-but-curios adversaries to the case of malicious adversaries. This compiler usually relies on complex zero-knowledge proofs for general NP statements.

Another workaround for the impossibility results is to let the two parties have access to a common reference string (CRS) drawn according to some fixed distribution before the execution starts. This has been successfully applied to the case of commitments [CF01], as well to oblivious transfer [CLOS02]. The OT protocol in [CLOS02] is used as a building block to extend the aforementioned feasibility result of [Can01] to dishonest majorities. It consists of a two-level design which can be implemented by any trapdoor permutation. The first part is basically the OT protocol of Goldreich et al. [GMW87] in the plain model but which is only secure against honest-but-curious adversaries. In the second step one patches the protocol to thwart malicious adversaries, using once more compiler techniques and zero-knowledge proofs. These zero-knowledge proofs then also use the common reference string model.

Recently, Garay et al. [GMY04] utilized the CRS model, too, and proposed an extended committed oblivious transfer (ECOT) protocol which is universally composable. In such an ECOT protocol the parties run an oblivious transfer but they are also committed to their data; additionally, the sender can prove in zero-knowledge some relation about his committed

values. The core of the ECOT protocol is of course a regular oblivious transfer and therefore the protocol in [GMY04] realizes the OT functionality securely in the CRS model.

The solution in [GMY04] does not rely on compiler techniques but is secure against malicious adversaries from scratch, under the decisional Diffie-Hellman assumption and the decisional composite residuosity assumption, and the strong RSA assumption or presuming chosen-message security of DSA. Yet, if implemented with the suggested efficient primitives the protocol is only known to be secure against adaptive corruptions if parties can erase internal data. Also, the protocol is geared towards evaluation of bit gates and therefore allows to transfer bit messages only; we are not aware if it can be extended easily to handle longer messages.

Another solution to bypass the two-party impossibility result, suggested in [PS04], is to lend super-polynomial power to the adversary in the real-life execution as well as in the idealized world, and to the environment trying to distinguish the two settings. This somewhat non-standard assumption about the computational power is done in a controlled way via so-called imaginary angels and it allows to overcome the need for a CRS in the general construction in [CLOS02]. The underlying oblivious transfer protocol in [PS04] is essentially identical to the one by Canetti et al. [CLOS02] and again needs a compiler and zero-knowledge proofs to handle malicious adversaries. Only the compiler is implemented differently by virtue of the imaginary angels, and can forgo the CRS.

1.2 Our Results

To overcome the two-party results of [Can01,CF01] we work in the setting of honest majorities. As explained before, for two parties this trivially boils down to a protocol between honest users. We are therefore interested in the case of three or more parties.

Assuming an honest majority one could try to reduce the design of a multi-party OT protocol in the plain model to known two-party solutions in the CRS model. For example, in the three-party case (in which the adversary can corrupt only one party) the third party could pick the CRS and the sender and receiver then run the two-party protocol on this CRS *over a secure channel*.[1] We do not pursue this approach, though, because it would not improve over existing solutions. Instead, we try to make better use of the additional parties.

Moreover, for more than three parties the straightforward approach for letting the other parties generate the CRS would require a more advanced protocol than in the three-party case. Otherwise the adversary could corrupt the sender or the receiver in addition to some of the other players, possibly allowing the malicious sender or receiver to cheat for an adversarial chosen CRS. But easy protocols for jointly generating unbiased common reference strings are not known, especially since the CRS often contains non-trivial values like an RSA modulus with unknown factorization.

We present several protocols implementing UC oblivious transfer in the multi-party scenario, depending on the number of helper parties and, especially, on the maximum of dishonest players among them. For the case of $n = 3$ parties, among which there is an honest majority and thus at most $t \leq 1$ corrupt users, we present a basic protocol to realize universally composable OT very efficiently, requiring the parties to essentially perform only one or two encryptions/decryptions. In case of static adversaries or if we alternatively presume reliable data erasure, our protocol can be implemented with any CCA-secure public-key encryption

[1] The confidential transmission guarantees that no information is revealed to the adversary, even if the third party is corrupt and chooses the CRS in a malicious way.

(assuming authenticated channels between the parties).[2] As usual in multi-party computations, adaptive corruptions are dealt with using the more expensive but presumably inevitable non-committing encryption [CFGN96,DN00]. Advantageously, in our protocol the number of bits which have to be encrypted in a non-committing way is limited by the length of the sender's messages and the receiver's choice, minimizing the usage of this encryption method.

We can extend our basic three-party protocol to more general n's and a limited number t of corrupt players. Although preserving the underlying cryptographic assumptions, the workload of the extended protocol increases exponentially with the number of corrupt players as we run many copies of our three-party protocol (yet, it remains within reasonable bounds for small t's). Hence, our protocol can tolerate up to $t = \mathcal{O}(\log k)$ dishonest players for security parameter k, where the exact bound on t depends on the relationship of the number n of honest users and t. For example, if we simply have an honest majority $n \geq 2t + 1$, then our protocol tolerates up to $t \approx \log \log k$ bad parties; if $n \geq t^2 + 2$ then we achieve the bound $t = \mathcal{O}(\log k)$. The description of this second protocol has been delegated to Appendix A.

In our third protocol, which is based on ideas developed by Bellare and Micali [BM90], we overcome the limitation $t = \mathcal{O}(\log k)$ by moving from secure encryption in general to the decisional Diffie-Hellman assumption. Our protocol utilizes UC *homomorphic* commitments which we show how to realize efficiently with Shamir's secret sharing and secure signatures in case of static corruptions and honest majorities. These homomorphic properties of commitments enable us to transfer well-known discrete-log based proof systems easily to the UC setting, resulting in an efficient OT protocol in connection with the DDH assumption. Since many practical RSA- or discrete-log-based proof systems in the stand-alone setting rely on similar homomorphic properties, our commitment protocol may be useful for the design of other efficient UC protocols in the multi-party setting.

The limitations of our commitment scheme, static corruptions and honest majorities, of course carry over to our OT protocol if implemented with this commitment protocol. However, given a UC homomorphic commitment scheme with stronger security properties, our OT protocol could tolerate any number of corrupt parties in principle. Yet, even then, our solution would only be secure against static corruptions —unless we allow reliable erasure in which case it would withstand adaptive adversaries as well.

2 Preliminaries

We work in the universal composition framework of [Can01]. In this section we give an overview over this framework, and refer the reader to [Can01] for a comprehensive introduction. Then we recall some useful basic functionalities.

2.1 UC Framework

As explained in the introduction, executions of a protocol which securely realizes some ideal functionality should be indistinguishable from executions with that functionality. This is formalized by considering two experiments as described below.

In the first experiment, the real-life execution, a probabilistic polynomial-time adversary $\mathcal{A}$ participates in a run of the interactive protocol π with a set of parties $P_1, \ldots, P_n$. All parties are connected through point-to-point communication channels. The channels are public, i.e.,

[2] Observe that stand-alone oblivious transfer in the *two-party* setting cannot be constructed from black-box public-key encryption, even for static, honest-but-curious adversaries [GKM+00].

the adversary can read all data transmitted between parties. The adversary is also responsible for delivery of messages.

Each party is initially honest and follows the predetermined program of π. The adversary may corrupt parties, either at the outset only (*non-adaptive* or *static* adversaries) or at any point during the execution (*adaptive* adversaries). An adversary corrupting at most t parties during any possible execution is called *t-limited*. Once a party is corrupted by $\mathcal{A}$ the party hands over all internal data including its input, previous incoming and outgoing communication and the content of the random tape to the adversary. If we allow *reliable erasure* then the party may delete some of these data during the execution which then remains hidden from the adversary in case of a corruption. If a party gets corrupted by the adversary then the party follows the adversary's instructions from then on. In particular, for so-called *malicious* adversaries the party may now deviate from its program.

In the second experiment, the ideal-model execution, a probabilistic polynomial-time adversary $\mathcal{S}$ (also called simulator) participates in an execution of (dummy) parties $P_1, \ldots, P_n$ with some ideal and trustworthy functionality $\mathcal{F}$. All parties are only connected to the functionality by secure channels and the simulator cannot read the content of transmissions. Once an honest dummy party gets some input it immediately forwards this input to the functionality which, at some point, may reply with output for some parties (including the simulator $\mathcal{S}$). If a party $P_1, \ldots, P_n$ receives such a message from the functionality it copies it to its output tape. We note that the simulator is responsible for delivery of these replies. Corruptions are dealt with as in the real-life setting.

In both settings an interactive distinguisher, the probabilistic polynomial-time environment $\mathcal{Z}$, is present. This environment can interact with honest parties by determining the inputs and by reading the output of these parties. Additionally, $\mathcal{Z}$ can communicate with the adversary $\mathcal{A}$ or $\mathcal{S}$, respectively. This interaction with the adversary may range from passing orders about corruptions to having the adversary report communications between parties. For both worlds the way the environment interacts with the adversary and the parties are identical. That is, $\mathcal{Z}$ only sees the input/output behavior of honest parties and the interactions with the adversary. In particular, if adversary $\mathcal{A}$ is t-limited and (non-)adaptive then so is the simulator $\mathcal{S}$.

At the end of an execution the environment should output a bit b indicating whether it thinks it observes an execution in the real-life world with protocol π and adversary $\mathcal{A}$ ($b = 0$), or in the ideal model with functionality $\mathcal{F}$ and adversary $\mathcal{S}$ ($b = 1$). The random variables describing the output distributions are denoted by $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, respectively. Informally, $\mathcal{A}$ should not have much more power attacking the interactive protocol π than $\mathcal{S}$ has in attacking the ideal functionality. Consequently, for a protocol π securely realizing $\mathcal{F}$ the view of any environment should be essentially the same, and the random variables should have roughly the same distribution:

Definition 1. *A protocol π securely realizes a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$ the random variables $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable. If the random variables are identically distributed then π securely realizes $\mathcal{F}$ in a perfect way.*

An important setting, which captures the intuition that a universally composable protocol can be used securely as a subprotocol, is the so-called hybrid model. There, an interactive protocol π is executed in presence of some ideal functionality $\mathcal{G}$, meaning that parties $P_1, \ldots, P_n$ and the adversary also have access to ideal functionality $\mathcal{G}$. Definition 1 straightforwardly carries over to this setting saying that $\text{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ should be indistinguishable.

The importance of the hybrid model becomes clear in light of the composition theorem [Can01]. If a protocol π securely realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid setting, and some protocol ρ securely realizes $\mathcal{G}$, then the protocol π^ρ (in which each call to $\mathcal{G}$ is replaced by running ρ) securely realizes $\mathcal{F}$. This can be extended to several functionalities $\mathcal{G}_1, \mathcal{G}_2, \ldots$ and protocols $\rho_1, \rho_2, \ldots$ realizing these functionalities. Additionally, nesting of functionalities (e.g., realizing $\mathcal{G}$ through ρ in some $\mathcal{H}$-hybrid setting and further realizing $\mathcal{H}$ by a protocol σ etc.) can be done up to constant depth.

On a technical note, protocol executions in the real-life and the ideal setting are always accompanied by session IDs. These IDs are provided and maintained by the system and enable the parties to distinguish between messages from different executions. Specifically, each invocation of a copy of some protocol or some functionality, respectively, is assigned a unique ID *sid*. For sake of readability we often omit mentioning these IDs for interactive protocols and note that any transmission in an interactive protocol is implicitly tagged by such a value *sid* as well as the identities of the sender and the receiver of the message.

2.2 Useful Functionalities

We usually show our OT protocols to be secure in the hybrid setting assuming some important functionalities as building blocks.

One important functionality for message transmissions is $\mathcal{F}_{\text{auth}}$. This functionality provides integrity for transmissions in the sense that the adversary cannot tamper messages undetected, nor can the adversary inject additional messages. Yet, the adversary still gets to read the content of transmission between parties. This functionality is often assumed implicitly by presuming authenticated channels between parties.

Another important functionality adding confidentiality to authenticated transmissions is $\mathcal{F}_{\text{smt}}$. This functionality can be securely realized (in the $\mathcal{F}_{\text{auth}}$-hybrid model) by CPA- or CCA-secure public-key encryption for static adversaries, and by non-committing encryption for adaptive adversaries [Can01] (or by assuming reliable erasure for semantically secure encryption). In both cases the functionality merely reveals the length of the transmission to the adversary.

Our protocols use two other basic functionalities, $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{sig}}$, for secure public-key encryption and secure signatures. Namely, $\mathcal{F}_{\text{pke}}$ allows to generate a public key enabling everyone to create ciphertexts which only the key generating party can decrypt. With $\mathcal{F}_{\text{sig}}$ a signer party can generate a verification key allowing to publicly verify signatures only the signer can create. In [Can01,Can04] it has been shown that $\mathcal{F}_{\text{pke}}$ can be realized with (non-committing) CCA-secure encryption schemes, and $\mathcal{F}_{\text{sig}}$ can be implemented through chosen-message secure signature schemes.

3 Universally Composable OT for Three Parties

We first discuss the case of three parties. Since we presume honest majorities the adversary can corrupt at most one of these three parties. The ideal functionality for oblivious transfer which our protocol should realize securely is given in Figure 1. We note that this protocol will then also provide the building block for larger n's and limited $t = \mathcal{O}(\log k)$.

3.1 The Protocol

The oblivious transfer protocol takes place between three parties: the sender S, holding two messages m_0, m_1, the receiver R with selection bit b, and a helper H with no input. If there

Functionality $\mathcal{F}_{\mathrm{OT}}$

$\mathcal{F}_{\mathrm{OT}}$ proceeds as follows, running with two parties P_i, P_j and an adversary $\mathcal{S}$, and parameterized by a value κ:

- If receiving a message $(\texttt{ot-transfer}, sid, P_i, P_j, m_0, m_1)$ with $m_0, m_1 \in \{0,1\}^\kappa$ from some party P_i store this message and ignore all further `ot-transfer` messages.
- If receiving a message $(\texttt{ot-choose}, sid, P_i, P_j, b)$ from some party P_j check if a message $(\texttt{ot-transfer}, sid, P_i, P_j, m_0, m_1)$ has been stored. If not, ignore this message. If so, send $(\texttt{ot-received}, sid, P_i, P_j, m_b)$ to P_j and ignore all future `ot-choose` messages.

Fig. 1. $\binom{2}{1}$-Oblivious Transfer Functionality $\mathcal{F}_{\mathrm{OT}}$ (adapted from [Can01])

are more than three parties then the helper position is filled in by the first party different from S and R (where we assume some order of the parties).

From a bird's eye view, the receiver and the helper each pick a random key and transfer the pair of keys to the sender. This is done such that, on the one hand, the receiver only knows one of the keys (where the order is determined by the receiver's choice) and, on the other hand, the sender remains oblivious about the owner of each key. The sender then encrypts each message with one of the keys and returns the ciphertext pair to the receiver. The receiver can decrypt message m_b with his secret key while the other message is protected by the third party's secret key.

We describe our protocol formally in Figure 2. We assume that the sender has already published a public key of an encryption scheme. Initially, the receiver and the helper both locally pick secret random string k_0 and k_1, respectively. Then they encrypt their string with the public key of S, confidentially exchange the ciphertexts and R also determines a random order of the ciphertexts which is only revealed to H. Both parties then transmit the re-ordered ciphertexts to the sender S.

In addition, R secretly sends a bit to S indicating another re-ordering of the ciphertexts. This bit, together with the first ciphertext re-arrangement between R and H, ensures that the receiver's string k_0 is encapsulated in the right ciphertext and that R later gets the message m_b. Viewed differently, the receiver's choice b is randomly split between the helper and the sender such that neither of the two parties alone can deduce b.

The sender waits to receive a ciphertext pair from each other party (and stops if it receives pairs that do not match). Then it sorts the ciphertexts according to R's request and decrypts them to obtain the strings $(K_0, K_1) = (k_b, k_{b\oplus 1})$ where b remains hidden from S. It masks the messages by $m_0 \oplus K_0$ and $m_1 \oplus K_1$ and confidentially returns them to R. The receiver unmasks the message m_b via $k_0 = K_b$, yet $m_{b\oplus 1}$ remains scrambled by H's secret string k_1.

3.2 Security

We prove security of our scheme in the ($\mathcal{F}_{\mathrm{pke}}$,$\mathcal{F}_{\mathrm{smt}}$,$\mathcal{F}_{\mathrm{auth}}$)-hybrid model. We note that no further cryptographic assumption is required given these ideal interfaces, and executions of our protocol in this setting are even perfectly indistinguishable from ideal-model executions with functionality $\mathcal{F}_{\mathrm{OT}}$. Of course, in order to realize functionality $\mathcal{F}_{\mathrm{pke}}$ for example, cryptographic primitives are usually needed and the realization "only" guarantees computational indistinguishability.

Protocol OT_3 in the $(\mathcal{F}_{\text{pke}}, \mathcal{F}_{\text{smt}}, \mathcal{F}_{\text{auth}})$-hybrid model

– **Setup:**
  - Upon receiving $(\texttt{ot-transfer}, sid, \mathsf{S}, \mathsf{R}, m_0, m_1)$, the sender S generates a key pk by calling $\mathcal{F}_{\text{pke}}$. S sends pk to the receiver R and the helper party H, each time via $\mathcal{F}_{\text{auth}}$.
  - H echos the key received by S via $\mathcal{F}_{\text{auth}}$ to R, and R aborts if the keys do not match.

– **Key Exchange Step:**
  - Receiver R gets as input $(\texttt{ot-choose}, sid, \mathsf{S}, \mathsf{R}, b)$. It picks a string $k_0 \leftarrow \{0,1\}^\kappa$ and a bit $\alpha \leftarrow \{0,1\}$, computes a ciphertext c_0 of k_0 under public key pk via $\mathcal{F}_{\text{pke}}$, and sends (c_0, α) to the helper H over the $\mathcal{F}_{\text{smt}}$ channel.
  - Having obtained (c_0, α) helper H chooses a string $k_1 \leftarrow \{0,1\}^\kappa$ and generates an $\mathcal{F}_{\text{pke}}$-ciphertext c_1 of k_1 under pk. It returns c_1 to R by $\mathcal{F}_{\text{smt}}$.
  - R and H then locally sort (c_0, c_1) according to α to $(c_\alpha, c_{\alpha\oplus 1})$ and send this ciphertext pair $(c_\alpha, c_{\alpha\oplus 1})$ to S via $\mathcal{F}_{\text{auth}}$.
  - The receiver also computes $\beta \leftarrow \alpha \oplus b$ and transmits β to S over $\mathcal{F}_{\text{smt}}$.

– **Transfer Step:**
  - The sender waits to receive the same pair $(c_\alpha, c_{\alpha\oplus 1})$ from R and H; if it receives distinct pairs, then the sender aborts. The sender also expects to get a bit β from R.
  - The sender S re-orders $(c_\alpha, c_{\alpha\oplus 1})$ to $(c_{\alpha\oplus\beta}, c_{\alpha\oplus\beta\oplus 1})$ and decrypts the pair to strings $(K_0, K_1) = (k_{\alpha\oplus\beta}, k_{\alpha\oplus\beta\oplus 1})$ by $\mathcal{F}_{\text{pke}}$. If any of the decryptions fails or does not yield a κ-bit string, then the sender aborts.
  - The sender masks the messages by $C_0 \leftarrow K_0 \oplus m_0$ and $C_1 \leftarrow K_1 \oplus m_1$, and sends (C_0, C_1) to receiver R over $\mathcal{F}_{\text{smt}}$.
  - Receiver R, upon getting (C_0, C_1), unmasks C_b with $K_b = k_{\alpha\oplus\beta\oplus b} = k_0$ to obtain message m_b. The receiver outputs $(\texttt{ot-received}, sid, \mathsf{S}, \mathsf{R}, m_b)$.

Fig. 2. $\binom{2}{1}$-Oblivious Transfer Protocol for Three Parties

The formalization in the hybrid model also abstracts the required properties of the primitives, allowing to switch between different implementations in case of adaptive and static corruptions. That is, we prove security in the presence of adaptive adversaries, guaranteeing security of the overall protocol if the functionalities $\mathcal{F}_{\text{pke}}$, $\mathcal{F}_{\text{smt}}$, $\mathcal{F}_{\text{auth}}$ are implemented by adaptively secure protocols. This also provides security against static adversaries but in this case the functionalities can also be realized by protocols which are merely secure against static corruptions. For instance, to realize functionality $\mathcal{F}_{\text{smt}}$ for adaptive adversaries only non-committing encryption schemes are known to work; for static corruptions CPA- or CCA-secure encryption schemes suffice.

Theorem 1. *Protocol* OT_3 *securely realizes functionality* $\mathcal{F}_{OT}$ *in the* $(\mathcal{F}_{pke}, \mathcal{F}_{smt}, \mathcal{F}_{auth})$*-hybrid model in a perfect way. This holds for* $n \geq 3$ *parties and* t*-limited malicious, adaptive adversaries,* $t \leq 1$*.*

Proof. We construct an ideal-model simulator $\mathcal{S}$ as follows. $\mathcal{S}$ runs a black-box simulation of the hybrid adversary $\mathcal{A}$ which is supposed to interact with ideal functionalities $\mathcal{F}_{\text{pke}}, \mathcal{F}_{\text{smt}}, \mathcal{F}_{\text{auth}}$ and the parties running the protocol. At the same time, $\mathcal{S}$ runs an execution with functionality $\mathcal{F}_{\text{OT}}$ in the ideal model. Recall that in this ideal setting none of the other functionalities is present.

For the black-box simulation $\mathcal{S}$ runs a virtual copy of the protocol execution for $\mathcal{A}$ and forwards all commands and reports between the environment $\mathcal{Z}$ and $\mathcal{A}$. It simulates each honest party up to the point when the adversary asks to corrupt this party (if at all). In case of corruption the simulator corrupts the corresponding party in the ideal model, learns the internal state and modifies and augments these data appropriately before handing it over to $\mathcal{A}$ in the black-box simulation.

To show that our protocol realizes the functionality perfectly it thus suffices to show that the black-box simulation generates the same adversarial view as in an actual attack for any inputs. Then the environment's output has the same distribution in both cases.

In the hybrid setting the adversary and the honest parties communicate with the functionalities $\mathcal{F}_{\text{pke}}, \mathcal{F}_{\text{smt}}, \mathcal{F}_{\text{auth}}$ over which the simulator has full control in the black-box simulation. In particular, the only information available to the adversary $\mathcal{A}$ about the execution are data sent over $\mathcal{F}_{\text{auth}}$ between honest parties, the (length of) encryption requests forwarded by $\mathcal{F}_{\text{pke}}$ to $\mathcal{A}$, and internal data of corrupted parties. The latter may include information previously transmitted securely over $\mathcal{F}_{\text{smt}}$.

We next describe how the simulator emulates the adversary in the black-box simulation. For this we define the simulator's steps for each honest party and for corruptions of these parties:

Simulation of Sender. $\mathcal{S}$ simulates an honest S by simply following the prescribed program, i.e., if the dummy sender in the ideal model passes a message to the ideal functionality then $\mathcal{S}$ generates a key pair via $\mathcal{F}_{\text{pke}}$, waits to receive ciphertext pairs, decrypts them and masks the messages. The only exception lies in the final step if S is supposed to transmit (C_0, C_1) to the already *corrupted* receiver. Recall that this is the only case where $\mathcal{A}$ immediately learns these information sent over $\mathcal{F}_{\text{smt}}$ at this point. If both parties are honest then the simulator does not have to pass any information to $\mathcal{A}$ about this communication (as it is virtually invisible to $\mathcal{A}$ in the hybrid model).

To simulate the transmission of (C_0, C_1) to a corrupted receiver first note that $\mathcal{S}$ knows both α and β, from the communication between R and the simulated H, and the communication between R and S. Furthermore, it knows the strings K_0, K_1 from decrypting the ciphertexts. The simulator sets $b \leftarrow \alpha \oplus \beta$ and sends $(\texttt{ot-choose}, \mathit{sid}, \mathsf{S}, \mathsf{R}, b)$ to the functionality $\mathcal{F}_{\text{OT}}$ in the ideal model to receive message m_b. The simulator sets $C_b \leftarrow K_b \oplus m_b$ and $C_{b\oplus 1} \leftarrow \{0,1\}^\kappa$ and sends (C_0, C_1) in the name of S to R over $\mathcal{F}_{\text{smt}}$.

If the adversary requests to corrupt the sender, then $\mathcal{S}$ first corrupts S in the ideal model and learns the sender's input $(\texttt{ot-transfer}, \mathit{sid}, \mathsf{S}, \mathsf{R}, m_0, m_1)$. The simulator then reveals the input and all the internal random coins to $\mathcal{A}$; no adaption is necessary for these genuine values. In addition, if (C_0, C_1) has already been sent then R must still be honest and the simulator can simply claim that the transmission over the adaptively secure channel was $(K_0 \oplus m_0, K_1 \oplus m_1)$.

Simulation of Receiver. In order to simulate an honest receiver the simulator runs a copy of R's program, with the only difference that $\mathcal{S}$ initially substitutes R's unknown input b by $\tilde{b} = 0$. This possibly causes the simulated receiver to later send $\tilde{\beta} = \alpha$ instead of $\beta = \alpha \oplus b$ (unless R is corrupted before).

If the adversary asks to corrupt R then $\mathcal{S}$ corrupts the party in the ideal model and learns the input $(\texttt{ot-choose}, \mathit{sid}, \mathsf{S}, \mathsf{R}, b)$. If this corruption happens before the receiver is supposed to send β then handing over the internal coin tosses and b to the adversary complies with the transmitted data. Assume that the receiver has already sent the substituted $\tilde{\beta}$ to S. Since the

sender is still honest the simulator is able to claim that $\beta = \alpha \oplus b$ has been transmitted over $\mathcal{F}_{\text{smt}}$ instead of $\tilde{\beta}$.

Finally, suppose the corruption of R takes place after S has supposedly sent the masked messages, in which case the adversary has not seen these values transmitted over $\mathcal{F}_{\text{smt}}$ (yet). Due to the corruption of the receiver in the ideal model the simulator learns the functionality's message $(\texttt{ot-received}, sid, \mathsf{S}, \mathsf{R}, m_b)$. The other values including β are faked as before and this time the simulator also sets $C_b \leftarrow K_b \oplus m_b$ and $C_{b\oplus 1} \leftarrow \{0,1\}^\kappa$ and claims that R has received (C_0, C_1).

Simulation of Helper. The simulator simply runs a copy of H and reveals all internal data to $\mathcal{A}$ if H is corrupted; no external input is provided to H.

Analysis. If all parties remain honest throughout the execution or if the adversary corrupts only H then the input-dependent transmissions β and (C_0, C_1) sent over $\mathcal{F}_{\text{smt}}$ are hidden perfectly from $\mathcal{A}$. Hence, the simulation perfectly mimics $\mathcal{A}$'s view in an actual attack carried out in the hybrid model, and therefore the environment's output is identically distributed in both cases.

If the sender is corrupted then, except for $\tilde{\beta}$, all values received by S and its internal data comply with the input and have the same distribution as in an actual attack. Since α is only sent via $\mathcal{F}_{\text{smt}}$ between honest parties R and H, and as $(c_\alpha, c_{\alpha\oplus 1})$ and $(k_\alpha, k_{\alpha\oplus 1})$ are independent of α, from the adversary's viewpoint the fake $\tilde{\beta} = \alpha$ and the expected $\beta = \alpha \oplus b$ are both uniformly distributed. Therefore, the simulation in case of corruption of S is perfect, too.

Suppose finally the adversary corrupts the receiver R. First note that H's secret string k_1 is protected information-theoretically from R via $\mathcal{F}_{\text{pke}}$. Then, since H is honest and hands $(c_\alpha, c_{\alpha\oplus 1})$ over to S for verification, either $c_{\alpha\oplus\beta}$ (if $\alpha \oplus \beta = 1$) or $c_{\alpha\oplus\beta\oplus 1}$ (if $\alpha \oplus \beta = 0$) must encrypt k_1. Here, α, β are the values sent by R to H and S, respectively. Moreover, the simulator predicts $b \leftarrow \alpha \oplus \beta$, and for this choice, $K_{b\oplus 1} = k_1$. Hence, the randomly chosen $C_{b\oplus 1}$ has the same distribution as $C_{b\oplus 1} = k_1 \oplus m_{b\oplus 1}$, and this case is also simulated perfectly. □

3.3 Variations

Several variations for the protocol above apply. Unless mentioned differently, similar modifications work in the case of more than three party as well.

1-*out-of-N Oblivious Transfer.* The basic protocol can be easily extended to a chosen 1-out-of-N oblivious transfer. Assume that the sender holds N messages $m_0, m_1, \ldots, m_{N-1} \in \{0,1\}^\kappa$ and the receiver would like to retrieve number $b \in \{0, 1, \ldots, N-1\}$. Our protocol can be modified as follows to implement this OT variant.

The receiver chooses again a string k_0 and encrypts it to c_0. The receiver picks a random integer $\alpha \leftarrow \{0, 1, \ldots, N-1\}$ which it sends together with c_0 to H, and sets $\beta \leftarrow b - \alpha \bmod N$. This time the helper selects $N-1$ strings $k_1, \ldots, k_{N-1}$, encrypts them to $c_1, \ldots, c_{n-1}$ and returns these ciphertexts to R. Both parties order the ciphertexts to $(c_\alpha, c_{\alpha+1 \bmod N}, \ldots, c_{\alpha+N-1 \bmod N})$ and send them to S.

Besides the ciphertexts the sender also waits to get β from R. It arranges the ciphertexts to $(c_{\alpha+\beta \bmod N}, c_{\alpha+\beta+1 \bmod N}, \ldots, c_{\alpha+\beta+N-1 \bmod N})$, decrypts them to $(K_0, K_1, \ldots, K_{N-1})$ and masks the messages by $C_i = m_i \oplus K_i$. It returns these values $(C_0, C_1, \ldots, C_{N-1})$ over a secure channel to R who unmasks $C_{\alpha+\beta \bmod N} = C_b$ with k_0 to recover m_b.

Rabin's OT. It is easy to modify our protocol into one realizing Rabin's OT functionality where the receiver either gets the sender's message m or receives $\perp$. This is accomplished by using $m_0 = m$ and $m_1 = \perp$ in the protocol above and letting the sender choose the bit β at random instead.

Re-Using the Public Key. The sender's public key pk can be re-used for several transfers from the sender if each invocation has a unique sub session ID *ssid*; see [CR03] for more details about such sub session IDs and joint state of functionalities. In this case the parties merely execute the protocol starting at the key exchange step for each transmission. This time, however, for each set of three parties (possibly with different receiver and helper) the sub session ID *ssid* and parties' identities are prepended to the encryptions of k_0, k_1. Then a malicious receiver cannot resubmit a ciphertext of a helper from some other execution undetected.

4 DDH-Based UC Oblivious Transfer

As mentioned in the introduction the description of our extended three-party protocol to tolerate up to $t = \mathcal{O}(\log k)$ corrupt players is given in Appendix A. In this section we present a UC oblivious transfer protocol which tolerates more than logarithmically many dishonest parties. While our protocol in principle withstands any number of corruptions given appropriate building blocks, our solution merely tolerates non-adaptive adversaries. Adaptive security can be achieved if we allow reliable data erasure.

Like the scheme in [GMY04] the resulting protocol here is a derivation of the protocol by Bellare and Micali [BM90]. As in the Bellare-Micali protocol we have the sender first generate and send an element $X = g^x$ of a group $\langle g \rangle$, such that the discrete logarithm x is only know to the sender. The receiver, holding bit b, next generates a pair $W_0 = g^v X^{-b}$ and $W_1 = W_0 X$ such that it knows only one of the disrete logarithms, namely, $\log W_b = v$. The receiver returns the pair W_0, W_1 to the sender who thus remains oblivious about the bit b in an information-theoretical sense. The sender encrypts the messages m_0, m_1 with the ElGamal scheme, using W_0 and W_1, respectively, as the public keys. The receiver can then decrypt m_b from the encryption under key W_b.

In order to ensure universal composition of the basic protocol the transmissions of the value X and of the pair W_0, W_1 are each accompanied by proofs of knowledge. In the first case this is a Schnorr-type proof of knowledge of a discrete logarithm [Sch91]; in the second case this corresponds to a proof of knowledge of one out of two discrete logarithms [CDS95]. Both protocols follow the well-known commitment-challenge-response structure, and to implement them in the UC framework efficiently, we present a homomorphic UC commitment scheme that is used for the initial commitment.

Below, we start by presenting our homomorphic UC commitment protocol. Given such a functionality we explain how to efficiently prove statements about discrete logarithms in the UC framework, and then use these proofs to construct our oblivious transfer protocol. We note that helper parties are only required in the implementation of the homomorphic commitment funtionality, while the discrete-log based protocol can be realized between the sender and the receiver only (in presence of an ideal homomorphic commitment functionality).

Since the suggested implementation of the efficient UC homomorphic commitment scheme merely tolerates non-adaptive adversaries and dishonest minorities. Hence, if implemented with this commitment protocol, we require $n \geq 2t + 1$ players and the assembled protocol achieves security only against static corruptions, even if we presume reliable deletion of data.

4.1 UC Homomorphic Commitments in the Multi-Party Case

In the multi-party setting with honest majority, a universally composable commitment scheme can be constructed along the lines of [BGW88]. Namely, start with Shamir's $(t, 2t)$-threshold secret sharing scheme [Sha79] where one can reconstruct a shared secret x from $t+1$ shares, yet any t or less shares are independent of x. In this scheme the dealer chooses a random polynomial f of degree t over a sufficiently large field such that $f(0) = x$. The dealer then distributes $x_i \leftarrow f(P_i)$ to party P_i (for some unique identity $P_i \neq 0$). To reconstruct the secret one interpolates the polynomial from $t+1$ shares and then evaluates it at 0.

Shamir's scheme has an additional feature useful in our context. Namely, the reconstruction algorithm is able to detect efficiently if, given more than $t+1$ shares, any two subsets would reconstruct to distinct secrets (in which case the algorithm returns $\perp$). This can be checked by reconstructing the polynomial from $t+1$ shares and then verifying that the polynomial evaluates to the right values for the other shares. We remark that this requires that identities of the parties are unchangeably associated to the shares.

For our homomorphic commitment scheme we also need a universally composable signature scheme $\mathcal{F}_{\text{sig}}$. According to [Can04] such a signature scheme can be derived from chosen-message secure signature schemes which, in turn, exist if one-way functions exist. We presume that the public verification key of the committer has already been reliably transmitted to each party, i.e., either by broadcast, or by sending it to each auxiliary party and to the receiver and letting the helpers forward a copy to the receiver for verification.

Basic UC Commitment Protocol. The basic version of a universally composable commitment protocol for $n \geq 2t+1$ goes as follows. To commit to a value x to some party P_j, the committer P_i first computes shares $x_1, x_2, \ldots, x_{2t}$ of x and, for each share x_i, also derives a signature σ_i by $\mathcal{F}_{\text{sig}}$. The committer sends, over secure channels $\mathcal{F}_{\text{smt}}$, share x_i and signature σ_i to the i-th of the first $2t$ parties other than the sender (but possibly including the receiver P_j). Each share holder informs P_j when it has received a share with a valid signature but keeps the actual value and the signature secret. The receiver outputs a receipt about a commitment taking place if it has obtained $2t$ of such confirmations.

To open the commitment, party P_i requests the other parties to reveal their shares. All auxiliary parties then forward their previously obtained share together with the signature to P_j via $\mathcal{F}_{\text{smt}}$. If all $2t$ of these shares carry valid signatures then party P_j runs the reconstruction to derive some value x; it accepts x if and only if $x \neq \perp$.

We do not prove formally that the protocol above is a universally composable commitment scheme against *non-adaptive* adversaries. The basic properties for such a protocol [CF01], extraction and equivocability, can be easily seen. We note that the proof relies on static corruptions as we need to be able to derive the value x from shares sent by a malicious sender to honest parties in the commitment phase. Adaptive adversaries, however, may be able to send out inconsistent shares at first and adapt those values later after corrupting some parties.

Homomorphic UC Protocol. Shamir's polynomial-based scheme also allows to perform additions on shares, i.e., having shared $x_1, \ldots, x_n$ via $x_{\ell,1}, \ldots, x_{\ell,2t}$ among the same parties computing, say, $x_{\ell,1} + \cdots + x_{\ell,n}$ locally generates a share of $x_1 + \cdots + x_n$. This homomorphic property carries over to the universally composable commitment, allowing the committer to open any linear combination $\sum a_\ell x_\ell$ of committed values $x_1, \ldots, x_n$ for known $a_1, \ldots, a_n$.

When applying the homomorphic properties of the sharing scheme some care with regard to the signatures is necessary, though. For example, in the opening step the commitment scheme reveals only sums of the secrets but possibly not the individual values. We solve this

Functionality $\mathcal{F}_{\text{hcom}}$

$\mathcal{F}_{\text{hcom}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$, and parameterized by an Abelian group $(\mathcal{A}, +)$.

- Upon receiving a message $(\texttt{hcom-commit}, sid, P_i, P_j, x_1, \ldots, x_n)$ for $x_1, \ldots, x_n \in \mathcal{A}$ from some party P_i, send $(\texttt{hcom-receipt}, sid, P_i, P_j, n)$ to P_j and $\mathcal{S}$. Ignore all further `hcom-commit` messages.
- If receiving a message $(\texttt{hcom-open}, sid, P_i, P_j, a_1, \ldots, a_n)$ for $a_1, \ldots, a_n \in \mathbb{N}_0$ from party P_i, check that some message $(\texttt{hcom-commit}, sid, P_i, P_j, x_1, \ldots, x_n)$ has been received from P_i before. If not, then ignore. Else compute $y \leftarrow \sum_{i=1}^{n} a_i x_i$ in $\mathcal{A}$, and send $(\texttt{hcom-open}, sid, P_i, P_j, a_1, \ldots, a_n, y)$ to P_j and $\mathcal{S}$.

Fig. 3. Homomorphic Commitment Functionality $\mathcal{F}_{\text{hcom}}$

by having the committer in the opening phase sign the sums $\sum a_\ell x_{\ell,k}$ of P_k's shares and send this signature to P_k. Then P_k can open the sum of the shares and prove correctness to the receiver by the additional signature. Party P_k will, however, disclose the individual shares with the initial signature if the sender's signature for the sum is invalid. The full protocol HCom is given in Figure 9 in Appendix B.

Proposition 1. *Protocol HCom securely realizes functionality $\mathcal{F}_{hcom}$ in the $(\mathcal{F}_{sig}, \mathcal{F}_{smt}, \mathcal{F}_{auth})$-hybrid model in a perfect way. This holds for $n \geq 2t + 1$ parties and t-limited malicious, non-adaptive adversaries.*

We remark that, following the functionality's description, our protocol allows to commit only once to a set of values. Yet, the committer is able to decommit to several linear combinations of these values.

Proof. The proof is again conducted by black-box simulation. The situation is slightly easier here as we merely deal with static corruptions. We assume that at least one of the two parties, the sender or the receiver, is honest; else the simulation is trivial.

Suppose the sender is malicious but the receiver is honest, and therefore at most $t - 1$ of the helpers can be dishonest. Then the simulator lets each honest party follow its program. If the receiver in the simulation outputs a receipt $(\texttt{hcom-receipt}, sid, P_i, P_j, n)$, then for each $\ell = 1, \ldots, n$ the simulator looks up the first $t + 1$ shares $x_{\ell,k}$ sent to honest parties P_k (with valid signatures). It runs the reconstruction algorithm on these $t + 1$ values to derive *some* value x_ℓ (which may not comply with the shares delivered to other honest parties). The simulator submits a commitment message to the ideal functionality for the obtained values $x_1, \ldots, x_n$.

In the opening step we proceed according to the program. If the simulated receiver finally receives $2t$ messages for values $(a_1, \ldots, a_n)$ it reconstructs $2t$ shares y_k as the actual receiver would, and from this it reconstructs some value y. Only if $y \neq \perp$ we have the sender in the ideal model send $(\texttt{hcom-open}, sid, P_i, P_j, a_1, \ldots, a_n)$ to the functionality.

We claim that the simulation is perfect. To see this note that the first $t + 1$ shares sent to honest helpers pin down some values $x_1, \ldots, x_n$ (as the corruptions are static the corresponding shares cannot be changed later). Also, if the malicious sender in the simulation is able to decommit to a value $y \neq \perp$ then it must hold $y = \sum a_\ell x_\ell$ for the extracted values $x_1, \ldots, x_n$. Otherwise the reconstruction algorithm on all $2t$ shares would output an error message. On

the other hand, the simulator only orders the sender to open $\sum a_\ell x_\ell$ in the ideal model if the simulated sender successfully decommits to some $y \neq \perp$.

Next assume that the receiver is malicious but the sender remains uncorrupted. Then the simulator picks values $\tilde{x}_1, \ldots, \tilde{x}_n \leftarrow 0$ on behalf of the sender at the beginning and otherwise follows the prescribed program. In particular, the simulated sender transmits signed shares of these dummy values to the (at most) t corrupt helpers.

Later, when the simulator learns a correct opening y for values $(a_1, \ldots, a_n)$ in the ideal model, it first computes dummy shares $\tilde{y}_k \leftarrow \sum a_\ell \tilde{x}_{\ell,k}$. Then it replaces the at least t shares for honest parties with shares $\tilde{y}_k$ such that reconstruction with all $2t$ shares $\tilde{y}_k$ yields y. This can be done by taking t predetermined values (including all values sent to corrupt parties), adding y as the point at 0 and interpolating a polynomial at those points. This polynomial then gives consistent values for the remaining honest parties. In addition, the sender signs all $2t$ shares. Note that malicious helpers cannot foist incorrect shares into the execution as the corresponding values have to be signed.

Note that the simulation is perfect, too, because there are at most t dishonest helpers and since the t dummy shares do not determine the true values. Also, the other shares have been sent over the secure channel $\mathcal{F}_{\text{smt}}$ to honest users, allowing the simulator to claim different values and signatures later.

Finally, suppose that both the sender and the receiver are honest. Then the simulation in case of an honest sender and corrupt receiver works here as well (where the receiver simply follows the program honestly). □

4.2 Efficient Proofs for Discrete Logarithms

Given a universally composable homomorphic commitment scheme we show how to build efficient proof systems for discrete-logarithm statements. In the sequel we presume that q is a prime and that g is a generator of a group of order q in which the decisional Diffie-Hellman problem (given g^x, g^y, g^z decide if $z = xy \bmod q$) is intractable. Moreover, let the additive group of the homomorphic commitment scheme be $(\mathcal{A}, +) = (\mathbb{Z}_q, +)$.

Proving Knowledge of Discrete Logarithms. We start by transforming Schnorr's basic protocol [Sch91] of proving knowledge of a discrete logarithm x of some group element $X = g^x$ into a UC protocol. In the original protocol the input to the prover and the verifier are the description of the group, some generator g of order q and an element$X = g^x$; the prover also gets x. The prover first chooses $r \in \mathbb{Z}_q$, computes $R \leftarrow g^r$ and sends R to the verifier. The verifier returns a random challenge $c \leftarrow \mathbb{Z}_q$ and the prover replies with $y \leftarrow r + cx \bmod q$. The verifier accepts if and only if $RX^c = g^y$.

We modify the protocol as follows. The prover, getting $(\texttt{dlzk-prove}, sid, P_i, P_j, X, x)$ as input, picks r as before and commits to r and x (in this order) by the UC commitment scheme. Additionally, instead of sending $R = g^r$ in the first step the prover also commits to R (using an independent instance of our UC commitment, possibly not homomorphic, or by first hashing R to $\mathbb{Z}_q$ and committing to the hash value together with r, x). The verifier, getting input $(\texttt{dlzk-verify}, sid, P_i, P_j, X)$, chooses $c \leftarrow \mathbb{Z}_q$ as before and sends it over $\mathcal{F}_{\text{auth}}$. But this time the prover decommits to $y \leftarrow r + cx \bmod q$ by the homomorphic commitment scheme, i.e., sends $(\texttt{hcom-open}, sid, P_i, P_j, 1, c)$ to functionality $\mathcal{F}_{\text{hcom}}$. The prover also decommits to R and the verifier accepts iff the decommitments R, y received by $\mathcal{F}_{\text{hcom}}$ satisfy $RX^c = g^y$. In this case, the verifier outputs $(\texttt{dlzk-verified}, sid, P_i, P_j, X)$. We call these steps DLZK.

Functionality $\mathcal{F}_{\mathbf{dlzk}}$

$\mathcal{F}_{\text{dlzk}}$ proceeds as follows, running with two parties P_i, P_j and an adversary $\mathcal{S}$, and parameterized by a group $\langle g \rangle$ of order q generated by g:

- If receiving a message $(\texttt{dlzk-verify}, sid, P_i, P_j, X)$ from a party P_j store this message and forward it to $\mathcal{S}$. Ignore all subsequent `dlzk-verify` messages.
- If receiving a message $(\texttt{dlzk-prove}, sid, P_i, P_j, X, x)$ from some party P_i check that a message $(\texttt{dlzk-verify}, sid, P_i, P_j, X)$ has been recorded. If not ignore, else verify that $g^x = X$. If so, deliver $(\texttt{dlzk-verified}, sid, P_i, P_j, X)$ to $\mathcal{S}$ and P_j and halt. Else ignore the message.

Fig. 4. Proving Knowledge of Discrete Logarithms Through Functionality $\mathcal{F}_{\text{dlzk}}$

The proof idea is that the simulator can extract the discrete logarithm of a malicious prover by inspecting the communication with the underlying commitment functionality. This reveals (possibly incorrect) values r, x, R satisfying $RX^c = g^y$, but with probability $1 - 1/q$ over the choice of c it must hold $x = \log X$. On the other, the equivocability property of the commitment enables the simulator to commit to dummy values and later adapt the openings.

Proposition 2. *Protocol DLZK realizes functionality $\mathcal{F}_{dlzk}$ in the $(\mathcal{F}_{hcom}, \mathcal{F}_{auth})$-hybrid model for any $t \leq n$ and t-limited malicious, adaptive adversary.*

Note that, in the proposition, we presume that we are given a commitment functionality $\mathcal{F}_{\text{hcom}}$ secure against adaptive corruptions. While such protocols can be designed in principle [Can01] our efficient solution in the previous section merely withstand non-adaptive adversaries. Hence, if implemented with this scheme our protocol here is also bound to static corruptions.

Proof. We again construct $\mathcal{S}$ by black-box simulation techniques. We remark that simulating an honest verifier (and adapting values in case of corruption) is trivial. So is the case of the two parties being corrupt from the start of the execution. Therefore, we focus on the simulation of the prover and the case of a corrupt prover and honest verifier.

The simulator can fake the honest prover's actions in a simulation of the $(\mathcal{F}_{\text{hcom}}, \mathcal{F}_{\text{auth}})$-hybrid model as follows. Having received $(\texttt{dlzk-verify}, sid, P_i, P_j, X)$ from the functionality the simulator claims that the prover has committed before to random $\tilde{r} \leftarrow \mathbb{Z}_q$, $\tilde{x} \leftarrow 0$ and $\tilde{R} \leftarrow g^{\tilde{r}} X^{-c}$, i.e., the simulator sends a `hcom-receipt` message to the verifier in the black-box simulation. Then it waits to receive the verifier's challenge c and in the opening step the simulator reveals valid values $\tilde{y} \leftarrow 1 \cdot \tilde{r} + c \cdot \tilde{x} = \tilde{r}$ and $\tilde{R}$. If the verifier in the simulation accepts and outputs $(\texttt{dlzk-verified}, sid, P_i, P_j, X)$ then we let the simulator deliver this message in the ideal model as well.

If the prover gets corrupted later then the simulator first learns the true value $x = \log_g X$ in the ideal model and adapts the internal values $\tilde{r}, \tilde{x}$ to $r \leftarrow \tilde{r} - cx \bmod q$ and x. Note that still $\tilde{y} = r + cx \bmod q$ and $R = \tilde{R}$. Similarly, if the prover gets corrupted during the proof we can easily adapt the values accordingly.

If the prover is already controlled by the adversary at the beginning of the simulation but the verifier is still honest, then the simulator learns X from the input and (possibly incorrect) values r, x, R through the universally composable commitments. The simulator sends a random challenge $c \leftarrow \mathbb{Z}_q$ in the simulation and waits to receive valid answers y, R, i.e., which satisfy $RX^c = g^y$. If so, it sends $(\texttt{dlzk-prove}, sid, P_i, P_j, X, x)$ on behalf of the malicious prover

in the ideal model to the proof functionality. $\mathcal{S}$ also delivers $(\texttt{dlzk-verified}, sid, P_i, P_j, X)$ in the ideal model to P_j and lets the verifier output this message in the simulation.

Note that in case of a corrupt prover, for $x \neq \log_g X$, there is at most one $c \in \mathbb{Z}_q$ satisfying $RX^c = g^{r+cx}$ for the committed values. Hence, except with probability $1/q$ the simulator sends the "right" $x = \log_g X$ to the functionality. In this case, the simulation is perfectly indistinguishable from an actual attack. □

Proving Or-Statements. In our oblivious transfer protocol we also require the receiver to prove that it knows one of two discrete logarithms. This is accomplished with help of the or-protocol of [CDS95]. The common input to the prover and the verifier are W_0, W_1 and X where the prover also knows w and a bit b such that $W_0 = g^w X^{-b}$ and $W_1 = W_0 X$. The aim of the prover is to convince the verifier that he indeed knows such values without revealing b. We show how to achieve this in the $(\mathcal{F}_{\text{hcom}}, \mathcal{F}_{\text{auth}})$-hybrid setting for any $t \leq n$ and adaptive adversaries.

Functionality $\mathcal{F}_{\text{dlor}}$

$\mathcal{F}_{\text{dlor}}$ proceeds as follows, running with two parties P_i, P_j and an adversary $\mathcal{S}$, and parameterized by a group $\langle g \rangle$ of order q generated by g:

- If receiving a message $(\texttt{dlor-verify}, sid, P_i, P_j, X, W_0, W_1)$ from some party P_j check that $W_1 = W_0 X$. If not ignore, else store this message and ignore all further `dlor-verify` messages. Also, send $(\texttt{dlor-verify}, sid, P_i, P_j, X, W_0, W_1)$ to $\mathcal{S}$.
- If receiving a message $(\texttt{dlor-prove}, sid, P_i, P_j, X, W_0, W_1, w, b)$ from some party P_i check that a message $(\texttt{dlor-verify}, sid, P_i, P_j, X, W_0, W_1)$ has been stored. If so and $W_0 = g^w X^{-b}$, then deliver $(\texttt{dlor-verified}, sid, P_i, P_j, X, W_0, W_1)$ to P_j and $\mathcal{S}$ and halt. Else ignore the message.

Fig. 5. Proving Knowledge of One of Two Logarithms Through Functionality $\mathcal{F}_{\text{dlor}}$

In our protocol DLOR the verifier gets $(\texttt{dlor-verify}, sid, P_i, P_j, X, W_0, W_1)$ as input. The prover with input $(\texttt{dlor-prove}, sid, P_i, P_j, X, W_0, W_1, w, b)$ picks $r_0, r_1, c_{b\oplus 1} \leftarrow \mathbb{Z}_q$ at random and computes $R_0 \leftarrow g^{r_0}(g^{-w}X)^{bc_{b\oplus 1}}$ and $R_1 \leftarrow g^{r_1}(g^w X)^{-c_{b\oplus 1}(1-b)}$. The prover commits to $r_0, r_1, (1-b)w, bw$ (in this order) and R_0, R_1 by $\mathcal{F}_{\text{hcom}}$.

The verifier sends a challenge $c \leftarrow \mathbb{Z}_q$ via $\mathcal{F}_{\text{auth}}$ and the prover divides c into $c = c_0 + c_1 \bmod q$ for the previously selected value $c_{b\oplus 1}$. The prover replies with c_0, c_1 over $\mathcal{F}_{\text{auth}}$ and decommits to R_0, R_1 and $y_0 \leftarrow r_0 + c_0(1-b)w \bmod q$ and $y_1 \leftarrow r_1 + c_1 bw \bmod q$ via $\mathcal{F}_{\text{hcom}}$, i.e., by sending $(\texttt{hcom-open}, sid, P_i, P_j, 1, 0, c_0, 0)$ and $(\texttt{hcom-open}, sid, P_i, P_j, 0, 1, 0, c_1)$. The verifier checks that $W_1 = W_0 X$, $R_0 W_0^{c_0} = g^{y_0}$, $R_1 W_1^{c_1} = g^{y_1}$ and $c = c_0 + c_1 \bmod q$ for the decommitted values. If so, it outputs $(\texttt{dlor-verified}, sid, P_i, P_j, X, W_0, W_1)$.

Proposition 3. *Protocol DLOR realizes functionality $\mathcal{F}_{dlor}$ in the $(\mathcal{F}_{hcom}, \mathcal{F}_{auth})$-hybrid model for any $t \leq n$ and t-limited malicious, adaptive adversary.*

Proof. The black-box simulation works as follows. We remark that the simulation of an honest verifier and the adaption of the verifier's internal state in case of a corruption are again trivial.

To simulate the honest prover the simulator first waits to be informed in the ideal model about $(\texttt{dlor-verify}, sid, P_i, P_j, X, W_0, W_1)$. Then the simulator claims that a commitment of

the prover has taken place and expects to receive the challenge c from the (possibly malicious) verifier. $\mathcal{S}$ prepares the answers by picking $\tilde{r}_0, \tilde{r}_1, c_0 \leftarrow \mathbb{Z}_q$ and letting $c_1 \leftarrow c - c_0 \bmod q$, $\tilde{R}_0 \leftarrow g^{\tilde{r}_0} W_0^{-c_0}$ and $\tilde{R}_1 \leftarrow g^{\tilde{r}_1} W_1^{-c_1}$. It claims to have committed to values $\tilde{r}_0, \tilde{r}_1$ and $\tilde{w}_0, \tilde{w}_1 \leftarrow 0$, such that opening these values as linear combinations $(1, 0, c_0, 0)$ and $(0, 1, 0, c_1)$ together with $\tilde{R}_0, \tilde{R}_1$ yields a valid decommitment. Moreover, the communication has the same distribution as in the hybrid model.

If the adversary asks to corrupt the prover then the simulator first learns (w, b) and can adapt the values as $r_b \leftarrow \tilde{r}_b - c_b w \bmod q$ and $r_{b\oplus 1} \leftarrow \tilde{r}_{b\oplus 1}$. Substituting also $\tilde{w}_0, \tilde{w}_1 = 0$ by the correct values $(1-b)w, bw$ yields a consistent adversarial view, having the same distribution as in the hybrid model.

We next show that, with high probability, the simulator can extract the input (w, b) from a malicious prover in the simulation when impersonating the honest verifier. Suppose towards contradiction that the prover commits to values $r_0, r_1, w_0, w_1, R_0, R_1$ but $w_0 \neq \log_g W_0$ and $w_1 \neq \log_g W_1$. By the first verification equation we have $\log_g R_0 + c_0 \log_g W_0 = r_0 + c_0 w_0 \bmod q$ and, because $w_0 - \log_g W_0 \neq 0$, there exists a single $c_0 \in \mathbb{Z}_q$ such that the committed values satisfy this equation. This follows analogously for $w_1 \neq \log_g W_1$ and c_1. Hence, once the prover has committed to values such that $w_0 \neq \log_g W_0$ and $w_1 \neq \log_g W_1$, there exists only one $c = c_0 + c_1 \bmod q$ for which the verification equations are fulfilled. In conclusion, except with probability $1/q$ the simulator learns $w_0 = \log_g W_0$ or $w_1 = \log_g W_1$, and the simulator simply submits w_b and the corresponding bit b to the ideal functionality. □

Note that the simulation above does not guarantee that the simulator extracts the right bit b. For instance, if the malicious prover knows $x = \log_g X$ then it can switch from input $(w, 0)$ to $(w - x \bmod q, 1)$ and vice versa. However, as we will later see this also requires knowledge of x and is therefore infeasible if the verifier chooses X secretly at random. In fact, this will be shown implicitly via the DDH assumption.

4.3 Oblivious Transfer Under the DDH Assumption

We show how to realize functionality $\mathcal{F}_{\text{OT}}$ in the $(\mathcal{F}_{\text{dlor}}, \mathcal{F}_{\text{dlzk}}, \mathcal{F}_{\text{auth}})$-hybrid model. Our protocol —as is— only withstands non-adaptive adversary; extending this to adaptive adversaries is possible if we allow reliable erasure, as discussed afterwards.

The full protocol is given in Figure 6. We have the sender create X with secret logarithm $x = \log_g X$ and the receiver chooses $W_0, W_1 = W_0 X$ such that it knows $\log_g W_b$ (but not $\log_g W_{b\oplus 1}$) for the selection bit b. We also let both parties prove knowledge via our ideal discrete-log functionalities. Then, the sender encrypts both messages such that one can decrypt m_a if and only if one knows $\log_g W_a$ for $a = 0, 1$. By this, it follows that the receiver can only retrieve one of the messages. In the protocol we assume for simplicity that the κ-bit messages have already been encoded in the group generated by g.

Theorem 2. *Protocol OT_{DDH} securely realizes functionality $\mathcal{F}_{OT}$ in the $(\mathcal{F}_{dlor}, \mathcal{F}_{dlzk}, \mathcal{F}_{auth})$-hybrid model under the decisional Diffie-Hellman assumption. This holds for n parties and t-limited malicious, non-adaptive adversaries, $t \leq n$.*

Proof. Once more, we construct our ideal-model simulator $\mathcal{S}$ via black-box simulation of $\mathcal{A}$ mounting an attack on the protocol in the hybrid model. We can presume that $\mathcal{A}$ either asks to corrupt the sender or the receiver at the beginning; else one can regard this as a simulation of an honest sender with a malicious receiver who follows the program trustworthily.

Protocol $\mathsf{OT}_{\mathrm{DDH}}$ in the $(\mathcal{F}_{\mathrm{dlor}}, \mathcal{F}_{\mathrm{dlzk}}, \mathcal{F}_{\mathrm{auth}})$-hybrid model

- Upon receiving $(\texttt{ot-transfer}, sid, \mathsf{S}, \mathsf{R}, m_0, m_1)$ the sender picks $x, y \leftarrow \mathbb{Z}_q$ at random and computes $X = g^x, Y = g^y$. It sends X to R over $\mathcal{F}_{\mathrm{auth}}$ and both parties engage in a proof for X via $\mathcal{F}_{\mathrm{dlzk}}$.
- The receiver gets $(\texttt{ot-choose}, sid, \mathsf{S}, \mathsf{R}, b)$ as input and chooses $w \leftarrow \mathbb{Z}_q$ and computes $W_0 = g^w X^{-b}$ and $W_1 = W_0 X$. It sends (W_0, W_1) to S via $\mathcal{F}_{\mathrm{auth}}$.
- The sender and receiver call functionality $\mathcal{F}_{\mathrm{dlor}}$ with inputs $(\texttt{dlor-verify}, sid, \mathsf{R}, \mathsf{S}, X, W_0, W_1)$ and $(\texttt{dlor-prove}, sid, \mathsf{R}, \mathsf{S}, X, W_0, W_1, w, b)$, respectively.
- The sender computes $C_0 \leftarrow m_0 W_0^y$ and $C_1 \leftarrow m_1 W_1^y$ and transmits (Y, C_0, C_1) over $\mathcal{F}_{\mathrm{auth}}$ to R.
- The receiver computes $m_b \leftarrow C_b Y^{-w}$ and outputs $(\texttt{ot-received}, sid, \mathsf{S}, \mathsf{R}, m_b)$.

Fig. 6. Oblivious Transfer based on DDH

Corrupt Sender. In the first case the adversary $\mathcal{A}$ corrupts the sender. Then the simulator waits to receive X from the sender and learns the sender's secret value x from the subprocedure call for $\mathcal{F}_{\mathrm{dlzk}}$ in the hybrid model. On behalf of the receiver in the simulation, the simulator picks $w \leftarrow \mathbb{Z}_q$ and sends $W_0 = g^w$ and $W_1 = W_0 X$. In the hybrid model, the simulator can easily simulate the interactions with functionality $\mathcal{F}_{\mathrm{dlor}}$.

After receiving C_0, C_1 and Y from the malicious sender the simulator computes $m_0 \leftarrow C_0 Y^{-w}$ and $m_1 \leftarrow C_1 Y^{-w-x}$ and sends these two messages in the sender's name in the ideal model through $(\texttt{ot-transfer}, sid, \mathsf{S}, \mathsf{R}, m_0, m_1)$. Clearly, because the distribution of the data in the simulation is independent of the receiver's input b, these steps perfectly simulate an attack in the ideal model.

Corrupt Receiver. Suppose $\mathcal{A}$ corrupts the receiver. Then the simulator first follows the prescribed program of the sender for the simulation. The simulator also learns the receiver's bit b from the invocation of $\mathcal{F}_{\mathrm{dlor}}$ and can then submit $(\texttt{ot-choose}, sid, \mathsf{S}, \mathsf{R}, b)$ to the OT functionality to retrieve message m_b. The simulator returns Y, $C_b \leftarrow m_b Y^w$ and $C_{b \oplus 1} \leftarrow g^z$ for a random $z \leftarrow \mathbb{Z}_q$ to the receiver.

We show that the simulation of an honest sender it computationally indistinguishable from an actual attack. For this we are given a DDH tuple (g^x, g^y, g^z) of group elements where z is either random or equals $xy \bmod q$. We are supposed to distinguish between these two cases, and we use an allegedly successful environment $\mathcal{Z}$ telling apart executions in the ideal and the hybrid model. We next describe our procedure $\mathcal{D}$ to refute the DDH assumption *under the condition that* S *remains honest.*

$\mathcal{D}$ follows the simulator's strategy but use the given values (g^x, g^y, g^z) instead. More precisely, $\mathcal{D}$ reads all communications between the environment and the parties, including the messages m_0, m_1 written on the sender's input tape. $\mathcal{D}$ first recovers some (w, b) such that $W_0 = g^w X^{-b}$ and $W_1 = W_0 X$ from the receiver's input to the call to functionality $\mathcal{F}_{\mathrm{dlor}}$. Note that $\mathcal{D}$ does not need to know x for the call to $\mathcal{F}_{\mathrm{dlzk}}$ as the simulator and $\mathcal{D}$ have full control over this functionality and its inputs in the hybrid model. Then $\mathcal{D}$ sends $Y \leftarrow g^y$ for the unknown y together with $C_0 \leftarrow m_0 Y^w (g^z)^{-b}$ and $C_1 \leftarrow m_1 Y^w (g^z)^{1-b}$. $\mathcal{D}$ outputs whatever the environment outputs.

If the given values (g^x, g^y, g^z) are of the form $z = xy \bmod q$ then our reply is identically distributed to the honest sender in an actual attack in the hybrid model. If, on the other

hand, z is random then $C_{b\oplus 1}$ is a random element, independent of $m_{b\oplus 1}$. Hence, in this case the distribution of the data is the same as in a simulation. By the DDH assumption it follows that the absolute difference of probabilities of $\mathcal{Z}$ outputting 0 (or 1) in either experiment must be negligible.

Analysis. We can now put the pieces together. For a given bit $a \in \{0,1\}$ let $\text{Prob}[\,\mathcal{Z} = a \,|\, \text{real}\,]$ be the probability that $\mathcal{Z}$ outputs a when interacting with $\mathcal{A}$ in the real-life setting. Analogously, let $\text{Prob}[\,\mathcal{Z} = a \,|\, \text{ideal}\,]$ be the probability that $\mathcal{Z}$ outputs a when interacting with $\mathcal{S}$ in the ideal world. Let SHon be the event that the sender remains honest and let SCor be the complementary event of S getting corrupted. Then,

$$\begin{aligned}
&\text{Prob}[\,\mathcal{Z} = a \,|\, \text{real}\,] - \text{Prob}[\,\mathcal{Z} = a \,|\, \text{ideal}\,] \\
&\quad = \text{Prob}[\,\mathcal{Z} = a \wedge \mathsf{SCor} \,|\, \text{real}\,] + \text{Prob}[\,\mathcal{Z} = a \wedge \mathsf{SHon} \,|\, \text{real}\,] \\
&\qquad - \text{Prob}[\,\mathcal{Z} = a \wedge \mathsf{SCor} \,|\, \text{ideal}\,] - \text{Prob}[\,\mathcal{Z} = a \wedge \mathsf{SHon} \,|\, \text{ideal}\,] \\
&\quad = \text{Prob}[\,\mathsf{SCor}\,] \left(\text{Prob}[\,\mathcal{Z} = a \,|\, \mathsf{SCor}, \text{real}\,] - \text{Prob}[\,\mathcal{Z} = a \,|\, \mathsf{SCor}, \text{ideal}\,]\right) \\
&\qquad + \text{Prob}[\,\mathsf{SHon}\,] \left(\text{Prob}[\,\mathcal{Z} = a \,|\, \mathsf{SHon}, \text{real}\,] - \text{Prob}[\,\mathcal{Z} = a \,|\, \mathsf{SHon}, \text{ideal}\,]\right)
\end{aligned}$$

where the last equation follows since the decision about corrupting S for the non-adaptive adversary $\mathcal{A}$ does not depend on the model.

As discussed above, the probabilities of $\mathcal{Z} = a$ in case of SCor for both settings are identical and cancel out. For an honest sender the absolute difference of the probabilities for $\mathcal{Z} = a$ are negligible by the DDH assumption and the argument above. Hence, the overall (absolute) difference must be negligible, showing that our protocol securely realizes the functionality. □

The protocol above remains secure against adaptive adversaries if we add another step where the sender erases $x, y \in \mathbb{Z}_q$ immediately after (Y, C_0, C_1) has been computed. We call this protocol $\mathsf{OT}^{\text{erase}}_{\text{DDH}}$. The additional step guarantees that the simulated sender can deny to know the secrets to unmask $m_{b\oplus 1}$. Otherwise, the values X, Y pin down x, y and therefore $C_{b\oplus 1}$ and $m_{b\oplus 1}$.

Proposition 4. *Protocol $\mathsf{OT}^{erase}_{DDH}$ securely realizes functionality $\mathcal{F}_{OT}$ in the ($\mathcal{F}_{dlor}$, $\mathcal{F}_{dlzk}$, $\mathcal{F}_{auth}$)-hybrid model under the decisional Diffie-Hellman assumption. This holds for n parties and t-limited malicious, adaptive adversaries, $t \leq n$, assuming reliable erasure.*

Proof. The proof is an easy extension of the previous proof for Theorem 2. Adaptive corruptions are straightforwardly dealt with according to the additional step erasing x, y. Also, if the receiver gets corrupted after the simulator has sent W_0, W_1 then we can change the simulator's value $(w, 0)$ to $(w - x \bmod q, 1)$ if necessary. Here, the value x is either known from the zero-knowledge proof via $\mathcal{F}_{\text{dlzk}}$ for a malicious sender, or chosen by $\mathcal{S}$ as part of the simulation of an honest sender.

The major difference to the non-adaptive case is that, there, we used the fact that the probability of S getting corrupted is independent of the model. Here we slightly change the definition and let SCor be the event that S gets corrupted before (Y, C_0, C_1) has been sent. Accordingly, SHon is the event that S remains honest, at least till (Y, C_0, C_1) has been sent. Then $\text{Prob}[\,\mathsf{SCor} \,|\, \text{real}\,] = \text{Prob}[\,\mathsf{SCor} \,|\, \text{ideal}\,]$ because, up to the point where (Y, C_0, C_1) is transmitted, the simulation and a real attack have the same distribution. Hence, the same holds for SHon and therefore the result follows as before. □

Acknowledgments

We thank the anonymous reviewers for valuable comments.

References

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. *Priced Oblivious Transfer: How to Sell Digital Goods*. Advances in Cryptology — Eurocrypt 2001, Volume 2045 of Lecture Notes in Computer Science, pages 119–135. Springer-Verlag, 2001.

[BCR87] Gilles Brassard, Claude Crpeau, and Jean-Marc Robert. *All-or-Nothing Disclosure of Secrets*. Advances in Cryptology — Crypto'86, Volume 263 of Lecture Notes in Computer Science, pages 234–238. Springer-Verlag, 1987.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1988, pages 1–10. ACM Press, 1988.

[BM90] Mihir Bellare and Silvio Micali. *Non-Interactive Oblivious Transfer and Applications*. Advances in Cryptology — Crypto'89, Volume 435 of Lecture Notes in Computer Science, pages 547–557. Springer-Verlag, 1990.

[Can01] Ran Canetti. *Universally Composable Security: A new Paradigm for Cryptographic Protocols*. Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) 2001. IEEE Computer Society Press, 2001.

[Can04] Ran Canetti. *On Universally Composable Notions of Security for Signature, Certification and Authentication*. Proceedings of Computer Security Foundations Workshop (CSFW) 2004. IEEE Computer Society Press, 2004.

[CDS95] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. *Proofs of Partial Knowledge and Simplified Desing of Witness Hiding Protocols*. Advances in Cryptology — Crypto'94, Volume 839 of Lecture Notes in Computer Science, pages 174–187. Springer-Verlag, 1995.

[CF01] Ran Canetti and Marc Fischlin. *Universally Composable Commitments*. Advances in Cryptology — Crypto 2001, Volume 2139 of Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. *Adaptively Secure Multi-Party Computation*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1996, pages 639–648. ACM Press, 1996.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. *Universally Composable Two-Party and Multi-Party Secure Computation*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2002, pages 494–503. ACM Press, 2002.

[CR03] Ran Canetti and Tal Rabin. *Universal Composition with Joint State*. Advances in Cryptology — Crypto 2003, Volume 2729 of Lecture Notes in Computer Science, pages 265–281. Springer-Verlag, 2003.

[Cre87] Claude Crepeau. *Equivalence Between Two Flavors of Oblivious Transfer*. Advances in Cryptology — Crypto'87, Lecture Notes in Computer Science, pages 350–354. Springer-Verlag, 1987.

[DN00] Ivan Damgård and Jesper Nielsen. *Improved Non-Committing Encryption Schemes Based on a General Complexity Assumption*. Advances in Cryptology — Crypto 2000, Volume 1880 of Lecture Notes in Computer Science, pages 432–450. Springer-Verlag, 2000.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. *A Randomized Protocol for Signing Contracts*. *Communications of the ACM*, 28(6):637–647, 1985.

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. *The Relationship between Public Key Encryption and Oblivious Transfer*. Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) 2000. IEEE Computer Society Press, 2000.

[GM00] Juan Garay and Philip MacKenzie. *Concurrent Oblivious Transfer*. Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) 2000, pages 314–324. IEEE Computer Society Press, 2000.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. *How to Play any Mental Game*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1987, pages 218–229. ACM Press, 1987.

[GMY04] Juan Garay, Philip MacKenzie, and Ke Yang. *Efficient and Universally Composable Committed Oblivious Transfer and Applications*. Theory of Cryptography Conference (TCC) 2004, Volume 2951 of Lecture Notes in Computer Science, pages 297–316. Springer-Verlag, 2004.

[Kil88] Joe Kilian. *Founding Crytpography on Oblivious Transfer*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 1988, pages 20–31. ACM Press, 1988.

[Lin04] Yehuda Lindell. *Lower Bounds for Concurrent Self Composition*. Theory of Cryptography Conference (TCC) 2004, Volume 2951 of Lecture Notes in Computer Science, pages 203–222. Springer-Verlag, 2004.

[NP01] Moni Naor and Benny Pinkas. *Efficient Oblivious Transfer Protocols*. Proceedings of the Annual Symposium on Discrete Algorithms (SODA) 2001, pages 448–457. ACM Press, 2001.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. *Privacy Preserving Auctions and Mechanism Design*. Proceedings of the 1st Conference on Electronic Commerce, pages 129–139. ACM Press, 1999.

[PS04] Manoj Prabhakaran and Amit Sahai. *New Notions of Security: Achieving Universal Composability without Trusted Setup*. Proceedings of the Annual Symposium on the Theory of Computing (STOC) 2004, pages 242–251. ACM Press, 2004.

[Rab81] Michael Rabin. *How to Exchange Secrets by Oblivious Transfer*. Technical Report TR-81, Aiken Computation Laboratory, 1981.

[Sch91] C.P. Schnorr. *Efficient Signature Generation by Smart Cards*. *Journal of Cryptology*, 4:161–174, 1991.

[Sha79] Adi Shamir. *How to Share a Secret*. *Communications of the ACM*, 22:612–613, 1979.

A OT for a Restricted Number of Corrupted Parties

The basic protocol for three parties in Section 3 does not directly transfer to the multi-party case. For example, if the adversary is allowed to corrupt two out of five parties, then she can always corrupt two of the three parties executing the basic protocol and learn more information than intended. Yet, we can still use the three-party protocol as a building block to derive an oblivious transfer protocol as long as the number of corrupted parties is sufficiently small. For example, our protocol works if roughly $t \approx \log\log k$ parties can be corrupt for security parameter k while we still have an honest majority $n \geq 2t+1$. Or, up to $t = \mathcal{O}(\log k)$ players may be dishonest if at the same time t is about a square root of n.

A.1 A Modified Oblivious Transfer Functionality

We slightly change the $\mathcal{F}_{\mathrm{OT}}$ functionality to $\mathcal{F}_{\mathrm{OT}_3^*}$ to capture the case of two or more corrupted parties among the three players running our basic protocol. This new functionality takes as input the name of a third party in addition to the sender's and receiver's identity. These identities allow to mimic the additional information available to the adversary if, say, the sender and the helper in an execution with an honest receiver are corrupt such that the adversary learns the receiver's choice b.

Specifically, functionality $\mathcal{F}_{\mathrm{OT}_3^*}$ accepts the same messages as before but takes two further messages from the adversary *at any time*: (`sender-panic`, *sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}$) and (`receiver-panic`, *sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}$). If the functionality receives a message `sender-panic` then it checks that the sender S is still honest, that the receiver R and the helper H are corrupt, and that it has already received a message (`ot-transfer`, *sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}$, m_0, m_1) from S. If so, it sends a message (`sender-dump`, *sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}$, m_0, m_1) to the adversary. Else it ignores the message. Analogously, the functionality responds to a `receiver-panic` message from the adversary with (`receiver-dump`,*sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}$, b) if the receiver is honest but the sender and the helper are not, and if it has already received a message (`ot-choose`, *sid*, $\mathsf{S}, \mathsf{R}, \mathsf{H}, b$) from the receiver before.[3]

Proposition 5. *Protocol* OT_3 *securely realizes functionality* $\mathcal{F}_{OT_3^*}$ *in the* ($\mathcal{F}_{pke}$, $\mathcal{F}_{smt}$, $\mathcal{F}_{auth}$)*-hybrid model in a perfect way. This holds for* $n \geq 3$ *parties and t-limited malicious, adaptive adversaries,* $t \leq n$.

Proof. The proof follows the one for the case of single corruptions closely. We only explain how to extend the simulation to the case of multiple corruptions; the first corruption is dealt with as before. If the simulated adversary demands to corrupt a second party then the ideal-model simulator first corrupts the corresponding party in the ideal setting and then proceeds as follows:

First Sender, now Receiver. If the sender has been corrupted before and now the adversary $\mathcal{A}$ asks to corrupt the receiver, then the simulator has already provided $\mathcal{A}$ with the sender's internal view (including m_0, m_1). The only critical data of the receiver is the input bit b and the related transmissions of α, β; all other values can be copied from the genuine simulation. If β has not been sent yet, then the simulator can easily fake R's view for b. Else the adversary has learned β, but since the third party H is still honest, $\mathcal{S}$ can simply claim that $\alpha \leftarrow \beta \oplus b$ has been transmitted over $\mathcal{F}_{\mathrm{smt}}$ from R to H before. This perfectly simulates a true execution.

First Receiver, now Sender. Suppose that the receiver is corrupt when the adversary and ideal-model simulator next corrupt the sender. Then $\mathcal{S}$ has already provided $\mathcal{A}$ with the receiver's internal view, possibly including already the masked messages C_0, C_1.Note that the adversary may choose $\alpha \oplus \beta$ to be different from R's original input b.

Now the simulator learns the sender's input m_0, m_1. If (C_0, C_1) has not been sent to R yet, then faking S's internal view is straightforward. Else, the only critical information about S's input is the pair (C_0, C_1) which has already been communicated to the corrupted receiver (in which case $\mathcal{A}$ sees these values, too). Although the receiver is able to recover $m_{\alpha\oplus\beta}$ from $C_{\alpha\oplus\beta}$, the string $K_{\alpha\oplus\beta\oplus 1} = k_1$, allegedly chosen by the helper before, is still information-theoretically hidden from the adversary via $\mathcal{F}_{\mathrm{pke}}$ at this point. So $\mathcal{S}$ can claim that $c_{\alpha\oplus\beta\oplus 1}$ has decrypted to $K_{\alpha\oplus\beta\oplus 1} \leftarrow C_{\alpha\oplus\beta\oplus 1} \oplus m_{\alpha\oplus\beta\oplus 1}$, implying a perfect simulation.

Helper and Another Party. If $\mathcal{A}$ asks to corrupt a second party, and H is among these two parties, then $\mathcal{S}$ first corrupts the corresponding second party in the ideal model. Then, it immediately issues a `sender-panic` or `receiver-panic` command —depending on which of the three parties is honest— in order to receive the input of this party. Given this data the simulator proceeds as in one of the two previous cases to adapt the information. Afterwards it

[3] The names ⟨`party`⟩`-panic` and ⟨`party`⟩`-dump` have been borrowed from the Unix/Linux world for unrecoverable system errors detected by the kernel (kernel panic) and a snapshot of the memory content written to disk in case of an error (core dump).

can easily modify $\mathcal{A}$'s view with these adjustments. We again stress that, although the sender or the receiver reveals its input in the `dump` reply, this party is formally not corrupted.

In any of the three cases corruption of the remaining third party is easy to simulate. □

A.2 Protocol for More Than Three Parties

The basic idea of extending our three-party protocol is to run several executions of $\mathcal{F}_{\mathrm{OT}_3^*}$. These executions are divided into T sets where each set consists of t runs of $\mathcal{F}_{\mathrm{OT}_3^*}$. Each execution of the functionality in such a set is carried out with a different auxiliary party, yet helpers may assist in more than one set.

Compiling Sets of Helpers. For the choice of the T sets of helper parties we assume that there is a selection function $\textsc{Select}(n, t)$ which takes n and t as input and returns (the description of) T sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_T$, each set comprising t helper parties different from sender and receiver. The sets need not be disjoint, yet it is guaranteed that, no matter which $t-1$ helpers the adversary will corrupt, there will always be a set $\mathcal{P}_i$ where all helpers remain honest.[4] Furthermore, 2^T must be polynomial in the security parameter k because in each execution we transfer messages of size $\kappa \cdot 2^T$. We call such an algorithm $\textsc{Select}$ *admissible*.

For example, let $t \leq \frac{1}{2}\log_2(c \log_2 k)$ for some constant c, and assume we have an honest majority $n \geq 2t+1$. Consider each of the at most $T = \binom{2t-1}{t} \leq 2^{2t} = \mathcal{O}(\log k)$ subsets of t players among the first $2t-1$ parties different from S and R, and let $\textsc{Select}$ output a description of these T sets. Clearly, since the adversary can control at most $t-1$ of the $2t-1$ parties there must be a set with honest helpers among these T sets. Note that, although T is exponential in t, for small parameters like $(n, t) = (5, 2)$ the number of invocations of our functionality $\mathcal{F}_{\mathrm{OT}_3^*}$ and the expansion factor of the messages are merely $(Tt, 2^T) = (6, 8)$.

As another example, let $t = \mathcal{O}(\log k)$ and suppose only about a square root of the $n \geq t^2+2$ players can be corrupted. Then the selection process divides the first $t^2 \leq n-2$ auxiliary parties into disjoint sets of size t and outputs descriptions of these $T = t = \mathcal{O}(\log k)$ sets. As the adversary can corrupt at most $t-1$ players in these disjoint sets, there must exist a set which contains honest helpers only. For values $(n, t) = (6, 2)$ or $(11, 3)$ this yields a reasonable overhead of $(Tt, 2^T) = (4, 4)$ and $(9, 8)$, respectively.

Protocol Description. Once the subsets have been selected, the parties start the executions with functionality $\mathcal{F}_{\mathrm{OT}_3^*}$. We first explain how the sender partitions its messages m_0, m_1; the case $T = 3$ is exemplified in Figure 7. The sender first prepares 2^{T-1} copies of each message m_0, m_1 and labels each copy by a string $r \in \{0,1\}^T$. Message m_r equals m_0 if r contains an even number of 1-bits, e.g., if $r = 011$, else m_r is set to m_1. Next, each message m_r is randomly partitioned into $m_r = m_r[1] \oplus m_r[2] \oplus \ldots \oplus m_r[T]$. For each $i = 1, 2, \ldots, T$ we combine 2^{T-1} of these parts $m_r[i]$ to strings $M_0[i]$ and $M_1[i]$, respectively, where $M_0[i]$ contains those strings $m_r[i]$ for which the i-th bit $r[i]$ in $r = r[1]r[2]\ldots r[T]$ equals 0, and $M_1[i]$ consists of the remaining 2^{T-1} parts. See again Figure 7.

The sender takes message pair $(M_0[i], M_1[i])$ for the executions with auxiliary parties in $\mathcal{P}_i$ for $i = 1, 2, \ldots, T$. In each of these T subprotocols the sender splits $M_0[i]$ into t random parts $M_0[i, j]$ subject to $M_0[i] = M_0[i, 1] \oplus M_0[i, 2] \oplus \ldots \oplus M_0[i, t]$. Analogously for $M_1[i]$. The

[4] We can assume that the adversary corrupts either the sender or the receiver in an execution, otherwise the simulation becomes trivial. This leaves at most $t-1$ additionally corruptions for the sets $\mathcal{P}_1, \ldots, \mathcal{P}_T$.

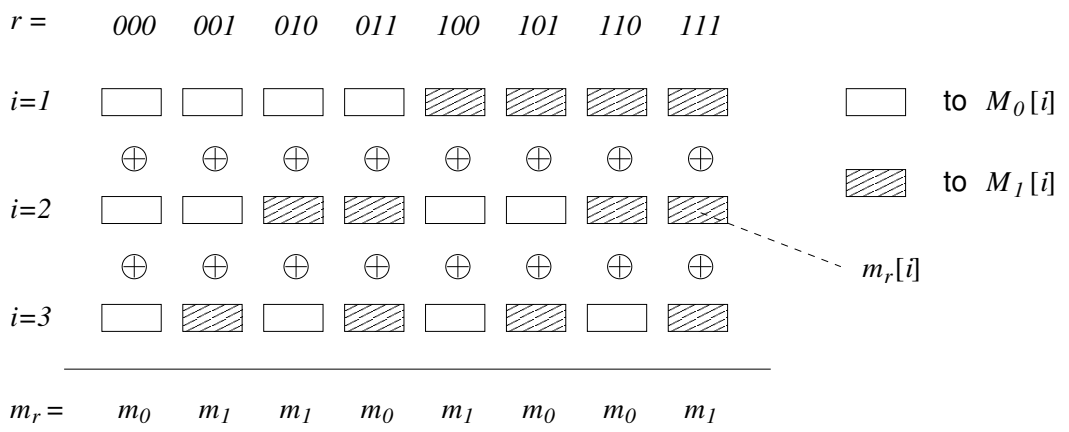

Fig. 7. Splitting messages m_0, m_1 in protocol $\mathsf{OT}_{\log}$ for $T = 3$

receiver, on the other hand, partitions its choice b only into $b = b[1] \oplus b[2] \oplus \ldots \oplus b[T]$ but does not further split the bits $b[i]$ and uses the same bit $b[i]$ in each of the executions with the helpers in $\mathcal{P}_i$.

The sender and the receiver then engage in t invocations of $\mathcal{F}_{\mathrm{OT}_3^*}$ for each set $\mathcal{P}_i$. In each execution one of the helpers $\mathsf{H}_j \in \mathcal{P}_i$ assists the two parties. In this invocation the sender uses pair $(M_0[i,j], M_1[i,j])$ while the receiver uses $b[i]$ as input. At the end, the receiver thus obtains $M_{b[i]}[i,j]$ for all $j = 1, 2, \ldots, t$, which allows him to reconstruct $M_{b[i]}[i]$. Given these T parts the receiver can extract all values $m_r[i]$, $i = 1, 2, \ldots, T$, for $r = b[1] \ldots b[T]$ and therefore $m_{b[1] \oplus \ldots \oplus b[T]} = m_b$.

The partitioning and expansion of m_0, m_1 into T strings $M_0[i]$ and $M_1[i]$ on the one hand ensures that, if one knows only one of the two strings $M_0[i], M_1[i]$ for each $i = 1, 2, \ldots, T$, then one can reconstruct either m_0 or m_1 but not both. This follows because if one knows $M_{b[i]}[i]$ for $b[1], \ldots, b[T]$ then one has all the pieces to assemble $m_b = m_{b[1] \ldots b[T]}$. Yet, for each other m_r with $r = r[1] \ldots r[T] \neq b[1] \ldots b[T]$ at least one part is missing, e.g., for the smallest j such that $r[j] \neq b[j]$ string $m_{b[1] \ldots r[j] \ldots b[T]}[j]$ is not included in $M_{b[j]}[j]$. This can be easily verified for the example given in Figure 7.

On the other hand, if a party knows one of the strings, $M_{b[i]}[i]$ for every $b[1], \ldots, b[T]$, and one is unaware of at least one of the choices $b[j]$, then one is completely oblivious about the message m_0 or m_1 this party can reconstruct. This can been seen because either choice for $b[j]$ allows to switch between m_0 and m_1 (because flipping one bit in r changes $m_r = m_0$ to $m_r = m_1$ and vice versa). Again, see Figure 7.

The first property allows the simulator to determine which of the two messages the adversary is able to reconstruct. Since there is an honest user in each of the T sets of helpers, the simulator gets to know all choices $b[1], \ldots, b[T]$ of a malicious receiver, and can conclude that the adversary learns $m_{b[1] \ldots b[T]}$ but not the other message. With the second property, and the fact that there is a set of honest helpers only, the simulator can determine both message parts $M_0[j], M_1[j]$ for some j and is therefore able to extract both messages m_0, m_1 from a malicious sender.

Protocol $\mathsf{OT}_{\log}$ in the $\mathcal{F}_{\mathrm{OT}_3^*}$-hybrid model

- Upon receiving $(\texttt{ot-transfer}, sid, \mathsf{S}, \mathsf{R}, m_0, m_1)$ and $(\texttt{ot-choose}, sid, \mathsf{S}, \mathsf{R}, b)$, respectively, the sender and the receiver locally run $\textsc{Select}(n, t)$ to get T sets $\mathcal{P}_1, \ldots, \mathcal{P}_T$ of t parties each.
- The sender defines for any string $r \in \{0,1\}^T$ a message m_r to be $m_{r[1]\oplus\ldots\oplus r[T]}$, i.e., to be the message m_0 or m_1, depending on the even or odd number of bits in r. For each r the sender picks T pieces $m_r[i]$ at random such that $m_r = m_r[1] \oplus m_r[2] \oplus \ldots \oplus m_r[T]$. The sender further defines $M_0[i]$ to be concatenation of all 2^{T-1} strings $m_r[i]$ such that the i-th bit $r[i]$ in r equals 0; define $M_1[i]$ analogously for each $i = 1, 2, \ldots, T$.
- The receiver picks random bits $b[i]$ subject to $b = b[1] \oplus b[2] \oplus \ldots \oplus b[T]$.
- For each of the T subsets $\mathcal{P}_i$ do the following:

- **Subprotocol for** $M_0[i], M_1[i], b[i], \mathcal{P}_i$**:**

  - The sender, holding $M_0[i], M_1[i]$, picks $M_0[i, j]$ for $j = 1, 2, \ldots, t$ at random such that $M_0[i] = M_0[i, 1] \oplus M_0[i, 2] \oplus \ldots \oplus M_0[i, t]$. Analogously for $M_1[i]$.
  - For each party H_j in $\mathcal{P}_i$ parties S, R and H_j execute the $\mathcal{F}_{\mathrm{OT}_3^*}$ functionality with inputs $(\texttt{ot-transfer}, sid, \mathsf{S}, \mathsf{R}, \mathsf{H}_j, M_0[i, j], M_1[i, j])$ for S and $(\texttt{ot-choose}, sid, \mathsf{S}, \mathsf{R}, \mathsf{H}_j, b[i])$ for R. By this, the receiver gets $M_{b[i]}[i, j]$ for $j = 1, 2, \ldots, t$.
  - R computes $M_{b[i]}[i] = M_{b[i]}[i, 1] \oplus M_{b[i]}[i, 2] \oplus \ldots \oplus M_{b[i]}[i, t]$.

- Having obtained $M_{b[i]}[i]$ from all T subprotocols the receiver finally extracts $m_{b[1]\ldots b[T]}[i]$ from the values $M_{b[i]}[i]$ and computes $m_b = m_{b[1]\ldots b[T]} = m_{b[1]\ldots b[T]}[1] \oplus \ldots \oplus m_{b[1]\ldots b[T]}[T]$. The receiver outputs $(\texttt{ot-received}, sid, \mathsf{S}, \mathsf{R}, m_b)$.

Fig. 8. $\binom{2}{1}$-Oblivious Transfer for More Than Three Parties

Theorem 3. *Let* $\textsc{Select}(n, t)$ *be an admissible selection algorithm. Then protocol* $\mathsf{OT}_{\log}$ *securely realizes functionality* $\mathcal{F}_{OT}$ *in the* $\mathcal{F}_{OT_3^*}$*-hybrid model in a perfect way for* n *parties and* t*-limited malicious, adaptive adversaries.*

Recall that $\textsc{Select}(n, t)$ is for example admissible if $t = \mathcal{O}(\log k)$ and $n \geq t^2 + 2$ or if $t \leq \frac{1}{2}\log_2(c \log_2 k)$ and $n \geq 2t + 1$.

Proof. As in the proof of Theorem 1 we again present an ideal-model simulator $\mathcal{S}$ via black-box construction. Yet, in the hybrid model except for the invocation of $\mathcal{F}_{\mathrm{OT}_3^*}$ no information is exchanged between the parties, and the simulator only has to provide the corresponding output to corrupted parties calling the functionality.

Simulation of Sender. To simulate an honest sender $\mathcal{S}$ assumes that the sender's input is $\tilde{m}_0 = \tilde{m}_1 = 0^\kappa$ and otherwise follows the prescribed program. In particular, $\mathcal{S}$ also divides the messages $\tilde{m}_0, \tilde{m}_1$ into random parts $\tilde{M}_0[i], \tilde{M}_1[i]$ for $i = 1, 2, \ldots, T$, and further splits these parts into strings $\tilde{M}_0[i, j], \tilde{M}_1[i, j]$ for $j = 1, 2, \ldots, t$ in each subprotocol execution.

Before we describe the simulation in detail we need some notations. We enumerate all invocations of $\mathcal{F}_{\mathrm{OT}_3^*}$ according to $i \in \{1, 2, \ldots, T\}$, the set $\mathcal{P}_i$ of helpers, as well as according to $j \in \{1, 2, \ldots, t\}$, the number of the third party H_j in $\mathcal{P}_i$ running $\mathcal{F}_{\mathrm{OT}_3^*}$ with the sender and the receiver. We call this *execution* (i, j). By *execution ensemble* i we refer to the set of executions (i, j) for $j = 1, 2, \ldots, t$.

If we are at some point in the simulation we call an execution (i, j) *open* if the receiver has not received a message $(\texttt{ot-received}, \mathit{sid}, \mathsf{S}, \mathsf{R}, *)$ yet, and if no $\langle\texttt{party}\rangle\texttt{-dump}$ message has been sent to the adversary so far. Any other execution is called *closed*. We call the execution ensemble i *open* if some execution (i, j) is still open, or if there are executions (i, j), (i, k) such that no `dump`-messages have been sent to the adversary in these executions, but the receiver has submitted distinct choices to $\mathcal{F}_{\mathrm{OT}_3^*}$. Else the ensemble is called *closed*. Note that in an open ensemble in which all individual executions are closed, the adversary has retrieved different parts $\tilde{M}_0[i, j]$ and $\tilde{M}_1[i, k]$ for $j \neq k$ and is neither able to reconstruct $\tilde{M}_0[i]$ nor $\tilde{M}_1[i]$. However, the adversary may still corrupt further parties in this ensemble and enforce `dump`-messages later to recover one of these messages.

We presume for simplicity that the adversary has corrupted the receiver at the outset. If not, since the sender is still honest, the adversary does not see any data in the execution up to the corruption step. Once the simulator has made up the receiver's internal view when the adversary asks to corrupt the receiver, we can think of these emulated choices as now being sent by a malicious receiver honestly following its program. The simulator then adapts the sender's view accordingly, as described next.

The ideal-model simulator proceeds as follows. As explained above, it uses the fake messages $\tilde{m}_0 = \tilde{m}_1 = 0^\kappa$ to run the invocations of $\mathcal{F}_{\mathrm{OT}_3^*}$ (also answering `panic`-requests according to the specification). This is done as long as there are open ensembles. Only if this is going to change then the simulator prepares to submit a message on behalf of R in the ideal model. We remark that the simulation up to this point is perfect. If there is still an open ensemble i then the adversary is still missing pieces to assemble $\tilde{M}_0[i]$ and $\tilde{M}_1[i]$ and thus $\tilde{m}_0$ and $\tilde{m}_1$, and the random parts the adversary has seen so far have the same distribution as in an actual attack.

Now assume that the last ensemble i is going to be closed because of an `ot-choose`-message or a `panic`-message in execution (i, j). For each ensemble we look at the executions in which the helper parties are still honest (and thus all `panic`-messages have been ignored so far). By choice of the T sets there must always be such an execution in each ensemble (as the adversary can only corrupt $t - 1$ helpers). Furthermore, as each ensemble k is or will be closed, the choices $b[k]$ submitted by the receiver to $\mathcal{F}_{\mathrm{OT}_3^*}$ in such executions with honest helpers in one ensemble are all consistent. Hence, for every $k = 1, 2, \ldots, T$ the simulator can determine the unchangeable bit $b[k]$ and compute $b = b[1] \oplus \ldots \oplus b[T]$. It is clear that the receiver cannot obtain $M_{b[i]\oplus 1}[i]$ from these execution ensembles k.

The simulator next sends the extracted bit b to the functionality in the ideal model to retrieve the original message m_b of the honest sender. Now it prepares the answers $\tilde{M}_0[i, j], \tilde{M}_1[i, j]$ for the final execution (i, j) by adding m_b via exclusive-or to the values at positions r with $r[1] \oplus \ldots \oplus r[T] = b$ in $\tilde{M}_{b[i]}[i, j]$ and $\tilde{M}_{b[i]\oplus 1}[i, j]$ and leaving any other values unchanged. It returns the adapted value $\tilde{M}_0[i, j]$ or $\tilde{M}_1[i, j]$ chosen by the receiver (or both if it replies with a `dump`-message to a `panic`-message).

Note that xoring m_b to the corresponding values in $\tilde{M}_{b[i]}[i, j]$ and $\tilde{M}_{b[i]\oplus 1}[i, j]$ consistently changes $\tilde{m}_b = 0^\kappa$ to m_b, and thus perfectly simulates an ideal execution for these parts. Also, since there must exist an execution with an honest helper in each ensemble k, the adversary misses at least one of the t pieces $\tilde{M}_{b[k]\oplus 1}[k, l]$ for some l. This implies that $\tilde{M}_{b[k]\oplus 1}[k]$ for all k and therefore $\tilde{m}_{b\oplus 1}$ are hidden information-theoretically from the adversary, and the distribution of the fake values $\tilde{M}_{b[k]}[k, l]$ are perfectly indistinguishable from genuine values for the unknown $m_{b\oplus 1}$.

If the adversary asks to corrupt the sender then the simulator corrupts the sender in the ideal world to learn both messages m_0, m_1. If there is still an open ensemble i then the

adversary has not yet learned $M_0[i,j], M_1[i,k]$ for some known j,k. Take the smallest of such indices j,k and add m_0 to all positions r in $M_0[i,j], M_1[i,l]$ with an even number of 1-bits, and m_1 for those r's with an odd number of 1-bits. This adapts the values from $\tilde{m}_0, \tilde{m}_1 = 0^\kappa$ to m_0, m_1 in a completely indistinguishable way.

Next assume that the sender is getting corrupted and that all ensembles are closed. If the receiver is still honest then the simulation is trivial as the adversary has not yet seen any values sent form $\mathcal{F}_{\mathrm{OT}_3^*}$ to honest parties. We therefore presume that the receiver is controlled by the adversary. As the ensembles are closed this in turn implies that m_b has already been sent to R, that the simulator knows the receiver's choices $b = b[1] \oplus \ldots \oplus b[T]$ in the ensembles, and that $\tilde{m}_b$ and the related values have been changed to m_b already.

In addition, for any $i = 1, 2, \ldots, T$ the adversary lacks knowledge of some piece $\tilde{M}_{b[i]\oplus 1}[i,j]$. For each $r \in \{0,1\}^T$ with $r[1] \oplus \ldots \oplus r[T] = b \oplus 1$, the simulator now chooses the smallest index i with $r[1] = b[1], \ldots, r[i-1] = b[i-1]$ but $r[i] = b[i] \oplus 1$. For this index, and the missing piece $\tilde{M}_{b[i]\oplus 1}[i,j]$, the simulator adds $m_{b\oplus 1}$ to the position r. Note that such an index i must exist and alters $\tilde{m}_{b\oplus 1} = m_{b\oplus 1}$ via the entries in the previously hidden pieces $\tilde{M}_{b[i]\oplus 1}[i,j]$. Then, revealing these adapted values mimics a corruption in the real world perfectly.

Simulation of Receiver. Next we discuss how to simulate an honest receiver. Similar to the previous case the simulator uses $\tilde{b} = 0$ as input and follows the prescribed program of R. That is, the simulator divides $\tilde{b}$ into $\tilde{b} = \tilde{b}[1] \oplus \ldots \oplus \tilde{b}[T]$ and uses $\tilde{b}[i]$ in the i-th ensemble. Similar to the previous case we presume that the sender is malicious, else the adversary does not see any information about the execution in the hybrid model.

The simulator now has to provide functionality $\mathcal{F}_{\mathrm{OT}}$ in the ideal model with two input messages of which one will be delivered to the honest receiver. The simulator, however, does not know beforehand the receiver's choice in the ideal setting, and must therefore be able to extract both messages from the adversarial controlled sender in the simulation.

In each execution ensemble i the simulated receiver retrieves a string $M_{\tilde{b}[i]}[i]$ from the malicious sender. In some ensembles the adversary may successfully call a `receiver-panic` to learn the bit $\tilde{b}[i]$. Yet, by the constructions of the T sets, there must exist a set in which all t helpers remain uncorrupted (forever). In this ensemble k any `panic`-request is and will be ignored, and the simulator waits till the sender transmits $(M_0[k,l], M_1[k,l])$ in all executions $l = 1, 2, \ldots, t$ to functionality $\mathcal{F}_{\mathrm{OT}_3^*}$. From these values it assembles $M_0[k]$ and $M_1[k]$. The values $M_{\tilde{b}[i]}[i]$ for $i \neq k$ in combination with $M_0[k], M_1[k]$ enable the simulator to recover $m_{\tilde{b}[1]\ldots 0\ldots \tilde{b}[T]}$ and $m_{\tilde{b}[1]\ldots 1\ldots \tilde{b}[T]}$, yielding messages m_0 and m_1. The simulator submits these two values in the ideal model on behalf of the dishonest sender.

Note that the set of values $\tilde{b}[i]$ for $i \neq k$ in the simulation has the same distribution as an actual choice by the honest receiver in a protocol execution; only the distribution of value $b[k] = b \oplus b[1] \oplus \ldots \oplus b[T] = b \oplus \tilde{b}[k]$ could possibly be different. Yet, the adversary and the environment cannot tell these choices apart because value $\tilde{b}[k]$ is perfectly hidden. Furthermore, given values $\tilde{b}[i] = b[i]$ for $i \neq k$ the receiver in an actual protocol execution would receive either m_0 or m_1 (depending on b and $b[k]$). Here, m_0, m_1 are the same messages the simulator submits in the ideal model such that the receiver gets the same message m_b in the ideal model. Hence, in the simulation and in the ideal world the receiver outputs the same message m_b.

Finally, it remains to discuss corruption requests for the receiver. If the adversary wants to take control over the receiver then there exists an execution ensemble k in which some execution (k,l) is still open or, if all executions are closed, in which all auxiliary parties are and will be honest. In any case, at least one of the bits $\tilde{b}[k]$ has again remained hidden from

the adversary so far. The simulator, learning the receiver's input b in the ideal model, can now substitute $\tilde{b}[k]$ such that $b = \tilde{b}[1] \oplus \ldots \oplus \tilde{b}[T]$ before handing these values to the adversary in the simulation. Since the adversary has been perfectly oblivious about $\tilde{b}[k]$ this simulation of the corruption is perfect, too.

Simulation of the helper parties is again easy as they do not actively participate in the protocol. This shows that the overall simulation is perfectly indistinguishable. □

B Homomorphic Commitment Protocol

In this section we present our UC homomorphic commitment protocol.

Protocol HCom for Abelian group $(\mathcal{A}, +)$ in the $(\mathcal{F}_{\text{sig}}, \mathcal{F}_{\text{smt}}, \mathcal{F}_{\text{auth}})$-hybrid model

– **Commitment:**
- Upon receiving (`hcom-commit`, *sid*, $P_i, P_j, x_1, \ldots, x_n$) the committer first generates a verification key vk of $\mathcal{F}_{\text{sig}}$ and sends it to the receiver P_i and to all $2t$ helper parties via $\mathcal{F}_{\text{auth}}$. The helper parties echo the key to the receiver over $\mathcal{F}_{\text{auth}}$ and the receiver stops if any of the keys do not match.
- The committer then computes shares $x_{\ell,1}, x_{\ell,2}, \ldots, x_{\ell,2t}$ of all values $\ell = 1, 2, \ldots, n$ as well as signatures σ_k of $(x_{1,k}, \ldots, x_{n,k})$ via $\mathcal{F}_{\text{sig}}$ for $k = 1, 2, \ldots, 2t$. It sends $(\textit{commit}, x_{1,k}, \ldots, x_{n,k}, \sigma_k)$ to party P_k over $\mathcal{F}_{\text{smt}}$ (where P_k is the k-th party different from the committer).
- Receiving $(\textit{commit}, x_{1,k}, \ldots, x_{n,k}, \sigma_k)$ from P_i party P_k first checks the signature σ_k by $\mathcal{F}_{\text{sig}}$. If the signature is invalid then P_k sends out $(\textit{commit-error}, n)$ to the receiver over $\mathcal{F}_{\text{auth}}$. Otherwise, if the signature is valid, then party P_k sends $(\textit{commit-ok}, n)$ to P_j over $\mathcal{F}_{\text{auth}}$.
- Only if the receiver gets $2t$ messages $(\textit{commit-ok}, n)$ from $P_1, \ldots, P_{2t}$ then it outputs (`hcom-receipt`, *sid*, P_i, P_j, n).

– **Opening:**
- When getting (`hcom-open`, *sid*, $P_i, P_j, a_1, \ldots, a_n$) as input, the committer P_i computes $y \leftarrow \sum_{\ell=1}^{n} a_\ell x_\ell$ and $y_k \leftarrow \sum a_\ell x_{\ell,k}$ for $k = 1, 2, \ldots, 2t$. Additionally, it computes signatures τ_k of $(a_1, \ldots, a_n, y_k)$. Send $(\textit{open}, a_1, \ldots, a_n, y_k, \tau_k)$ over $\mathcal{F}_{\text{smt}}$ to each P_k.
- Party P_k, when receiving $(\textit{open}, a_1, \ldots, a_n, y_k, \tau_k)$, verifies that $y_k = \sum a_\ell x_{\ell,k}$ for the previously received values and also checks the signature τ_k for $(a_1, \ldots, a_n, y_k)$. If all tests succeed then it sends $(\textit{open-ok}, a_1, \ldots, a_n, y_k, \tau_k)$ to P_j over $\mathcal{F}_{\text{smt}}$; else it sends $(\textit{open-error}, a_1, \ldots, a_n, x_{1,k}, \ldots, x_{n,k}, \sigma_k)$, i.e., reveals all shares including the signature.
- The receiver waits to receive $2t$ messages including values $(a_1, \ldots, a_n)$, either of type *open-ok* or of type *open-error*. Having received such values the receiver checks the signature in each message with the help of $\mathcal{F}_{\text{sig}}$ and vk, i.e., the receiver verifies that τ_k is valid or that σ_k is valid. If so, for each *open-error* message the receiver reconstructs $y_k \leftarrow \sum a_\ell x_{\ell,k}$ and checks with these parts and the ones in *open-ok* that the reconstruction algorithms yields $y \neq \perp$. If all tests succeed then the receiver outputs (`hcom-open`, *sid*, $P_i, P_j, a_1, \ldots, a_n, y$).

Fig. 9. Universally Composable Homomorphic Commitment Scheme