

On Robust Combiners for Oblivious Transfer and other Primitives

Danny Harnik* Joe Kilian† Moni Naor* Omer Reingold* Alon Rosen‡

Abstract

A (1,2)-robust combiner for a cryptographic primitive \mathcal{P} is a construction that takes two candidate schemes for \mathcal{P} and combines them into one scheme that securely implements \mathcal{P} even if one of the candidates fails. Robust combiners are a useful tool for ensuring better security in applied cryptography, and also a handy tool for constructing cryptographic protocols. For example, we show how robust combiners are useful for obtaining universal schemes for cryptographic primitives (a universal scheme is an explicit construction that implements \mathcal{P} under the sole assumption that \mathcal{P} exists).

In this paper we study what primitives admit robust combiners. In addition to known and very simple combiners for one-way functions and equivalent primitives, we show robust combiners for protocols in the world of public key cryptography, namely for Key Agreement (KA).

The main point we make is that things are not as nice for Oblivious Transfer (OT) and in general for secure computation. We prove that there are no “black-box” robust combiners for OT. Since all of the known combiners for other primitives are black-box ones, this impossibility result essentially separates OT and secure computation from the primitives that do admit combiners. On the positive side we show a black box construction of a (2,3)-robust combiner for OT.

At the mouth of two witnesses ... shall the matter be established
Deuteronomy Chapter 19.

1 Introduction

Not putting all your eggs in one basket is commonly considered good advice and this should be no different in cryptography. Suppose that we have a two cryptographic schemes that we generally trust to be secure for some task. It makes a lot of sense to try and combine these two into one scheme that is guaranteed to be secure even in the unfortunate case that one of the two original schemes was broken. For example, we have several encryption schemes that are based on various unproven number theoretic assumptions, such as the hardness of factoring or of computing discrete logarithms. We would like to combine these into one encryption scheme that is secure if at least one of these unproven assumptions happens to be true. We call such a construction a *Robust Combiner*.¹ This is a scheme that combines two different schemes and is robust to the failure of just one of them.

*Dept. of Computer Science and Applied Math., Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: {danny.harnik, moni.naor, omer.reingold}@weizmann.ac.il.

†Computer Science Department, Rutgers University. E-mail: jkilian@cs.rutgers.edu.

‡DEAS, Harvard, Cambridge MA, E-mail: alon@eecs.harvard.edu.

¹This notion is called a *Tolerant Construction* in [17].

Definition 1.1 ((k, n) -Robust Combiner (Informal)) *A (k, n) -Robust Combiner for a cryptographic primitive \mathcal{P} is a construction that takes n candidate schemes for \mathcal{P} and combines them into one scheme such that if at least k of the candidates indeed implement \mathcal{P} then the combiner also implements \mathcal{P} .*

In general, the most interesting combiners are $(1, 2)$ -robust combiners as they are essential and at times sufficient for constructing $(1, n)$ -robust combiners (n is some parameter, typically related to the security parameter). For ease of notations we will sometimes write just robust combiner or simply combiner when we actually mean a $(1, 2)$ -robust combiner.

Robust combiners are by all means not new in cryptography. Several practical constructions try to combine several primitives to achieve stronger security guarantees. For example, Asmuth and Blakely [1] suggest a method of combining two encryption schemes of which only one can be trusted. Another example is the widely used idea of repeatedly encrypting a message several times with different keys in order to enhance security, an idea that dates back as far as Shannon [24] and found in many applications since. This relates to combiners as security holds in the case that the integrity of some of the keys is compromised, but at least one remains secure. Also, Herzberg [17] discusses the notion of combiners explicitly.

There are plenty of other practical motivations for combiners, we briefly give a few: For example, using software from a few sources that are not entirely trusted (for instance when running an election and using electronic ballots from a few vendors). Combiners can also be used to avoid bugs in software, rather than checking the correctness of a software (as in [5]), combine several different versions, hoping that at least one is correct. One can also consider physical sources used for cryptography (e.g. noisy channels) that cannot necessarily be trusted.

From the point of view of theoretical cryptography robust combiners are also valuable. Combiners are useful tools in constructions and reductions between cryptographic primitives. This happens in scenarios where it is guaranteed that one of several constructions exist. We give two examples:

- Levin [19] (see exposition in [13]) introduced a *Universal-one way function (OWF)* which is an explicit construction that is guaranteed to be a OWF under the sole assumptions that one-way functions exist at all. The property of one-way functions that allows for this universal constructions is the fact that they admit robust combiners.
- In the construction of pseudo-random generators (PRG) from OWFs by Hastad et al. [16] a polynomial number of candidates for PRG are given, one of which is guaranteed to be a PRG. These are then combined (the combiner is a simple XOR of the output) into one PRG construction.

1.1 Our Contributions

In this paper we study what cryptographic primitives have or don't have robust combiners. We start by showing that simple robust combiners exist for OWF (this is common knowledge) and its equivalents (such as private key encryption, pseudo-random generators, functions and permutations, digital signatures and bit commitment). We then present a robust combiner for Key Agreement (KA) and, similarly, Public Key Encryption (PKE).

On Robust Combiners for Oblivious transfer: The abundance and relative simplicity of robust combiners may lead to the belief that all primitives have simple combiners. However, this is not the case for the fundamental oblivious transfer primitive (OT) and thus for any non trivial task of secure computation. We define the notion of black-box combiners, giving several refinements to this notion. Our main result shows the following:

Theorem 1.2 *There exists no “transparent black-box” construction of a robust OT-combiner.*

Transparent black-box combiners are black-box combiners with a specific property. In general, it is required that every time a party calls one of the candidates, then the other party learns about this call (all messages generated by the candidate are actually sent to the other party).

A crucial point is that all the known examples of combiners (whether simple or more involved) are transparent black-box combiners. Thus this result can be viewed as a strong indication of the hardness of the problem. Theorem 1.2 hence gives a separation of OT along with essentially all of secure computation from the other cryptographic primitives mentioned above.

Positive results for OT: On a more positive note, we show a very efficient black box construction of a (2,3)-robust OT-combiner. We also point out that it is easy to construct an OT protocol based on the assumption that at least one of the assumptions regarding factoring or the discrete logarithms is correct. This is because there are known constructions of OT from such assumptions (and in general from any trapdoor permutation [12]) that have perfect (and guaranteed) security for the receiver, in which case constructing combiners is simple.

(1,n)-robust combiners and universal schemes: We discuss the notion of a universal scheme for a cryptographic primitive (following Levin’s [19] universal OWF) and show that primitives that admit (1,n)-robust combiners also have universal schemes. We then study cases where (1,2)-combiners are sufficient (1,n)-combiners. Among others, it is shown that a (1,2)-robust combiner for OT also gives a construction of a universal scheme for OT (the construction makes use of the efficient (2,3)-robust combiner for OT shown here).

Other points:

- We define robust combiners and discuss the notion of black-box combiners. For interactive primitives (such as OT and KA) we give a refinement of this notion. In particular, we define three forms of black-box combiners, each being a restriction of the previous.
- A delicate point when discussing combiners for a primitive \mathcal{P} is the question of functionality. In some settings, while one of the input candidates is guaranteed to be secure, the other one is not even guaranteed to have the functionality of \mathcal{P} , making things more involved. In general, one way to overcome this is by first testing the functionality of a possibly faulty candidate. For instance, the combiner for KA first constructs a KA where the two parties agree only with reasonably high probability, and then reduces the probability of disagreement to a negligible one using an error correcting code.

1.2 Related Work

As mentioned before, robust combiners have already been used and studied. In particular the work of Herzberg [17] also focuses on robust combiners in cryptography. This work puts more emphasis on efficiency and specifically the use of the parallel and cascade constructions as combiners and shows combiners for various primitives including OWF, signatures, MACs and others.

Implicit use of combiners is abundant. For example, the idea of using multiple encryptions is widely used in practice. This practice is in fact advocated in the NESSIE consortium recommendations [21]. Lately Dodis and Katz [11] studied the use of multiple encryptions with respect to CCA2 security, giving a robust combiner for CCA2 secure encryption schemes. Another related concept is given in Brickell and McCurley [6] and Shoup [25] that show schemes that achieve two different types of security based on two different number theoretic assumptions.

The work of Damgard, Kilian and Salvail [10] is somewhat relevant to the OT-combiner. This work discusses a weak version of OT called (p, q) -OT that has probability p of compromising the sender's security and probability q of compromising the receiver's. It is shown that a fully secure OT can be constructed from a (p, q) -OT if and only if $p + q < 1$. In our work two candidates for OT are given, one can obtain a (p, q) -OT with $p = q = \frac{1}{2}$ simply by choosing one of the candidates at random. Therefore, the impossibility result of [10] for $p + q \geq 1$ gives some intuition for the impossibility of OT-combiners. However the result for $p + q \geq 1$ relies heavily on the fact that the errors p and q are assumed to be uncorrelated events, which is not the case in the setting of combiners. On the other hand, for $(2,3)$ -robust combiners, we can get an OT protocol with $p = q = \frac{1}{3}$ and use the reduction from [10] (although the $(2,3)$ -robust OT-combiner presented here is much more efficient, a property that is used in Section 5.1).

1.3 Paper Organization

Section 2 contains definitions of cryptographic primitives and of robust combiners. In Section 3 we present some general observations regarding combiners and specific combiners for the OWF and KA primitives. Section 4 discusses combiners for OT, showing the impossibility of transparent black box constructions and a construction of a $(2, 3)$ -robust combiner for OT. In Section 5 we discuss $(1, n)$ -combiners and their relation to universal schemes. Finally, some open problems are discussed in Section 6. Due to space limitations, some proofs and discussions were moved to the Appendix.

2 Notations and Definitions

2.1 Cryptographic Primitives

We denote by *PPTM* a probabilistic polynomial time Turing machine.² An Oracle PPTM is a PPTM that also has access to one or more oracles.

We start by defining a cryptographic primitive following the outline of [23]. This definition attempts to capture a wide range of cryptographic primitives. In general, the definition includes a description of the functionality of the primitive, along with a definition of security. The functionality defines what the primitive should do, for example, in a one way permutation, the functionality is merely the family of all permutations. Typically, a function from this family should be computable by a PPTM. The security deals with the ability of an adversary of a certain class (e.g. the class of all PPTM) to learn something from an implementation. This is modelled by a relation between possible machines (modelling the adversary) and functions (modelling the implementation). The relation defines when a machine *breaks* an implementation. For an implementation to be secure, it is required that no machine in the class of adversaries can break this implementation. Formally:

Definition 2.1 (Cryptographic Primitive) A primitive \mathcal{P} is a triplet $\langle F_{\mathcal{P}}, \mathcal{A}_{\mathcal{P}}, R_{\mathcal{P}} \rangle$, where $F_{\mathcal{P}}$ is a set of functions $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ defining the functionality of \mathcal{P} , $\mathcal{A}_{\mathcal{P}}$ is the class of adversary machines and $R_{\mathcal{P}}$ is a relation over pairs $\langle f, A \rangle$, including machines $A \in \mathcal{A}_{\mathcal{P}}$ that **break** functions $f \in F_{\mathcal{P}}$. We say that f **implements** \mathcal{P} if $f \in F_{\mathcal{P}}$ and is computable by a PPTM. A **secure implementation** is an f that no $A \in \mathcal{A}_{\mathcal{P}}$ breaks. The primitive \mathcal{P} **exists** if there exists an implementation of \mathcal{P} that is secure.

²The definitions given in this paper generally view adversaries as uniform machines, though all of the results in this paper apply for definitions of security against non-uniform adversaries as well.

Interactive Primitives: The above definition clearly captures primitives such as one-way functions and permutations, but also captures the more involved *interactive primitives* such as key agreement protocols. In such cases the functionality f can be seen as divided into two parts: The “next message” function M (where the function computes the next message to be sent when given all the previous history known to the player sending the message) and an output function O (that is the view of one of the parties of the whole interaction and outputs the parties local output). A protocol is then generated by each of the sides alternately generating their next message by calling the function M given their local inputs, randomness and the history of the interaction up to that point. At the end of interaction each sides gives their view to the function O to get their output. The functionality in the key agreements is therefore formalized by the family functions $f = (M, O)$ that generate a key agreement protocol (where the two side end with the same output).

Standard definitions of relevant primitives (KA and OT) are given in Appendix A.

2.2 Robust Combiners

We now formally define the central concept of this paper. The combiner should receive as input candidates for implementing a primitive \mathcal{P} . In this context, the representation of these candidates is crucial. For example, the candidates can be given as the code of a PPTM or alternatively via an oracle that implements it.

Definition 2.2 ((k, n) -Robust Combiner) *Fix a representation for (candidate) schemes for a cryptographic primitive \mathcal{P} . A (k, n) -Robust Combiner for \mathcal{P} is a PPTM that gets n candidate schemes as inputs and implements \mathcal{P} such that:*

- *If at least k candidates securely implement \mathcal{P} then the combiner also securely implements \mathcal{P} .*
- *The running time of the combiner is polynomial in the security parameter m , in n and in the lengths of the inputs to \mathcal{P} .³*

Note that a general combiner may totally ignore its inputs and simply implement \mathcal{P} directly. However, we are interested in combiners whose security relies on the security guarantees of the candidates (as is the case with fully black-box combiners, discussed next).

We introduce three types of black-box combiners that are of interest.

Definition 2.3 (Black-Box (BB) Combiners)

- **A (fully) black-box (1,2)-robust combiner** *is a combiner where:*
 1. *The implementation is black-box: The combiner is an oracle PPTM given access to the candidates via oracle calls to their implementation function.*
 2. *The proof is black-box: For every candidate there exists an oracle PPTM R^A (with access to A) such if adversary A breaks the combiner, then the oracle PPTM R^A breaks the candidate.⁴*
- **A transparent black-box combiner** *is a fully black-box combiner for an interactive primitive, where every call to a candidate’s next message function M is followed by this message being sent to the other party.*

³We make the implicit assumption that the candidates themselves run in polynomial time. See a further discussion in Section 3.1.

⁴In the case of (n, k) -robust combiners then there are at least $n - k + 1$ candidates that can be broken in this manner.

- **A third party black-box combiner** is a fully black-box combiner where the candidates behave like third parties. The candidates give no transcript to the players, but rather take their inputs and return outputs.

In the case of non-interactive primitives such as one-way functions the three notions defined above are equivalent. However for interactive primitives they form a distinct hierarchy of black-box combiners. The third party black box combiner is the most natural one, that ignores the implementation and protocol and simply relies on the functionality and security of one of the candidates (for example, the combiner for KA). However, in some situations this is insufficient and a transcript is actually needed (for example, constructing a OWF function cannot be done from a third party implementation for OT). Therefore, we relax this notion to the notion of a transparent BB combiner that is allowed access to the transcripts of the protocols (an example is the combiner for bit commitment). The most general fully BB combiner is given unlimited off-line access to the oracles that generate the protocol. Although these combiners may be more powerful, at this point, we do not have examples of combiners that are not third party or transparent black-box combiners.

3 Positive Results

3.1 The General Framework for Robust Combiners

Cryptographic primitives are mainly about security, as this is what distinguishes them from general functions and protocols. So naturally the emphasis when constructing robust combiners will be that these primitives indeed remain secure in face of the unfortunate case that one of the candidates actually breaches security. However, there are some subtleties that need to be discussed. In some settings, hardly anything is known about the candidates at hand other than the fact that one of them is good. Specifically, only one candidate is guaranteed to have the intended functionality. For example, a faulty candidate for a OWF, may not only be easy to invert, but might also be hard to compute in the easy direction (computing the function might be impossible for all PPTM). Other primitives might have additional functionalities (other than running time) that should be taken into consideration. For example, in the KA (key agreement) both parties should output the same key (the agreement). In this section we present approaches for dealing with these issues, dealing separately with running time and other functionalities:

Running time: In general, one cannot expect to be able to check that a candidate for a cryptographic primitive always halts in polynomial time unless the specific polynomial bound on the running time is known in advance. We therefore assume that the polynomial bound is given as input to the scheme. For example, a robust OWF-combiner gets as input a polynomial $p(\cdot)$ and the security parameter 1^m along with the two candidates f_A, f_B . Now, when a combiner invokes a candidate, it allows it to run for at most $p(m)$ steps, and if it does not halt then the output of the candidate is set to some fixed value (e.g. to the all zero string).

Thus when presenting a combiner, we assume that the combiner also gets as an input a polynomial $p(\cdot)$ that is the bound on the running time of the good candidate. However, unless it is relevant, we omit this parameter from the text and simply assume that the running time of all candidates is polynomial, a fact that is essential for most proofs of security. It should be noted that in many applications of combiners this bound can be found by analyzing the candidates programs. In other cases, the mere knowledge that some polynomial bound exists suffices in order to get a bound (an example of this case is given in the construction of universal primitives in Section 5.1).

Functionality Test: A possible approach for testing the functionality of a candidate (such as the agreement in key agreement or the transfer of the chosen secret in oblivious transfer) is presented. This method may sometimes be helpful but at other times impossible, depending on the specific primitive at hand. The idea is as follows: Each party simulates n^2 random executions of the candidate, and accepts only if the candidate always satisfies its defined functionality. For example, in a key agreement, each party simulates a random execution by playing the roles of both players and checking whether they agree. A candidate is executed according to its inputs and the randomness of the parties involved. Denote these by the string x . Denote by :

$$X_{bad} = \{x \mid \text{candidate } f \text{ fulfills functionality when running with } x\}$$

Then we have that:

$$\Pr[\text{Candidate } f \text{ passes test} \mid \Pr_x(X_{bad}) > \frac{1}{n}] \leq (1 - \frac{1}{n})^{n^2} \leq O(2^{-n})$$

After passing such a test we are assured that with overwhelming probability, the candidate does what it is supposed to with probability at least $1 - \frac{1}{n}$. While this is a rather weak guarantee, it is sometimes sufficient (as in the case of KA-combiners, see Section 3.3).

Note: The functionality and time tests may not be necessary in many specific contexts. For example, when trying to combine two constructions based on two different computational assumptions, the functionality and running time are already guaranteed by the design of these constructions. These tests are necessary however in the general case where nothing is known (for example, in the case of universal schemes (Section 5.1)).

3.2 Robust Combiners for OWFs and Equivalent

It has long been known that one-way functions (OWF) have simple robust combiners. For example, notice that simple concatenation of the OWF candidates on independent inputs suffices (as pointed out in [17]). More precisely, given candidates f_A and f_B , denote $F(x, y) = f_A(x) \parallel f_B(y)$ (where f_A and f_B run in polynomial time).

Lemma 3.1 *F is a robust OWF-combiner.*

Proof: Suppose that there exist an inverting PPTM C such that $C(F(x, y)) \in F^{-1}(F(x, y))$ with polynomial probability, then this C can be used to invert both f_A and f_B . For instance to invert $f_A(x)$ simply choose a random y and run $C(f_A(x) \parallel f_B(y))$. Note that this is easy due to the fact that f_B is assured to run in polynomial time. The output (x', y') is in $F^{-1}(F(x, y))$ with polynomial probability and therefore also $x' \in f^{-1}(f(x))$. \square

Lemma 3.1 means that in fact all of the primitives that were shown to be *equivalent* to OWF have robust combiners. By equivalent we mean, primitives that have reductions to and from OWFs. Some of the more noteworthy equivalent primitives are:

- semantically secure private key encryption.
- pseudo-random generators, functions and permutations.
- digital signatures.
- bit commitment.

The combiners for these primitives follow since given two candidates for primitive \mathcal{P} (from the list above), one can use the reduction from OWF to \mathcal{P} to create two candidates for OWFs. These two are then combined using the OWF-combiner, and this is used to construct the primitive \mathcal{P} from a OWF (with the opposite reduction from \mathcal{P} to OWFs).

Note, however, that for most of these primitives going via the reductions to and from OWF is an overkill, and much more efficient and direct combiners can be found. For example a combiner for pseudo-random generator is simply one that XORs the outputs (thus the heavy reduction of [16] from pseudo-random generators to OWFs may be avoided). An exception is the case of bit commitments for which we are only aware of the combiner via the OWF. Unlike the non-interactive primitives in the list (that have third party BB combiners), the combiner for commitment is a transparent BB combiner and highly inefficient (this issue is further discussed in Section 6).

3.3 Robust Key Agreement Combiner

Theorem 3.2 *There exists a robust KA-combiner scheme. The combiner reaches agreement with all but a negligible probability. Furthermore, its round complexity is at most that of the candidate with the higher number of rounds.*⁵

Observe that a KA-combiner can be easily achieved if the functionality of both candidates is guaranteed. The KA-combiner simply outputs an XOR of the outputs in the two candidates. If the functionality is not guaranteed, then the combiner for KA is constructed in two stages. First we show a KA-combiner with *relaxed agreement*, (a protocol in which the parties agree with all but a polynomially small fraction). Then this is turned into a KA where the agreement happens with overwhelming probability using an error correction code.

Proof: A KA-combiner with relaxed agreement can be easily achieved after running the functionality test. The relaxed KA-combiner simply takes an XOR of the outputs in the two candidates. It suffices that one of these keys is indistinguishable from random in order for their XOR to be indistinguishable from random as well. In the resulting protocol the parties might not always agree on the same key but because of the functionality test this happens with probability at most $1 - \frac{1}{n}$.

Next we show that this can be turned into a key agreement that has agreement with overwhelming probability.

Lemma 3.3 *If there exists a KA protocol with relaxed agreement, then there exists a KA with agreement up to a negligible error.*

Proof: We point out that a KA can easily be modified so that one side will choose the key to be agreed upon. Suppose that Alice ends up with key K_A and Bob ends with key K_B . We can assume w.l.o.g that Alice ends the protocol with relaxed agreement. Alice chooses the target Key K'_A and in the last round of communication, and sends to Bob $K' = K_A \oplus K$. Bob's key will be $K'_B = K' \oplus K_B$. If $K_A = K_B$ then also $K'_A = K'_B$ and clearly the modified protocol maintains the secrecy of the key. The new protocol will use any Error Correcting Code *ECC* that can recover from a constant fraction of errors (e.g. $\frac{1}{4} - \delta$ errors, for constant δ).

1. Alice chooses a Key K and encodes it using the ECC into a string $E(K)$ of length m (a polynomial in $|K|$).
2. For each $i \in [m]$, the two sides run an independent relaxed KA protocol where Alice chooses the key to be $E(K)_i$ (the i th bit of $E(K)$). Let Bob's output in this relaxed KA be the bit W_i . Altogether Bob outputs an m bit string W .
3. Bob decodes the string W and outputs the result $D(W)$

⁵By round complexity we mean the worst-case round complexity.

As long as the string W has hamming distance of less than $\frac{m}{4}$ then the decoding $D(W) = K$ and we get an agreement. Since the relaxed KA agrees with high probability (every bit in W_i is different than $E(K)_i$ only with probability at most $\frac{1}{n}$), then the probability of having at least $\frac{m}{4}$ errors is exponentially small (using a Chernoff bound). Thus agreement is achieved with overwhelming probability. The security follows from the fact that an efficient eavesdropper cannot distinguish between two legitimate outcomes of the series of key agreements (by a hybrid argument), and specifically, cannot distinguish between two codewords $E(K)$ and $E(K')$. \square

The relaxed KA runs in the same number of rounds as the candidates, as the candidate protocols can run in parallel and the extra XOR operation does not require any messages. Moving to a KA where one of the parties chooses the output key requires one more message. But this message can be united with the last message in the protocol (the party sending the last message already knows the output before sending it and can therefore create the mask K and incorporate it into his last message). Finally the reduction to agreement with negligible error probability can also run all of the protocols in parallel and not increase the number of rounds. \square

We note that the KA-combiner is a third party BB combiner. Also, it is known that a 2 message KA protocol is equivalent to semantically secure (against chosen plaintext attacks) Public Key Encryption (PKE), and since the KA-combiner maintains the same round complexity, we also get for free a robust PKE-combiner.

4 On Robust Combiners for Oblivious Transfer

4.1 Impossibility of Black Box Robust OT-Combiner

In contrast to all the other primitives mentioned here that had very simple combiners, the situation of OT is left open. We do not know of any OT-combiner, simple or complicated. The following result indicates that this is indeed a much harder problem:

Theorem 4.1 *There exists no construction of a transparent black-box robust OT-combiner.*

Recall that a transparent black-box combiner (defined in Section 2) is one in which the candidates are given as a pair of oracles (the “next message” oracle and the output oracles). Whenever one of the parties calls a next message oracle it is required to send the message generated to the other party. We note that it is considerably simpler to show that no third party BB combiner exists for OT. However, we work a bit harder in order to capture the notion of transparent BB combiners, and in particular combiners that can also use the transcript of the protocol. This is important since this wider notion of black box combiners covers all the robust combiners that we are aware of.

Proof:

In similar fashion to a many black box impossibility results, starting with the seminal paper of Impagliazzo and Rudich [18], Theorem 4.1 is proved by trying to show a “world” in which OT exists, but OT-combiners do not. The argument however must be changed, since in every world that has OT, an OT-combiner does exist, simply by running the correct OT protocol. Instead, the actual proof shows two worlds such that every transparent black-box OT-combiner is insecure in at least one of them (we show this even in the semi-honest model⁶).

An OT oracle: Before we present the two worlds, we present an oracle that enables the execution of an OT protocol. This oracle is called here an *OT oracle* and is composed of a triplet of functions $OT = (f_1, f_2, R)$ as follows:

⁶Recall that in the *Semi-Honest* model the parties involved follow the protocol as prescribed, and perhaps later try to learn more information than what they were intended to.

- f_1 is a length tripling random function that takes the receiver's choice bit c and randomness r_R and outputs $m_1 = f_1(r_R, c)$ that is used as the receiver's message.
- f_2 is also a length tripling random function that takes the sender's inputs s_0, s_1 and randomness r_S and the receiver's message m_1 and outputs the sender's message $m_2 = f_2(r_S, s_0, s_1, m_1)$.
- R is called by the receiver, it takes m_2 along with r_R and c and outputs the secret s_c (if the inputs are consistent).

The above process constitutes a secure OT protocol (as defined in Appendix A), since the receiver learns the secret of his choice but since the parties cannot invert the random functions the messages can simply be simulated by random messages. Moreover, this is true even in the presence of a PSPACE-complete oracle. This is made formal in the following claim (proved in Appendix B):

Claim 4.2 *The procedure defined by the oracle (f_1, f_2, R) is a secure OT protocol even in the presence of a PSPACE-complete oracle.*

In addition to the functions constituting an OT oracle, we add another oracle for breaking such an OT. This oracle simply inverts the functions f_1, f_2 , and thus leaks both secrets to the receiver and the choice bit to the sender.⁷

The two worlds: We can now define the two worlds we discuss.

- **World 1**, contains:
 1. A PSPACE-complete oracle.
 2. Two sets of OT oracles, denoted $OT_A = (f_1^A, f_2^A, R^A)$ and $OT_B = (f_1^B, f_2^B, R^B)$.
 3. The oracle Inv_A for inverting OT_A .
- **World 2**, contains:
 1. A PSPACE-complete oracle.
 2. Two sets of OT oracles, denoted $OT_A = (f_1^A, f_2^A, R^A)$ and $OT_B = (f_1^B, f_2^B, R^B)$.
 3. The oracle Inv_B for inverting OT_B .

Clearly an OT protocol exists in each of the two world via the one OT oracle that remains secure. Now consider a robust OT-combiner that is given OT_A and OT_B as candidates. By the definition of the combiner, it should securely implement an OT protocol in each of the two worlds. Our goal is therefore to show that every OT-combiner (that is transparent black-box) is insecure in at least one of the two worlds. This is done by showing an attack on any given OT-combiner.

Simulating the combiner in the bare world (bare-OT): For every transparent black-box combiner we give a matching protocol in the bare world that contains just the PSPACE oracle but not the OT oracles. First we explain how a party can simulate an OT oracle. A party simulates an oracle by answering every query to the functions f_1 or f_2 by a random value. In addition, the party records all the answers he gave to queries during the protocol's execution. When the function R of the OT oracle is queried, the party simply inverts the functions using the records he stored in memory, allowing him to reply with the proper answer.⁸ The new protocol in the bare world

⁷This inverting oracle is possible since with overwhelming probability f_1 and f_2 are one-to-one functions.

⁸We assume here that the OT oracle answers a \perp whenever an illegal input is given. The simulator simply does the same when he gets a query with an input that was not previously in his memory (and thus not a legal input).

imitates the combiner with the exception that the sender simulates the oracle OT_A and the receiver simulates OT_B .

The first thing to notice is that for every execution of bare-OT there are choices for OT_A and OT_B that give an identical execution of the combiner using the real OT_A and OT_B . Thus bare-OT indeed has the functionality of an OT protocol (perhaps up to a negligible error). The second observation is that bare-OT cannot be a secure OT protocol. This is due to the known fact that there exists no unconditional construction for OT (this may be traced back to Chor and Kushilevitz [7] or even Ben-Or, Goldwasser and Wigderson [4]). Formally:

Claim 4.3 *There exists no (semi-honest) OT protocol in a world consisting solely of a PSPACE-complete oracle.*

We give a more precise interpretation of the above claim. An OT protocol is defined by the parties inputs s_0, s_1 and c , along with their respective random coins r_S and r_R . These parameters fully describe the transcript of the protocol. Denote by $view_S^{OT}$ (and $view_R^{OT}$) the view of the sender (receiver) in this protocol (that is, the party's input, randomness and the messages in the transcript). The exact formulation of Claim 4.3 says that for every execution of such an OT protocol, either the sender or the receiver can learn all of the other side's input.

Claim 4.4 *There exist polynomial time procedures A_S and A_R with access to the PSPACE-complete oracle such that for every implementation of OT, for every choice of s_0, s_1, c, r_S, r_R we have either that $A_S(view_S^{OT}) = c$ or $A_R(view_R^{OT}) = (s_0, s_1)$.*

The above two procedures constitute a break of the bare-OT. This is since at least one of them succeeds with probability at least $\frac{1}{2}$. Say, w.l.o.g. that A_S succeeds with probability $\frac{1}{2}$, then the sender can output the receiver's choice bit with probability at least $\frac{3}{4}$ (probability $\frac{1}{2}$ for success plus a coin flip otherwise) which is significantly more than he is allowed in a secure protocol. A proof of Claim 4.4 is brought in Appendix B.

The attack on the combiner To conclude the proof, we show that the attack A_S achieves the same success in World 1 as it does on the bare OT. The attack A_R does the same in World 2.

Every choice of oracles OT_A and OT_B is consistent with some randomness of the sender and receiver in the bare-OT. Thus, the view of the receiver in bare-OT may be simulated in World 1. To do this, the receiver runs the combiner as prescribed (recall that the bare-OT follows the same prescription), but whenever the oracle OT_A is called (by either side), the receiver calls the inverting oracle Inv_A and records the inputs and outputs to the oracle. Here it is crucial that the receiver will be aware of all the answers that the sender received for his queries to OT_A , a fact that is guaranteed by the transparent black-box structure of the combiner. Putting things together, for every execution of the combiner with inputs s_0, s_1 and c , there exists an execution of bare-OT with the same inputs, of which the receiver can produce $view_R^{OT}$. Thus whenever $A_R(view_R^{OT}) = (s_0, s_1)$, there is also a procedure A'_R such that $A'_R(view_R^{World1}) = (s_0, s_1)$. Respectively, in World 2, for the same execution of the combiner a procedure A'_S simulates A_S in the same corresponding execution of bare-OT.

Altogether, there exist procedures A'_R and A'_S , such that for every execution of the combiner, either A'_R breaks it in World 1 or A'_S breaks it in World 2. \square

4.2 (2,3)-Robust OT-Combiner

The results of the previous section indicate that (1,2)-Robust OT-combiners seem out of our reach at this point. We can however give a solution to the slightly more modest task of (2,3)-Robust

OT-combiner. This solution is a third party black-box combiner and relies on some often used techniques of Crépeau and Kilian [9] for amplifying the security in weak versions of OT protocols.

Claim 4.5 *There exists a (2,3)-robust OT-combiner scheme.*

Furthermore, the (2,3)-combiner is very efficient, making just 6 calls to the candidates. The efficiency is essential for the application in the next section.

Proof Sketch: We present the construction along with a high level sketch of why it is secure. For simplicity we will discuss OT on single bits, although everything can be generalized for strings in a straightforward manner.

Consider 3 candidates for oblivious transfer OT_A, OT_B, OT_C . We first use a construction that takes 2 OT candidates and always maintains the security of the receiver.

$R(OT_A, OT_B)(s_0, s_1; c)$ is defined as follows:

1. The sender chooses a random bit r
2. The receiver chooses random bits c_0, c_1 such that $c_0 \oplus c_1 = c$
3. The parties run $OT_A(r, r \oplus s_0 \oplus s_1; c_0)$ and $OT_B(r \oplus s_0, r \oplus s_1; c_1)$
4. The receiver outputs the XOR of his outputs in both executions.

We next present another construction that takes 3 candidates for OT and strongly protects the sender. Define $S(OT_A, OT_B, OT_C)(s_0, s_1; c)$ as follows:

1. The sender chooses random bits r_0^A, r_0^B, r_0^C and r_1^A, r_1^B, r_1^C subject to $r_0^A \oplus r_0^B \oplus r_0^C = s_0$ and $r_1^A \oplus r_1^B \oplus r_1^C = s_1$.
2. The parties run $OT_A(r_0^A, r_1^A; c)$, $OT_B(r_0^B, r_1^B; c)$ and $OT_C(r_0^C, r_1^C; c)$.
3. The receiver outputs the XOR of his outputs in the three candidates.

Finally, define $OT_{AB} = R(OT_A, OT_B)$, $OT_{AC} = R(OT_A, OT_C)$ and $OT_{BC} = R(OT_B, OT_C)$. The (2,3)-robust OT-combiner is defined as $S(OT_{AB}, OT_{AC}, OT_{BC})$.

It is simple to see that the above construction has the functionality of an OT protocol as long as the candidates have this functionality. The sketch of the proof of security follows:

Notice that the construction R has the following properties:

- If at least one of OT_A and OT_B is a secure OT protocol then the sender learns nothing about the receiver's bit c (this is because he cannot learn at least one of c_0 and c_1).
- If both OT_A and OT_B are secure OT protocols then $R(OT_A, OT_B)$ is a secure OT protocol (the receiver only learns two values that are random subject to the fact that their XOR equals s_c).

Also use the properties of the reduction S :

- If the sender cannot learn c in all 3 candidates then the value c remains secure.
- If at least one of the 3 candidates is a secure OT protocol then the receiver cannot learn both secrets (the receiver must learn all 6 values to learn both s_0 and s_1).

Since two of the three candidates are guaranteed to be secure OT protocols then the three intermediate constructions OT_{AB}, OT_{AC} and OT_{BC} must all contain at least one secure input and therefore all the 3 protect the receiver's bit b . Also at least one of the 3 is fully secure. After using the construction S , we have that both the receiver and sender are protected and the scheme is secure.

We note that the above description assumes the functionality of OT_A, OT_B and OT_C . Some additional work needs to be done in the case that the functionality is not guaranteed. Specifically, after testing for this property (as described in Section 3.1), we only need to handle the case that one of the candidates might error with probability at most $\frac{1}{n}$. The error can be reduced into a negligible error probability using an error correcting code (for example, use a code that allows to recover the secret if $\frac{2}{3}$ of the shares are received). \square

An alternative proof is to create an OT that is secure with probability $\frac{2}{3}$ simply by first randomly choosing one of the three candidates and then applying it. In [10] it was shown how such an OT can be amplified to one that is secure with all but a negligible probability. However the construction presented here is much more efficient, a fact that is later used in Section 5.1.

5 From (1,2)-Combiners to (1,n)-Combiners

(1,2)-robust combiners are essential for the existence of (1,n)-robust combiners. It is interesting to study under what conditions (1,2)-combiners suffice for the construction of (1,n)-combiners.

For some primitives, (1,k)-combiners can be reached as a simple extension of the construction of (1,2)-combiners (for instance, the KA-combiner presented in Section 3.3 extends easily). However, this is not clear for all combiners, and relies on the specific primitive at hand. We try to give more generic answers to the question posed above.

The natural construction takes the k candidates and organizes them as leaves of a binary tree, and applies the (1,2)-Robust \mathcal{P} -combiner scheme for every internal node (in a bottom up fashion). Now for every node that is a secure \mathcal{P} scheme, its ancestor must also be a secure \mathcal{P} scheme. The output of the whole tree must therefore also become a \mathcal{P} scheme since the root is an ancestor to all leaves. This construction suffices as long as its running time remains polynomial. But since the depth of this tree is logarithmic in k , and in order to allow the recursive application of the (1,2)-combiner, we must limit the running time of the (1,2)-combiner. We distinguish between general (polynomial time) combiners and very efficient ones. A combiner is said to be **very efficient** if its running time is bounded by a constant times the running time of its candidates. Examples of very efficient combiners are those for OWFs and pseudorandom generators. We have that:

Lemma 5.1 *For any \mathcal{P} and for all k , any very efficient (1,2)-Robust \mathcal{P} -combiner can be turned into a (1, k)-Robust \mathcal{P} -combiner.*

On the other hand, the tree construction is not efficient when the running time of the (1,2)-combiner is simply polynomial time. This is troubling since if a (non-BB) OT-combiner is eventually found, it is not very likely that it will be a very efficient one. Nevertheless, it will still suffice for constructing (1,n)-combiners for OT. We observe that given a very efficient (2,3)-combiner, one can construct (1,n)-combiners from any (not necessarily very efficient) (1,2)-combiner. This result along with the efficient (2,3)-combiner for OT (Section 4.2) allow us to focus our attention on constructing (1,2)-combiners for OT.

Theorem 5.2 *Any (1,2)-robust combiner for OT, can be used to construct a (1, k)-Robust combiner FOR OT scheme.*

The idea is to divide the k candidates into 3 groups (each of size $\frac{2}{3}k$), such that each candidate appears in at least two of them. A $(1, \frac{2}{3}k)$ -combiner is then recursively run on each of the groups and the 3 outcomes are then united using the $(2,3)$ -combiner. Formally:

Proof: The construction of the $(1,k)$ -combiner makes use of the $(2,3)$ -robust OT-combiner presented in Section 4.2. The crux being that the $(2,3)$ -combiner for OT is very efficient (in fact it makes just 6 calls to its candidates, though we simply use the multiplicative constant c). Divide the k candidates into three groups of size $\frac{2}{3}k$ such that each candidate appears in at least two of the groups. For instance, take the first two thirds as group 1, the second two thirds as group 2 and the first and last thirds as group 3. The construction recursively computes a $(1, \frac{2}{3}k)$ -combiner on each of these groups. The 3 outcomes of these combiners are given as input to the $(2,3)$ -combiner.

Since one candidate is guaranteed to be secure, then at least 2 of the combiners on the 3 groups implement secure OT protocols, and therefore the outcome of the $(2,3)$ -combiner also securely implements OT. Let $t(k)$ be the running time of the $(1,k)$ -combiner. The base stage of the recursion is a $(1,2)$ -combiner that takes a polynomial time (say $t(2) = n^d$ for constant d). The recursion gives us running time $t(k) = 3c \cdot t(\frac{2k}{3})$. Altogether this gives $t(k) = (3c)^{\log_{3/2} k} \cdot n^d$ which is polynomial.

Note that the $(2,3)$ -combiner is very efficient given that the candidates indeed implement the OT functionality (the receiver gets the bit of his choice). In the $(1,n)$ -combiner, we test the functionality of all candidates in advance, so that we only use candidates that error in the functionality with probability at most $\frac{1}{t(n)^2}$ (see Section 3.1 for a description of how candidates are tested). Since there are at most $t(n)$ calls to candidates, then by a union bound, the overall probability of error is at most $\frac{1}{t(n)}$. Using an error correction code, the error may then be reduced to a negligible one. \square

5.1 Universal Schemes for Primitives

Definition 5.3 (Universal Schemes) *A universal scheme \mathcal{U} for a cryptographic primitive \mathcal{P} is an explicit construction with the property that if the primitive \mathcal{P} exists, then \mathcal{U} is a secure implementation of \mathcal{P} .*

Levin [19] introduced such a scheme for OWFs. He showed an explicit function which is a OWF under the sole assumption that OWFs exist. In a sense, the meaning of such a universal scheme \mathcal{U} for \mathcal{P} is that any proof of existence for \mathcal{P} is guaranteed to be a constructive one, since, once \mathcal{P} is proved to exist then \mathcal{U} is an explicit implementation of \mathcal{P} .

The property that allowed Levin's universal-OWF schemes is the existence of robust combiners for OWFs. We try to formalize this connection for other primitives as well.

Lemma 5.4 *For any cryptographic primitive \mathcal{P} , if:*

1. *There is a known polynomial $p(\cdot)$ such that if there exists an implementation for \mathcal{P} then there also exists an implementation for \mathcal{P} with running time bounded by $p(n)$.*
2. *\mathcal{P} admits $(1, k)$ -robust combiners (for k a super-constant in the security parameter n).*
then we can provide a Universal- \mathcal{P} scheme.

Proof: The general idea of the universal scheme is to go over all possible implementation programs, hoping that at least one of them will fulfill our need. Then use the combiner to unite all of the programs into one that implements the primitive \mathcal{P} . More precisely, the universal scheme \mathcal{U} with security parameter 1^n goes over all of the Turing machines⁹ of description length at most $\log n$ and

⁹This step depends highly on the nature of the primitive \mathcal{P} . For example, if \mathcal{P} is an interactive protocol (like key agreement), then we enumerate interactive Turing machines.

unites them into one program using the $(1,n)$ -Robust \mathcal{P} -combiner with polynomial $p(n)$ as a time bound. So if scheme for \mathcal{P} exists then for some large enough n , this program is included in the n protocols that the \mathcal{U} includes, and by the robustness of the \mathcal{P} -combiner scheme we have that \mathcal{U} implements a scheme for \mathcal{P} . \square

Lemma 5.4 requires two properties of a primitive, the first asks that a time bound will be known on some implementation of \mathcal{P} . This property is very likely to be true about cryptographic primitives due to the following **padding argument**: If a primitive \mathcal{P} exists then there is an implementation of \mathcal{P} that runs in no more than n^2 steps. This observation follows from a padding argument. Suppose the existing implementation of \mathcal{P} runs in time $q(n)$ (where $q(\cdot)$ is a polynomial). We define a new primitive that is essentially the same: it receives an input 1^l but actually runs the protocol for the input 1^n such that n is the largest integer with $q(n) \leq l$ (observe that $n < l$ but they are polynomially related). Finding the parameter n and executing the program for \mathcal{P} can be done in time at most l^2 (simply by enumerating all possible n and halting once l is exceeded). So altogether the new \mathcal{P} scheme gets parameter 1^l and runs in time at most l^2 .

The padding argument works for most of the primitives we can think of. However care needs to be taken with primitives such as pseudorandom generators where padding of the input must also involve padding of the output. In the case of pseudorandom generators, for instance, it is easy to find a slightly modified argument that will work.

As corollaries of the above claims we get explicit constructions of many cryptographic primitives such as Universal-OWF and Universal-KA. Due to Theorem 5.2 We further get:

Corollary 5.5 *Any $(1,2)$ -robust combiner for OT, can be used to construct a universal-OT scheme.*

Note that in a computational setting, a $(1,2)$ -combiner for OT can simply ignore the candidate and run a universal-OT scheme (this is a non-black-box combiner). Thus, in this setting we can say that $(1,2)$ -combiners for OT exist if and only if universal schemes for OT exist.

6 Open Problems

The most intriguing question that rises from this paper is whether robust OT-combiners exist or not. Black box impossibility results have already been bypassed in the past, for instance, in the work of Barak [2]. We believe however, that solving this problem will require an altogether new technique. The techniques of [2, 3] do not seem to help here. The technique makes use of an explicit description of the *adversary's* program, which is of importance when dealing with malicious behavior. However our problem is interesting also in the semi-honest model, where such a program is constant. Other directions would be to try and reach a full impossibility result for general (rather than transparent) black-box combiners.

An interesting question about combiners regards the bit commitment primitive. For computationally hiding and statistically binding bit commitments we know how to build robust combiners, via the reduction to OWFs (given a OWF, commitments can be constructed using the reductions of Naor [20] and Hastad et al. [16]). This gives a transparent BB combiner that is highly inefficient. It would be interesting to find a more efficient combiner for commitments. For statistically hiding (computationally Binding) commitments the question of combiners is altogether open. A construction of such commitments from OWF would definitely suffice for combiners, however, at this point constructing statistically hiding commitments from OWFs remains an open question. It is worth noting that there is a very efficient $(2,3)$ -robust combiner for commitments (shown in [17]) that also works for statistically hiding commitments.

References

- [1] C.A. Asmuth and G.R. Blakely. An efficient algorithm for constructing a cryptosystem which is harder to break than two other cryptosystems. *Computers and Mathematics and Applications*, 7:447–450, 1981.
- [2] B. Barak. How to go beyond the black-box simulation barrier. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 106–115, 2001.
- [3] B. Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *43rd IEEE Symposium on Foundations of Computer Science*, pages 345–355, 2002.
- [4] M. BenOr, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.
- [5] M. Blum and S. Kannan. Designing programs that check their work. In *21st ACM Symposium on the Theory of Computing*, pages 86–97, 1989.
- [6] E. Brickell and K. McCurley. An interactive identification scheme based on discrete logarithms and factoring. *Journal of Cryptology*, 5(1):29–39, 1992.
- [7] B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM Journal on Disc. Math.*, 4(1):36–47, 1991. preliminary version in STOC 89.
- [8] C. Crépeau. Equivalence between two flavours of oblivious transfers. In *Advances in Cryptology - CRYPTO '87, Lecture Notes in Computer Science*, volume 293, pages 350–354. Springer-Verlag, 1987.
- [9] C. Crépeau and J. Kilian. Achieving oblivious transfer using weakened security assumptions. In *29th IEEE Symposium on Foundations of Computer Science*, pages 42–52, 1988.
- [10] I. Damgård, J. Kilian, and L. Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *Advances in Cryptology - Eurocrypt '99, Lecture Notes in Computer Science*, volume 1592, pages 56–73. Springer, 1999.
- [11] Y. Dodis and J. Katz. Chosen ciphertext security of multiple encryption. In *2nd Theory of Cryptography Conference - (TCC '05)*, volume 3378 of *Lecture Notes in Computer Science*, pages 188–209, 2005.
- [12] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
- [13] O. Goldreich. **Foundations of Cryptography**. Cambridge University Press, 2001.
- [14] O. Goldreich. **Foundations of Cryptography - Volume 2**. Cambridge University Press, 2004.
- [15] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.
- [16] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 29(4):1364–1396, 1999.
- [17] A. Herzberg. On tolerant cryptographic constructions. Electronic Colloquium on Computational Complexity (ECCC), TR02-135, 2002.
- [18] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM Symposium on the Theory of Computing*, pages 44–61, 1989.
- [19] L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7:357–363, 1987.
- [20] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
- [21] Nessie. Portfolio of recommended cryptographic primitives. www.cryptonessie.org, 2003.
- [22] M.O. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard, 1981.

- [23] O. Reingold, L. Trevisan, and S. Vadhan. Notions of reducibility between cryptographic primitives. In *The 1st Theory of Cryptography Conference – (TCC '04)*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, 2004.
- [24] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949.
- [25] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT ' 2000, Lecture Notes in Computer Science*, volume 1807, pages 275–288. Springer, 2000.

A Some Standard Definitions

For completeness we give some definitions of cryptographic primitives. The following definitions are standard and given for in a slightly informal fashion. For rigorous formal definitions the reader is referred to [14].

Definition A.1 (Key Agreement (KA)) *A Key Agreement (KA) is an efficient (polynomial time) protocol between two Interactive Turing Machines Alice and Bob. The input to the two machines is a security parameter n (in unary). At the end of a conversation between them both Alice and Bob output a k -bit string with the following two properties:*

Agreement : *Alice and Bob output the same string K (the Key).*

Secrecy : *No PPTM that only sees n and the messages passed between Alice and Bob, can distinguish with non-negligible probability between the key K and a uniformly distributed k -bit string.*

Oblivious transfer was originally introduced by Rabin [22] and has several equivalent flavors. The one used here is the $\binom{1}{2}$ -OT of [12]¹⁰ This is a game between Alice with two secrets s_0, s_1 and Bob with a choice bit bc . At the end of the protocol Bob should learn the secret s_c but not the other, while Alice should not learn the bit c . More formally this is given via the framework of secure computation:

Definition A.2 (Oblivious Transfer (OT)) *Oblivious Transfer is between Alice with secrets s_0, s_1 and Bob with a choice bit c has:*

Functionality : *If Alice and Bob follow the protocol then Bob outputs s_c (Alice has no output).*

Security :

- *For any strategy A^* of Alice there exists a simulator S_{A^*} (with essentially the same complexity) with access to 1^n , Alice's input s_0, s_1 and any a-priori information that produces an output that is computationally indistinguishable from the view of Alice running A^* in the protocol.*
- *For any strategy B^* of Bob there exists a simulator S_{B^*} (with essentially the same complexity) with access to 1^n , Bob's input c any a-priori information and to $s_{c'}$ for c' of his choice, that produces an output that is computationally indistinguishable from the view of Bob running B^* in the protocol.*

¹⁰The $\binom{1}{2}$ -OT was shown to be equivalent to Rabin's OT by Crépeau [8].

A protocol is said to be in the *Semi-Honest* model if the parties involved follow the protocol as prescribed, and perhaps later try to learn more information than what they were intended to. The main justification for this model stems from the work that Goldreich Micali and Wigderson [15] showed that given OWFs, any protocol that is secure in the semi honest model can be transformed into a protocol secure against maliciously behaving parties. OT in the semi-honest model is defined similarly to the above, but requires the existence of simulators only for the honest strategies of Alice and Bob.

B Proofs

This section contains proofs of Lemmas from Section 4.1.

Proof Sketch: (of Claim 4.2) For every strategy of the receiver, a simulator can simulate the message he received from the sender simply by choosing a random value. This is in a sense simulating the OT oracle without knowing what value was queried. Any efficient strategy will not be able to distinguish this message from the true one. The answer for the receiver's query to R is determined by $s_{c'}$, where the choice bit c' is determined by the original query to the OT oracle by the receiver's strategy. Recall that any illegal query is answered by \perp thus the simulator can also give a correct simulation for inconsistent queries. The simulation for the sender is likewise. \square

Proof: (of Claim 4.4) Suppose the sender and receiver run an execution of an OT protocol, that yields the transcript T . The inputs of the sender in this execution are s_0, s_1 and randomness r_R . The inputs of the receiver are c and the randomness r_R . Denote by $\mathcal{R} = \mathcal{R}(T, s_0, s_1, r_S)$ the set of all inputs for the receiver c' and r'_R that are consistent with the transcript T . By consistent we mean that the receiver using c', r'_R running the OT protocol a sender using s_0, s_1, r_S yield a transcript identical to T . Note that \mathcal{R} is not empty since it contains (c, r_R) . Call \mathcal{R} *monochromatic* if for all $(c', r'_R) \in \mathcal{R}$ we have that $c' = c$, otherwise, \mathcal{R} is called *non-monochromatic*. Consider the two possible cases:

- **\mathcal{R} is monochromatic:** The idea is that if the set \mathcal{R} is monochromatic and the sender can both verify this and calculate the value c then the sender learns the receiver's choice bit. The sender knows T, s_0, s_1, r_S so the procedure A_S can simply ask the PSPACE-complete oracle whether $\mathcal{R}(T, s_0, s_1, r_S)$ is monochromatic and what value c is it consistent with.
- **\mathcal{R} is not monochromatic:** In this the receiver can learn both secrets. One secret, s_c is automatically given by the functionality of OT-combiner. If the receiver could find a pair $(c', r'_R) \in \mathcal{R}$ with $c' \neq c$ then by simulating the execution of the OT protocol with inputs (c', r'_R) he learns the other secret as well.

Note that the receiver does not know s_0, s_1, r_S so the idea from the previous item cannot be repeated. However, since the sender does not get any information from the receiver other than the messages in T and since the messages generated by using r'_R and c' are consistent with T , they must also be consistent with s_0, s_1, r_S . To check consistency with T , it therefore suffices for the receiver to see T and the procedure A_R can use the PSPACE oracle to find such c', r'_R and learn both secrets.

Altogether, each execution of the OT protocol is broken by either A_R or A_S . \square