The Simplest Protocol for Oblivious Transfer

Preliminary Report in MTAT.07.022 Research Seminar in Cryptography, Fall 2015

Author: Sander Siim
Supervisor: Pille Pullonen

December 15, 2015

Abstract

This report discusses a fundamental primitive protocol in cryptography called oblivious transfer. It is a core protocol used in many techniques for secure computation, and as such, requires thorough research for achieving better security guarantees with the best possible efficiency. In this report, we present a general discussion about the security of cryptographic protocols and concepts used in security proofs and relate this to the context of oblivious transfer. We then present and discuss the properties of a few well-known oblivious transfer protocols in the literature, focusing on a recent protocol by Chou and Orlandi in 2015.

Contents

1 Introduction

In this report, we study protocols for an important cryptographic task called *oblivious transfer*. Although a conceptually simple primitive, oblivious transfer (OT) is one of the most fundamental protocols in cryptography. A famous result of Kilian from 1988 [22] shows that having a *black box* that implements oblivious transfer is enough to construct a protocol for *any* secure computation task.

In OT, two parties called the *sender* $\mathcal{S}$ and *receiver* $\mathcal{R}$ aim to do the following. $\mathcal{S}$ has two bit-strings M_0, M_1 and $\mathcal{R}$ has a choice bit $c \in \{0,1\}$. The goal of OT is to send the message M_c to $\mathcal{R}$ (according to $\mathcal{R}$'s choice), such that the sender $\mathcal{S}$ does not learn which of the messages was chosen (privacy of $\mathcal{R}$'s input) and also, that the receiver $\mathcal{R}$ learns nothing about the other message M_{1-c} (partial privacy of $\mathcal{S}$'s input). This particular variant of OT is called 1-out-of-2 OT (denoted more compactly $\binom{2}{1}$-OT), since the receiver chooses one of two messages. There also exist many other variants and generalizations of this basic definition (Section 2.2 gives a more complete overview).

Although $\binom{2}{1}$-OT is enough to evaluate any function (even with malicious adversaries), the general construction is, of course, not efficient enough for practice[1]. However, OT is still used as an irreplaceable subroutine in many *efficient* secure computation protocols. For example, OT plays a integral role in the well-known Yao's garbled circuits protocol for efficient two-party computation [31, 26]. As such, OT is a well-studied protocol in the literature, since having a more efficient OT protocol also makes other protocols relying on it more practical. Similarly, a successful OT protocol should guarantee the highest level of security as possible, since a chain is only as strong as its weakest link.

Report outline. We first start with defining some notation and discussing different variants of OT in Section 2. We then provide a summary on defining and proving security of cryptographic protocols in general in Section 3. There we describe and compare different adversarial models and methodologies for defining security. This serves as a basis for comparing the difference in security guarantees that well-known OT protocols in the literature provide. We present the recent and very efficient Chou-Orlandi15 OT protocol [14] in Section 4 and provide a brief comparison with other well-known OT protocols from the literature. Finally, a concluding overview and table comparing efficiency of different OT protocols appears in Section 6.

[1] The seminal paper by Goldreich-Micali-Wigderson [19] proves a completeness theorem for secure multi-party computation in the case where a majority of participants is honest. The protocol evaluates a binary circuit for given functionality, such that XOR gates use local computations and each AND gate uses a $\binom{4}{1}$-OT. The extension to the malicious case is done by adding zero-knowledge proofs to prove each party behaved correctly. The semi-honest GMW protocol is in some cases competitive in terms of performance, whereas the malicious case definitely is not.

2 Preliminaries

2.1 Notation

Throughout this report we consider finite cyclic additive groups. We denote as $(\mathbb{G}, G, p, +)$ an additive group $\mathbb{G}$ of order p with generator $G \in \mathbb{G}$. Thus, $\mathbb{G} = \{xG : x \in \mathbb{Z}_p\}$. Given any set S we denote by $x \leftarrow S$ sampling the element x uniformly randomly from S. We use the shorthand $[n] = \{1, \ldots, n\}$ and use $\{0,1\}^n$ to denote the set of bit-strings with length n.

We also consider symmetric encryption schemes (E, D) where E is the encryption function $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and D the decryption function $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$. Here$\mathcal{M}$ is the message space, $\mathcal{K}$ the key space and $\mathcal{C}$ the ciphertext space. Each is a set of bit-strings with fixed length. The basic correctness requirement is that $D(k, E(k, m)) = m$ for all $m \in \mathcal{M}$ and $k \in \mathcal{K}$.

2.2 Oblivious transfer

The notion of oblivious transfer was first proposed by Rabin in 1981 [29] and then formalized with the current standard definition of 1-out-of-2 by [15].

Later authors have generalized this to $\binom{n}{1}$-OT (the receiver chooses one message out of n) and $\binom{n}{k}$-OT (the receiver chooses a subset of size k from among n messages). An even more general variant is called *reactive* $\binom{n}{k}$-OT, where the receiver chooses a total of k messages, but each next index is chosen after receiving the previous message, allowing the previous messages to influence the decision of the receiver. In this work, we focus mostly on $\binom{n}{1}$-OT, as this variant has many applications in general secure multi-party computation techniques.

In many such applications (e.g Yao's garbled circuits [31]), many separate invocations of $\binom{n}{1}$-OT are required. In such cases, a technique called OT-extension is typically used [1, 2, 21]. OT-extension allows to extend a small number of *base* OT invocations (done using standard OT protocols discussed in this report) to a much larger number of OT invocations, such that the extra OT-s are then much cheaper to perform (using only symmetric operations as opposed to asymmetric operations).

Some OT-extension techniques (especially the passively secure [1]), allow the base OT-s to be independent of the inputs used for the more efficient extended OT-s, which makes it possible to perform these as a pre-computation. The base OT used then is a variant called *random-OT*, where the sender does not choose the messages himself. Rather, as an output of the protocol, he receives n random messages (in the case of $\binom{n}{1}$-OT), one of which is transferred to the receiver in an oblivious way.

3 Security of cryptographic protocols

In this section, we try to give an overview of the main methodologies used today to prove security of cryptographic protocols. This serves as a basis for understanding the concrete security guarantees provided by different OT protocols discussed in this report and the practical implications of these formal, abstract definitions. This exposition mainly follows that of [20]. We do not claim to cover all aspects of protocol security in this discussion, but try to explain (in a self-contained manner, using little formalism) the key aspects that come up in analyzing security of protocols, and in particular oblivious transfer.

3.1 Properties of a secure protocol

We start first by discussing the conceptual properties that a secure protocol can and should have. In other words, what are the characteristics that make a protocol *secure*. We later see how to formulate security definitions that guarantee a secure protocol indeed has these properties in many possible settings. To illustrate these properties, we use the setting of a secure auction as an example.

- *Privacy* — The most common property of a secure protocol is maintaining the privacy of the parties' inputs. Namely, each party should only learn its intended output from the protocol and nothing else. In the secure auction example, a party's bid should remain private from all other bidding parties.

- *Correctness* — Each party should have a guarantee that the output it receives from the protocol is indeed correct (according to the functionality of the protocol). We want a guarantee that only the bidder with the highest bid can win the auction.

- *Independence of inputs* — It should not be possible for a malicious party to choose its inputs to the protocol in any relation to honest parties' inputs. This property is related to *malleability* attacks. Consider the auction case. A party might choose his bid particularly in such a way that is definitely higher than all other bids made so far. Note that this might be possible even if the privacy property is maintained. For example, it might be possible to modify the ciphertext of bid x such that it decrypts to $x + 1$, without learning x in the process.

- *Guaranteed output delivery* — Honest parties should be guaranteed to receive the output of the protocol after the protocol execution has begun. This relates to an adversary trying to disrupt the computation with some sort of denial-of-service attack. For example, as a contrived example, a poorly designed auction protocol could make the entire protocol halt whenever an ill-formed bid is sent by some party.

- *Fairness* — We would like to guarantee that corrupted parties receive their protocol outputs if and only if the honest parties also receive their outputs. For example, say a corrupted bidder learns the winner of the auction before others, and decides to halt the entire protocol if he is not happy with the outcome. A fair protocol would not allow this.

Note that we certainly do not require all of these properties for every possible protocol and application. In particular, guaranteed output delivery and fairness are very difficult or even impossible to attain in many cases. This highly depends on the application and context. In most cases we at least require privacy and correctness. We will later discuss (in Section 3.8) how these properties relate to oblivious transfer.

3.2 Defining the adversary

All of the discussed characteristics of a secure protocols might be either trivial, very difficult or even impossible to achieve depending on how powerful adversaries we want to guarantee security against. Specifying the allowed behaviour of an adversary is the key element that defines the level of security that a protocol achieves (if it is proven secure with respect to such adversaries).

Generally in cryptography, the adversary concept is used to model real-world attackers that try to break the security of the protocol in some way. Basically, the adversary is modelled as a computer program that performs some (probabilistic) algorithm (formally, interactive Turing machines or similar computational models are used) and somehow interacts with the parties involved in a protocol[2].

In a security analysis of a protocol involving many parties, we allow the adversary to *corrupt* some subset of these parties, which models either a performed attack by a real-world attacker or that the party itself is trying to attack the protocol. As such, we often talk about security *against a corrupted party*, by which we mean security in the event that this party is corrupted by an adversary. Also, other interactions besides corruption may be specified, and in particular, the adversary is usually allowed to see all network communication between parties, even honest ones (and in some cases, also be able to malform the messages without corrupting parties).

Static vs adaptive First we can consider at what time during the protocol execution is the adversary allowed to corrupt the parties. *Static* corruption assumes that the adversary chooses which parties to corrupt before the protocol starts and these parties remain corrupted throughout the execution.

Adaptive corruption is a slightly more realistic model, where adversaries are allowed to corrupt parties at any time during the protocol execution based on what they have seen so far. It is then assumed that the parties stay corrupted until the end of the protocol[3].

[2] For showing concrete security properties of cryptographic primitives (as opposed to protocols, with interaction between parties), the adversary is also usually put into the context of a specific "mental game", which is presented as a kind of protocol between the adversary and a *challenger*. This "protocol" is referred to as a *security game*, where the adversaries objective is to produce output that is different from some specified distribution, given the concrete interaction between adversary and challenger. In this context, the adversary's interaction is uniquely specified, but local computations are arbitrary, whereas in the context of protocols, the adversary can also interact with the parties in an arbitrary way.

[3] One can also consider *mobile* corruption where parties can again start behaving honestly at some

Semi-honest vs malicious One of the most common distinctions is whether the adversary behaves *semi-honestly* or is allowed arbitrary *malicious* behaviour.

A semi-honest adversary sees the internal state and all messages sent to and from the corrupted party during the protocol execution and can try to extract some private information from this view. However, he is not allowed to influence the corrupted party's execution of the protocol — the corrupted party still exactly follows the protocol description and performs all required computations correctly.

In the case of malicious corruption, the adversary completely takes control of the corrupted party. In addition to seeing its internal state, the adversary can now choose to send completely different messages than are defined in the protocol description or not send messages at all.

In practice, the semi-honest adversary models an attacker who gains access to a server executing the protocol and can monitor its internal memory during the protocol execution. The attacker corresponding to a malicious adversary potentially also influences the code that executes the protocol. Note that insider attacks are always possible, where the adversary is in fact the same party that performs the protocol.

Concerning the security properties stated before, protection against semi-honest adversaries essentially only deals with maintaining privacy. All other properties are naturally obtained by trivially constructing the protocol as such (e.g the protocol description involves all parties receiving their outputs). In the malicious setting, achieving all the different properties is significantly more challenging.

Computationally bounded vs unbounded. We can also consider some bounds on the computational capability of the adversary. When we refer to security in the *computational model*, we (usually) consider only adversaries whose running-time is asymptotically at most polynomial in the security parameter. Security proved in this model is called computational security, and the standard method to guarantee it, is to reduce any successful attack to that of solving a well-studied computational problem, that is believed to be hard (given a fixed amount of computational resources). The most common example is that of solving the discrete logarithm in some finite groups.

However, we can also choose to consider adversaries that have unlimited computational power. In this model, we prove *information-theoretic security*, for which the probability of adversary's success does not depend on its available computational power. This gives a stronger guarantee that no adversary is able to compromise security and the guarantee will also hold in the future as computational power becomes cheaper.

Network channels and their security Another important aspect is to consider how much control the adversary has over the communication links between the protocol parties. Quite often, the assumption of *authenticated* channels is necessary. This means that the parties know that they are communicating with the intended party and that

time after corruption.

the messages are not changed along the way, however an adversary is allowed to see all communication on this channel.

In addition, the network model is considered as either *synchronous* or *asynchronous*. In a synchronous network, it is assumed that a global clock exists, such that parties expect protocol messages to arrive after a certain amount of cycles. In the asynchronous model, the adversary can delay, reorder or replay messages arbitrarily. Note that this is also allowed for semi-honest adversaries. A synchronous model is a stronger assumption that assumes the messages must arrive in some fixed amount of time, otherwise the party will abort, which reduces the adversary's options.

Finally, note that all of these different criteria can be considered in arbitrary combinations. From this list, the model with the least restrictions on adversarial behaviour is *malicious* adversaries with *adaptive* corruption, that are *computationally unbounded* and have control over the protocol parties' *asynchronous* communication.

3.3 Basic simulation-based security

Now we turn to the task of actually formulating and proving security, given the conceptual idea of what we want to achieve with a secure protocol and under which assumptions should it remain secure. The task is not an easy one, as we allow all sorts of unpredictable behaviour from the adversary.

A straightforward way of approaching this problem is to somehow try to define the worst attack an adversary can do, and then show that this attack is not feasible. Although this approach might work for cryptographic primitives with very specific security goals, it is not always possible in the case of protocols. Trying to enumerate all possible attack scenarios is error-prone and does not give confidence that indeed no imaginable attack can compromise the security of a protocol.

We take a first step towards formulating security in a very general way that considers every possible adversary, by using *simulation-based security*. Let us consider for the moment only static and semi-honest adversaries and some two-party protocol with parties $\mathcal{P}_1$ and $\mathcal{P}_2$ where each gives some input to the protocol and expects an output. For semi-honest adversaries the security goal is to preserve the privacy of the parties' inputs — a corrupted party $\mathcal{P}_1$ should not be able to learn the input of $\mathcal{P}_2$ and vice-versa.

For clarity, let us fix some protocol π with parties $\mathcal{P}_1$, $\mathcal{P}_2$ that computes a function $f(x_1, x_2) = (y_1, y_2)$, where x_i is the input from party $\mathcal{P}_i$ and y_i is similarly P_i's output[4]. The intuition behind simulation-based security in this setting is that to preserve privacy of P_i's input, the messages that P_i sends to other parties should be independent of its input. In that case, it should be possible to *simulate* these messages reasonably accurately (such that no adversary can tell the difference), without having access to the input of P_i.

[4] Notice that we can formulate $\binom{2}{1}$-OT as computing the function $f((M_0, M_1), c) = (\lambda, M_c)$, where λ is an empty string.

We therefore construct a mind game, where the corrupted party $\mathcal{P}_i^*$, instead of interacting with $\mathcal{P}_{1-i}$, interacts with a program $\mathcal{S}$ called the *simulator* that we specifically construct. We then ask whether we can construct $\mathcal{S}$ in such a way that, given access to $\mathcal{P}_i^*$'s input and output only, it produces a view for $\mathcal{P}_i^*$, that together with the joint output of both parties, is indistinguishable from an actual protocol execution. If we can do this (in a general way for every possible adversary in our model), then we can conclude that everything $\mathcal{P}_i^*$ sees in the protocol execution, he could actually simulate for himself, without needing to interact with the honest party, and therefore the information he sees cannot depend on the honest party's input. We stress that the distributions of the joint outputs must also coincide, otherwise we lose the correctness guarantee. If we consider only the view of the corrupted party, we get a security definition that is concerned with privacy of inputs only (which can also be sometimes useful).

3.4 The real vs ideal world paradigm

Before, we considered only semi-honest adversaries, who are guaranteed to use their real inputs in the protocol[5]. However, a malicious adversary, among other things, can choose any input to the protocol. As such, the above reasoning fails for malicious adversaries, since the simulator cannot reliably use the corrupted party's input as a basis of the simulation.

We thus turn to a slightly different methodology of simulation-based security, namely the *real vs ideal world paradigm.* With this approach, we first formalise an *ideal functionality* $\mathcal{F}$ of our protocol in the form of an incorruptible trusted third party. In the ideal world, the parties then outsource the protocol execution entirely to $\mathcal{F}$, meaning that they simply give all inputs to $\mathcal{F}$, who then reliably gives back output to the parties[6].

We then also formalize the real-world execution of the protocol under consideration, where parties exchange messages and the adversary can corrupt and listen on communication channels. The simulation idea is then similar, we want to show that the real world is close to (indistinguishable from) the ideal world. Since we design the ideal world model specifically to not include any unwanted adversarial behaviour, then if the two execution models are indistinguishable, we can conclude that the adversary also has no chance of success in the real.

Similarly to the semi-honest case with simulating parties' views, here the goal is to

[5] In general, we make no assumptions about the distribution of inputs. However, in the case of semi-honest adversaries, once the input is fixed it is used throughout the protocol. In particular, a simulator has the guarantee that the corrupted party's input is used in the real-world execution of the protocol, allowing the view to be simulated.

[6] In fact, we cannot always model fairness and guaranteed output delivery into the ideal model, since it might be impossible to achieve this with actual protocols. For example tossing a completely fair unbiased coin with two parties is impossible. The corrupted party, having received its output, may simply abort before sending the final message that would give the honest party its output. Even when possible, fairness is usually a very costly property to achieve without an honest majority of participants.

simulate the view of the adversary (corrupted party). The simulator should "fool" the adversary into thinking that he is participating in the real world protocol execution, while actually being in the ideal world setting where computations are done by $\mathcal{F}$, instead of the parties. For security, the joint output of honest and corrupted parties must be indistinguishable in both experiments for any set of inputs. Intuitively, if honest party's outputs would not be indistinguishable, then the adversary can succeed in making the protocol return wrong results. If the adversary's output is not indistinguishable, then he potentially gets more information in the real world than we allow for in the ideal world. As before, the simulator does not have access to the honest parties' inputs and acts as a interface to the adversary, who assumes he is communicating with the real parties.

For this definition to make sense, the ideal functionality must include all the allowed adversarial behaviour. Namely, the ideal functionality *defines exactly* the security guarantees that the protocol gives us. When a protocol is proven secure using the ideal vs real world paradigm in the way described and with respect to all parties that could be corrupted, we say the protocol then achieves *fully-simulatable* security.

3.5 Universal composability

We now consider the problem of *composing* protocols in a secure manner. All the discussion and security notions so far have implicitly used a crucial assumption — namely that the protocol is strictly executed in a *stand-alone* setting. However, in the real world, instances of protocols are quite often executed concurrently or sequentially composed with instances of the same protocol or even instances of other arbitrary protocols. A secure composition of protocols that are stand-alone secure is crucial for modular protocol design, otherwise the complexity of the protocol must be analyzed as a whole, which can be daunting for larger protocols.

Unfortunately, stand-alone security for a protocol does not imply security under *general concurrent composition* directly. For example, given two concurrently running copies of the same two-party protocol, an adversary might corrupt one party in the first copy of the protocol and the other party in the second, and then execute some man-in-the-middle attack. Stand-alone security gives no guarantees in case of such situations. Naturally, we would like a security definition that also excludes any sort of attack that occurs in concurrent environments, which realistically models protocols running over the public Internet for example.

Defining security with respect to general concurrent composition is an even more complicated task than in the stand-alone model, but this also has been solved in a rigorous formal manner. This is the level of security that a *universally composable* protocol provides. The universal composability (UC) framework proposed by Canetti in 2001 is one of the cornerstones of modern cryptography [8]. The main contribution of Canetti is presenting a methodology of modelling ideal-world and real-world executions of a protocol that takes into account arbitrary concurrent protocols running in the environment, but abstracting all the inherent complexity of concurrency away from the security proof. Namely, Canetti gives a security definition that *UC-secure* protocols must meet, and proves that a UC-secure protocol remains UC-secure when arbitrarily composed with any other protocols. Roughly said, UC-security against malicious adaptive adversaries is the strongest notion of security for cryptographic protocols that one would hope to achieve (assuming that the ideal functionality defined is a reasonable one).

However, as is to be expected, achieving UC-security is a strictly harder task than stand-alone security. In fact, there are many impossibility results regarding UC-security against malicious adversaries in the *plain model*[7]. For example, UC-secure two-party computation and commitment schemes are shown to be impossible against malicious adversaries in the plain model [20, 9]. However, some added trust assumption or idealization of the real world, (see common reference string and random oracle model below) may allow UC-secure protocols to be constructed.

Similarly to fully-simulatable security in the stand-alone model, UC-security uses the ideal and real world paradigm. The ideal functionality should capture all adversarial behaviour that the protocol needs to tolerate. In the context of this report, we will skip many of the technical details regarding the UC framework, since they are quite numerous and elaborate. We refer the interested reader to [8].

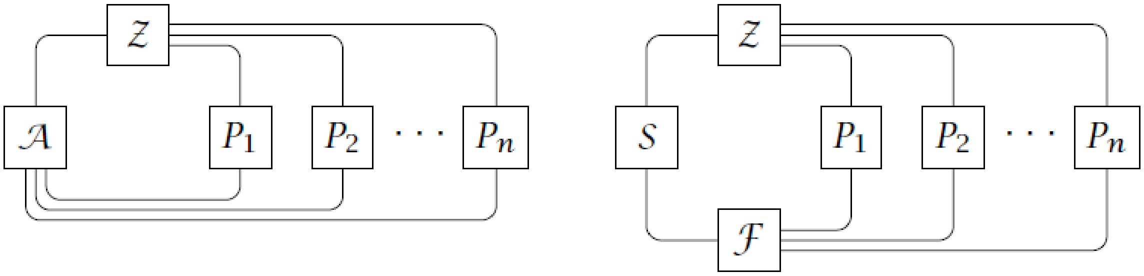
The real world execution model is broadly defined as follows (see Fig. 1). We introduce a new entity into our execution model, namely the *environment* $\mathcal{Z}$. The environment models everything happening in the network as the protocol is being executed. Specifically, the environment chooses the inputs that the parties use in the protocol, and reads their outputs after execution is complete. Also, there is free information flow between the environment and the adversary $\mathcal{A}$ at any time. Roughly this means that the adversary has access to all side-information from the concurrent environment and the inputs to the protocol may depend on results of some other protocols.

In the real world, the adversary can (in principle) corrupt all network communication between parties. Thus, in the standard UC model, the parties exchange all their messages using $\mathcal{A}$ as a kind of proxy. A semi-honest $\mathcal{A}$ will send these messages forward unchanged, a malicious adversary may not (if there are no authenticated channels).

In the ideal world however, we put the ideal functionality $\mathcal{F}$ into the model and now the parties directly communicate with $\mathcal{F}$. Also, the simulator $\mathcal{S}$ now replaces $\mathcal{A}$ and has a communication line with $\mathcal{F}$ to model any admissible leakage. The rough security requirement is that for any $\mathcal{A}$ in the real world, there exists a $\mathcal{S}$ in the ideal world, such that the view of the *environment* is indistinguishable in both experiments.

Note that when we were discussing stand-alone security, the goal was to simulate

[7] The plain model refers to assuming only the standard model with standard computational hardness assumptions and excluding, among other things the random oracle model and any additional trusted setup assumption such as the common reference string model. Authenticated channels, however, are assumed, since they are implied by secure public key encryption schemes (although this does implicitly imply that public keys can be securely exchanged, which means trusted setup assumptions are not avoided completely). When nothing is assumed about the network channels, one usually refers to the *bare model*.



(a) Real world.

(b) Ideal world.

Figure 1: UC framework real and ideal world protocol execution (taken from [18])

honest parties' messages to an adversary. In UC, the simulator now *replaces* the adversary, and instead, we are simulating the adversarial behaviour to the environment. The environment therefore acts as a interactive distinguisher between the two experiments. The intuition for security is that if we can simulate any $\mathcal{A}$ in the ideal world, then every attack that $\mathcal{A}$ could in the real world can also be done in the ideal world (which is ideal by definition). Note that the $\mathcal{S}$ is allowed to have non-rewinding black-box access to $\mathcal{A}$ for the simulation.

3.6 Random oracle model

3.6.1 Definition

Dating back to Fiat and Shamir [16] and later advocated by Bellare and Rogaway [5], the random oracle model (ROM) is a security model for cryptographic protocols, that allows to construct rigorous proofs in cases where very strong properties are required from a hash function. Namely, it allows the use of a *random oracle* (RO), which is a public function H that is made available to all the parties (including the adversary). The assumption is, that H behaves as a completely random function, meaning, its output on every new input is a uniformly random element from the range of the function. Another equivalent definition would be that H is sampled uniformly randomly at the beginning of the protocol from the set of all possible functions with given domain and range.

In a security proof using the ROM, the random oracle is modelled as a separate entity from other parties in the protocol and must be explicitly queried with a given input x to receive the output $H(x)$. The oracle is public and typically all parties are allowed access to it, most notably, the adversary. In the stand-alone model, this construction allows simulators in security proofs to also simulate the answers of the random oracle to the adversary. This means that the simulator can actually choose the answers to random oracle queries that the adversary makes. This is called *programmability* of the random oracle and is one of the main reasons, why proofs in the ROM are much easier than in the standard model.

3.6.2 Random oracle in practice

It is easy to show that computing with random functions is either computationally or storage-wise infeasible in practice (for the necessary domain and range sizes). Thus, when in a real-world instantiation of the protocol, the random oracle is replaced by an actual hash function used in practice (such as SHA-256), the hash function is not guaranteed to act at all similarly to a random function. This however, in a rigorously formal sense, means that the security proof is meaningless regarding the real-world instantiation of the protocol, since the premises made in the proof do not hold.

However, there are also advantages to using the ROM. First, from the cryptographer's perspective, the security proofs that use a random oracle are typically much easier to construct (and also to validate). Second, the resulting protocols are more efficient in terms of communication and/or computation complexity than protocols providing the same security guarantees in the *standard model* that rely solely on some well-known hardness assumptions, such as the discrete logarithm or Diffie-Hellman problems (if this is not the case, a protocol based on the standard model is naturally preferable). Third, a rigorous proof in the random oracle model still rules out all other attacks that do not exploit some specific weakness of the used hash function and might be considered better than "no proof at all". However, a ROM security proof should be considered to provide a sort of heuristical security.

So far, in practice no protocol that has been proved secure in the ROM has been explicitly broken, with the exception of some examples specifically designed to show that a proof in the random oracle model does not logically imply security when any real-world hash function is used [10]. However, these few examples are arguably contrived and "unnatural" [23], and the ROM is today still used quite often in constructing new efficient protocols (for example, the Chou-Orlandi OT protocol [14] discussed here). On the other hand, many researchers specifically avoid the ROM and aim to make protocols in the standard model more efficient, to provide an alternative with "more provable" security.

Even if one is willing to accept the random oracle model as a plausible security proof model, care should be taken to use the ROM in a reasonable manner. For example, some schemes may use a random oracle that provides outputs that are longer than any practical real-world hash function can provide. Instantiating such random oracles is not a straight-forward task and can lead to vulnerabilities when not done correctly [24].

3.6.3 Using ROM in the UC framework

Combining ROM and the UC framework together has an important requirement. Namely, for the composition theorem to hold, it is required that each protocol instance uses a *separate* random oracle that is local to the concrete instance. Obviously, this is not what is typically done in ROM, where we simply replace the random oracle by a *global* hash function when instantiating the protocol. In this case, there is no guarantees for composability, since other instances of the same protocol or entirely different protocols could be using the same hash function.

Canetti has recently proposed also a UC model, that allows access to a global random oracle and still retains composability [11]. However, the downside is that this global random oracle cannot be programmed by the simulator, which makes it significantly less powerful for security proofs.

3.7 Common reference string model

For completeness, we mention another common model that is used to get past some impossibility results for UC-security with malicious adversaries. Namely, that of the *common reference string model* (CRS). The model assumes a trusted setup phase that occurs before the protocol execution. The basic idea is that each participant in the protocol is given (in a secure, idealized manner) access to a common string from some specified distribution $\mathcal{D}$. When $\mathcal{D}$ is the uniform distribution, the model is referred to as the *common random string* model.

This model is usually adopted specifically to get around the impossibility for UC-secure protocols with malicious adversaries in the plain model [9, 12]. As such, it is very questionable, that given the strong adversarial model, it is possible in practice to implement the common string in any "reasonably trusted" way. Usually, the common string can only be reused to some extent and needs to be refreshed after a while, which makes the assumption even more difficult in practice.

Similarly to the random oracle model, there is the same locality issue with CRS when considering UC-security. The same string should not be used in two different protocol executions, which is highly wasteful of the common string. However, if the ideal functionality is defined in a specific way, there is a possibility to use the CRS globally for some number of protocol instances [13].

3.8 Flavours of OT security

Given the previous discussion about security of any general cryptographic protocol, we now look more specifically at oblivious transfer protocols and their security. For ease of exposition, we first define a completely abstract $\binom{2}{1}$-OT protocol. For the purely conceptual view, we assume here that the protocol has only two communication rounds (such OT protocols do in fact exist at least for semi-honest adversaries). The abstract protocol is presented below as Protocol 1.

Sender		**Receiver**
Input: $(M_0, M_1) \in \mathcal{M}^n$		Input: $c \in \{0, 1\}$
Output: none		Output: M_c
		$a \leftarrow \alpha(c)$
	$\xleftarrow{\quad a \quad}$	
$b \leftarrow \beta(a, M_0, M_1)$		
	$\xrightarrow{\quad b \quad}$	
		$M_c \leftarrow \gamma(b, c)$

Protocol 1: Abstract two-message $\binom{2}{1}$-OT protocol in plain model

An immediate thing to notice in the case of OT, is the asymmetry of outputs. Only the receiver gets a meaningful output from the protocol. Therefore, OT is a protocol where fairness is inherently guaranteed (if the receiver is corrupted, there are no other honest parties that should receive an output). We now consider what different types of security we can achieve with an OT protocol, in particular, to understand the difference of more stronger guarantees compared to weaker notions of security.

First and foremost, we should be interested in the privacy of both parties' inputs in the case that the other is corrupted. Note that there is no need to analyze the case where an adversary *does not* corrupt any party, but only taps in to the network communication, since corrupting either of the parties already includes all the information he would see.

Semi-honest $\mathcal{A}$, privacy and correctness. If we consider only semi-honest adversaries and for example, a corrupted receiver, we will achieve privacy simply by ensuring that the message b is independent of (M_0, M_1). The proof would construct a simulator $\mathcal{S}_1$, that based on the receiver's input, can simulate b with the same distribution as in the real protocol, such that the output of the receiver is always M_c. In the case of computational security, the view and output together would be computationally indistinguishable (not perfectly the same distribution).

Malicious $\mathcal{A}$, privacy. For malicious adversaries, we can consider privacy alone as the security goal (and disregard correctness). Then, we would have to give an ideal/real world proof showing that the adversary's output in both settings is indistinguishable, but disregarding the *joint output distribution* of adversary and honest parties.

Malicious $\mathcal{A}$, privacy and correctness. Continuing the last example, to also guarantee correctness, we would consider same experiment, but now we do need the joint output to be indistinguishable, since the honest parties need to get the *correct* output. However, notice that since the sender has no output, the argument for showing correctness w.r.t corrupted receiver is easier. Namely, there is nothing to show, since privacy already implies the corrupted receiver's output is simulatable.

One-sided simulatability. Usually we consider security equally against a corrupted sender and corrupted receiver. However, in some cases having better security against one of the corrupted parties may be useful. As we have seen, showing simulatability for a corrupted receiver in the case of OT is much easier, since we do not have to worry about the correctness of the sender's output, since he has no output. Therefore, an OT protocol that guarantees privacy against malicious adversaries is *one-sided simulatable*, i.e simulatable against a corrupted receiver, but only guaranteeing privacy against a corrupted sender.

Full simulatability. For completeness, we refer to *full simulatability* or say the protocol is *fully-simulatable* if it simulatable against both corrupted parties.

UC security. First, we remember the fundamental result that UC-secure OT against malicious adversaries in the plain model is impossible [8]. Thus, we would need to assume some sort of trusted setup[8]. Following that, a simulator has to be constructed which can (very informally) simulate in the ideal world everything the adversary does in the real world, such that the environment cannot distinguish the two settings.

4 Chou-Orlandi15 OT protocol

We now describe the recent OT protocol proposed by Chou and Orlandi this year [14]. The main idea is inspired by the well-known Diffie-Hellman key exchange protocol from 1976. The authors claim to achieve UC-security against adaptive malicious adversaries in an asynchronous network, assuming authenticated channels — a very strong level of security. However, the proof is done in the (standard, programmable) random oracle model, which immediately introduces the issue of the oracle's locality in the UC context, which we discuss in the analysis (see Section 4.3.2).

The authors also provide a highly-optimized reference implementation, which leverages latest Intel architecture vectorized instructions. The low-level engineering work is very interesting, since the details discussed are usually not considered in theoretical cryptography publications, although in practice, they play a huge role in both performance and security (e.g constant-time implementation to negate timing-based side-channel attacks). However, the code as presented is unfortunately not very portable, since it contains a lot of inline assembler.

[8] For instance, the global random oracle model from [11] or CRS in UC with joint state model [13].

4.1 Robust and non-committing encryption

We first define some properties of symmetric encryption schemes that the symmetric encryption scheme used as a building-block in the Chou-Orlandi protocol needs to satisfy (the definitions are the same as in [14]).

Definition 1 (Non-committing encryption scheme)**.** *We say a symmetric encryption scheme (E, D) is* non-committing *if there exist PPT (probabilistic polynomial-time) algorithms S_1, S_2 such that $\forall M \in \mathcal{M}$, (e', k') and (e, k) are computationally indistinguishable where $e' \leftarrow S_1(1^\kappa)$, $k' \leftarrow S_2(e', M)$ and $k \leftarrow \mathcal{K}$, $e \leftarrow E(k, M)$ (S_1, S_2 are allowed to share a state).*

Intuitively, this allows a "fake" ciphertext to be generated, such that it can be later explained as an encryption of any plaintext. This "cheating" then cannot be distinguished from normal usage of the encryption scheme given the ciphertext and key. The property is useful for simulating a ciphertext with some distribution, when the corresponding plaintext is not known at the time of encryption. Later, a suitable key can be generated such that the ciphertext decrypts to a specific plaintext.

Definition 2 (Robust encryption scheme)**.** *Let S be a set of random keys from $\mathcal{K}$ and $V_{S,e} \subseteq S$ the subset of valid keys for a given ciphertext e i.e., the keys in S such that $D(k, e) \neq \perp$.*

We say (E, D) satisfies robustness if for all ciphertexts $e \leftarrow A(1^\kappa, S)$ adversarially generated by a probabilistic polynomial-time $\mathcal{A}$, $|V_{S,e}| \leq 1$ except with negligible probability.

The intuition here is that it should be hard for an adversary to generate a ciphertext which can be decrypted to more than one valid plaintext using any polynomial number of randomly generated keys (even for adversaries who see those keys before generating the ciphertext). Note that this definition implicitly implies that the number of keys $|S|$ is at most polynomial (while $\mathcal{K}$ has exponential size) since the adversary is polynomial-time. Otherwise, say if $S = \mathcal{K}$, achieving the above definition would introduce an almost one-to-one mapping between ciphertexts and keys that can decrypt them, which makes the scheme completely useless. In particular, it is easy to see that Definition 1 would then be impossible to achieve.

The above two definitions together or separately most probably imply some sort of privacy guarantee for the encryption scheme (that it is hard to learn something about the plaintext given the ciphertext), although this is not explicitly stated in the article nor used in the proof. The authors give an example of a very simple scheme that satisfies both definitions.

Let $\mathcal{M} = \{0,1\}^l$ and $\mathcal{K} = \mathcal{C} = \{0,1\}^{l+\kappa}$ where κ is the security parameter. The encryption algorithm $E(k, m)$ parses k as (α, β) and outputs $e = (m \oplus \alpha, \beta)$. The decryption algorithm $D(k, e)$ parses $k = (\alpha, \beta)$ and $e = (e_1, e_2)$ and outputs $\perp$ if $e_2 \neq \beta$ or outputs $m = e_1 \oplus \alpha$ otherwise.

In other words, the encryption scheme is a one-time pad, where the last bits of the ciphertext fix a set of keys (exponential in size) that can decrypt it. Given this and the fact that for any ciphertext $e = (\alpha, \beta)$ and any plaintext m, we have $D((m \oplus \alpha, \beta), e) = m$, the following lemma holds.

Lemma 1. *The encryption scheme described above satisfies Definitions 1 and 2.* □

4.2 The protocol

Public parameters.

- Security parameter $\kappa \in \mathbb{N}$ and key space $\mathcal{K} = \{0,1\}^\kappa$.
- An additive group of prime order $(\mathbb{G}, G, p, +)$ such that given a representation of some group element x, it is efficient to verify whether $x \in \mathbb{G}$.
- A function $H : (\mathbb{G} \times \mathbb{G}) \times \mathbb{G} \to \mathcal{K}$ modelled as a random oracle.
- A symmetric encryption scheme (E, D) with message space $\mathcal{M}$ and key space $\mathcal{K}$ (non-committing and robust).

Note that in the following we use $H_{(A,B)}(C)$ to denote $H(A, B, C)$. This is to show that (A, B) is used as a "seed" for H. We present the Chou-Orlandi15 protocol as Protocol 2.

4.2.1 Correctness.

If both parties behave honestly, then $z^i_{c^i} = M^i_{c^i}$. This can be proven by showing that $k^i_{c^i} = k^i_{\mathcal{R}}$, in other words, the encryption key computed by $\mathcal{R}$ is exactly $k^i_{c^i}$ computed by $\mathcal{S}$. This holds for all $i \in [n]$, since

$$yR^i - c^iT = c^iT + x^iS - c^iT = x^iS$$

and therefore

$$k^i_{c^i} = H_{(S,R^i)}(x^iS) = k^i_{\mathcal{R}}$$

4.2.2 Privacy.

We will argue briefly why privacy of both parties is guaranteed in this protocol. First, let us consider receiver's privacy. The requirement here is intuitively that each R^i should reveal nothing about c^i. This is quite trivially the case, since R^i is a uniformly random element from $\mathbb{G}$, because x^i is uniformly randomly sampled. We can calculate the probability that R^i equals some element $aG \in \mathbb{G}$:

$$\forall a \in \mathbb{Z}_p : \Pr[R^i = aG] = \Pr[x^i = a - c^iS] = 1/p$$

independently of c^i. For sender's privacy, the authors show that if a corrupted $\mathcal{R}^*$ could produce $k^i_{j_0} \neq k^i_{j_1}$, then $\mathcal{R}^*$ could be used to break the computational Diffie-Hellman problem in $\mathbb{G}$ $((G, xG, yG) \mapsto xyG)$, with a quadratic reduction factor. The authors refer here to a proof from [7].

Sender		**Receiver**
Input: $(M_1^i, \dots, M_n^i) \in \mathcal{M}^n$		Input: $c^i \in [n]$
Output: none		Output: M_c
$y \leftarrow \mathbb{Z}_p$		
$S = yG, T = yS$		$x^i \leftarrow \mathbb{Z}_p$
	$\xrightarrow{S}$	
		if $S \notin \mathbb{G}$: abort
	$\xleftarrow{R^i = c^iS + x^iG}$	
if $R^i \notin \mathbb{G}$: abort		
$\forall j \in [n]:$		
$k_j^i = H_{(S,R^i)}(yR^i - jT)$		$k_{\mathcal{R}}^i = H_{(S,R^i)}(x^iS)$
	$\xrightarrow{e_j^i = E(k_j^i, M_j^i)}$	
		$z_{c^i}^i = D(k_{\mathcal{R}}^i, e_{c^i}^i)$

Protocol 2: Chou-Orlandi15 OT protocol. In the whole protocol, $i \in [m]$ for m parallel $\binom{n}{1}$-OT-s.

4.3 Analysis of the protocol

4.3.1 Ideal functionality.

The authors define the ideal functionality of the protocol directly for m parallel invocations of $\binom{n}{1}$-OT, for the message space $\mathcal{M} = \{0,1\}^l$ and κ-bit security between a sender $\mathcal{S}$ and receiver $\mathcal{R}$. This parallel definition is given in order to reuse the first message of the protocol for all m parallel transfers (saving on communication size of the first message) and prove security of this optimized parallel invocation.

The authors define a functionality $\mathcal{F}_{OT}^{-}$ (that defines the security of the protocol), which is slightly weaker than how an ideal parallel invocation would naturally be imagined. Conceptually, the functionality $\mathcal{F}_{OT}^{-}$ acts as follows. It receives as input m vectors of messages $\{(M_1^i, M_2^i, \dots, M_n^i)\}_{i\in[m]}$, $M_j^i \in \mathcal{M}$ from the sender $\mathcal{S}$ and a vector of choice indices $(c^1, \dots, c^m) \in [n]^m$ from the receiver $\mathcal{R}$. Then $\mathcal{F}_{OT}^{-}$ sends $(z^1, \dots, z^m)$ to $\mathcal{R}$ such that $z^i = M_{c^i}^i$.

However, the authors intentionally weaken the guarantees about independence of inputs for a corrupted receiver. In particular, a corrupted receiver $\mathcal{R}^*$ is allowed to provide his inputs $c^1, \dots, c^m$ *reactively* based on previous received messages. So in fact, $\mathcal{F}_{OT}^{-}$ does not perform the m OT-s perfectly *in parallel*. Instead, after receiving choice bit c^i from $\mathcal{R}^*$, $\mathcal{F}_{OT}^{-}$ sends back M_{c_i} and then waits for the next choice bit c^{i+1}. All M_j^i

are however sent to $\mathcal{F}_{OT}^{-}$ in a single message by $\mathcal{S}$. In short, independence of inputs in case of corrupted receiver is not entirely guaranteed by this proof for the m $\binom{n}{1}$-OT-s that reuse the first messages. We stress that independence of inputs is still guaranteed for any single $\binom{n}{1}$-OT execution as the receiver's input can only depend on messages from *previous* OT-s.

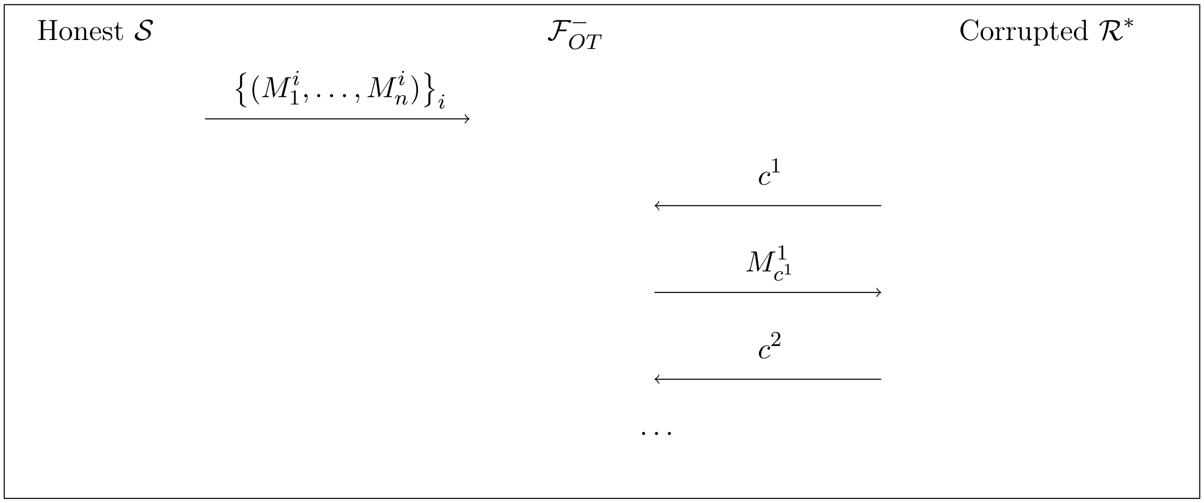


Figure 2: Description of ideal functionality $\mathcal{F}_{OT}^{-}$.

It might seem at first, that the security provided by functionality $\mathcal{F}_{OT}^{-}$ is undesirable in practice, since the corrupted receiver can maliciously choose his choices according to previous messages received, and that this somehow seems insecure. However, it is important to understand that the authors' formalization in fact provides slightly *stronger security guarantees* than if the ideal functionality would be defined for a single execution of $\binom{n}{1}$-OT only.

Consider that we have a UC-secure black box for doing a single $\binom{n}{1}$-OT. Even though the box is secure under general concurrent composition, this does not imply directly that executing *m different instances* of this box would somehow protect against independence of inputs *between* different OT-s. The UC composition theorem only allows to prove UC-security of a parallel OT protocol in a hybrid world, where the real-world protocol parties have access to an ideal $\binom{n}{1}$-OT functionality. But this means that the parallel OT is an entirely new functionality, whose security needs to be proven separately. It is not in general possible to construct this ideal parallel functionality from individual OT black boxes, without introducing some form of additional communication between the parties.

So in conclusion, the fact that there is guaranteed independence between all m OT-s for a corrupted sender, may be useful in some context (e.g when used as a subroutine in some larger protocol that can requires this). Still, one can argue, that this guarantee should hold for *all* simulatable receiver-private OT protocols, since the sender does not actually receive any output from the protocol. So informally speaking, since privacy of receiver's inputs should be guaranteed in any case, then the sender should not be able to learn any useful information that would allow him to choose his inputs in some meaningful way. Therefore, it is reasonable to assume that the motivation for defining $\mathcal{F}_{OT}^{-}$ in such a way in the Chou-Orlandi protocol is that of reducing network communication (the first message can be reused for parallel executions), not adding security. However, it is not clear why the weakening of the functionality is actually required, the conjecture of this report's author is that the provided security proof would otherwise not work, however, the concrete cause for this was not found.

4.3.2 Security proof.

The authors prove UC-security of the protocol against malicious adaptive polynomial-time adversaries in an asynchronous network model. They make the standard assumption that the parties use an authenticated (not private, but non-malleable) channel to communicate.

To prove security, the authors model H as a random oracle. Concerning computational assumptions, the proof needs that the computational Diffie-Hellman problem is hard in $\mathbb{G}$ and that the encryption scheme (E, D) satisfies the non-committing and robustness definitions presented above (the encryption scheme described in Section 4.1 is for example suitable).

If the reader has paid close attention to the discussion in Section 3 concerning random oracles, then a discrepancy here should be immediately visible. Namely, the authors model H, a hash function, as a random oracle in a UC security proof. However, a public hash function cannot be said to be local to each protocol instance, which is the requirement for the UC composability theorem to hold.

The authors remark on this issue and argue that always prepending the transcript (S, R^i) to input of H (basically, seeding H with (S, R^i)) "helps" in making the calls to H local to the specific protocol instance, since S, R^i are randomly generated and therefore, with high probability, these values are unique for every instance of the protocol. Also note that S and R^i are generated by different parties, so at least one of these values is generated correctly. However, there is no formal proof that quantifies, to which extent this assumption holds. It is a small technical detail, but one that nevertheless raises questions about the practical security of the protocol in concurrent composition (since the formal requirements for the UC composition theorem are not met). At least, the proof does soundly show full simulatability in the stand-alone model with respect to $\mathcal{F}_{OT}^{-}$ (but still relies on a random oracle).

4.3.3 Practical implementation

The protocol as described above implicitly assumes a number of things from a concrete instantiation of $\mathbb{G}$. First, an efficient representation of group elements as bit-strings is required, since the hash function H takes group element inputs. The representation should be on the one hand as concise as possible (to optimize communication) and that the same time, group operations should be efficient to compute. Also, the malicious security relies on an efficient operation to check whether a given bit-string representation is valid for the group $\mathbb{G}$.

The authors propose an elliptic curve group based on the the Curve25519 by Bernstein. The chosen curve is an Edwards curve that is birationally equivalent to Curve25519, and is also used for example in the EdDSA digital signature scheme. For more details, see [6]. The authors specify concrete representations for points on this curve and give a compressed 256-bit representation for transferring group elements. They use different representations for local computations, that allow to optimize the performance of costly group operations.

The authors demonstrate a concrete, optimized implementation of the protocol using this Edwards curve and leveraging vectorized hardware instructions. However, the code includes a lot of inline assembler instructions, and as such, is very platform specific and non-portable, which is often very much needed in practice. For the hash function H, the authors propose Keccak with 256-bit output [17], and as such, the ciphertext size for the robust, non-commiting encryption scheme is also 256 bits.

5 Comparison with other OT protocols

The classical most well-known OT protocols are Bellare-Micali89 [4] and Naor-Pinkas01 [27]. Neither is known to be fully-simulatable (let alone UC-secure), but achieve one-sided simulatability. Against malicious adversaries, only privacy is guaranteed, but not correctness or other properties that are discussed here earlier.

Other notable more recent protocols include Lindell08 [25], which is the first efficient plain model OT protocol secure against malicious adversaries that is fully-simulatable in stand-alone setting. The protocol describes a $\binom{2}{1}$-OT and generalizes to $\binom{n}{k}$-OT. However, compared to Naor-Pinkas01 for example, the complexity overhead is at least a multiplicative factor of a statistical security parameter (showing the probability that a simulation "fails"). Lindell claims 2^{-40} is sufficiently small failure rate, therefore, the communication and computation complexity is about 40 times larger, which is a *very* significant overhead.

Also, the ALSZ13 paper [1] presents an efficient protocol for doing m $\binom{n}{1}$-OT-s in the standard model, secure against semi-honest adversaries. Their proof is based on indistinguishability of corrupted party's views, not simulatability, which is enough for the semi-honest case. The ALSZ13 protocol can also be used in efficient OT-extensions presented in the same paper [1].

5.1 Bellare-Micali89

We now present one of the first well-established OT protocols — the Bellare-Micali89 protocol [4] (see Protocol 3). The essence of the protocol is that the receiver chooses an ElGamal secret and private key pair, and also a random public key based on a sender's message. Intuitively, the receiver can only decrypt messages encrypted with only one of these public keys. Therefore, the receiver encrypts both messages with the two keys, and the receiver can only decrypt the ciphertext corresponding to his chosen message.

Sender		**Receiver**
Input: $(M_0, M_1) \in \mathcal{M}^2$		Input: $c \in \{0, 1\}$
Output: none		Output: M_c
$S \leftarrow \mathbb{G}$		$x \leftarrow \mathbb{Z}_p$
	$\xrightarrow{\quad S \quad}$	
		$\mathsf{PK}_c = xG$
		$\mathsf{PK}_{1-c} = S - \mathsf{PK}_c$
	$\xleftarrow{\quad \mathsf{PK}_0, \mathsf{PK}_1 \quad}$	
if $\mathsf{PK}_0 + \mathsf{PK}_1 \neq S$: abort		
$r_0, r_1 \leftarrow \mathbb{Z}_p$		
$E_i = (r_i G, H(r_i \mathsf{PK}_i) \oplus M_i)$		
	$\xrightarrow{\quad E_0, E_1 \quad}$	
		$z_c = E_c[1] \oplus H(xE_c[0])$

Protocol 3: Bellare-Micali89 protocol for $\binom{2}{1}$-OT.

The security of the protocol relies on the DDH assumption in the group $(\mathbb{G}, G, p, +)$, however, the scheme only achieves one-sided simulatable security, meaning that correctness against malicious adversaries is not guaranteed. The proof also requires that the keyed hash function H that is used is *entropy smoothing* [3]. Intuitively, this property requires that given random k and random input m, the output of the hash function $H(m)$ is indistinguishable from a uniformly random bit-string. Such hash functions can be constructed using the left-over hash lemma [30].

The $\binom{2}{1}$-scheme does not generalize to naturally $\binom{n}{1}$-OT and a general construction from $\binom{n}{1}$-OT to $\binom{n}{1}$-OT has to be used (for example, the one discussed in 5.3).

5.2 Naor-Pinkas01

We present now the Naor-Pinkas01 OT protocol. We consider only the ROM protocol presented in the paper, that has a nice generalization to doing m $\binom{n}{1}$-OT-s. The paper also proposes a standard model protocol. Both protocols achieve one-sided simulatable security, but the ROM version is more efficient. The ROM version relies on the computational Diffie-Hellman assumption (CDH) and the hash function H is a random oracle.

Sender		**Receiver**
Input: $(M_1^i,\dots,M_n^i)\in\mathcal{M}^n$		Input: $c^i\in[n]$
Output: none		Output: M_c
$r\leftarrow\mathbb{Z}_p$		
$S=rG$		
$\forall j\in[n-1]:S_j\leftarrow\mathbb{G}$		$x^i\leftarrow\mathbb{Z}_p$
	$\xrightarrow{S,S_1,\dots,S_{n-1}}$	
		$\mathsf{PK}^i_{c^i}=xG$
		if $c^i\neq n:\mathsf{PK}^i_n=S_{c^i}-\mathsf{PK}^i_{c^i}$
	$\xleftarrow{\mathsf{PK}^i_n}$	
$R^i\leftarrow\{0,1\}^{2\log m}$		
$\forall j\in[n-1]:$		
$(r\mathsf{PK}^i_j)=rS_j-r\mathsf{PK}^i_n$		
$E^i_j=H(r\mathsf{PK}^i_j,R^i,j)\oplus M^i_j$		
	$\xrightarrow{R^i,E^i_1,\dots,E^i_n}$	
		$z^i_c=E^i_c\oplus H(x^iS,R^i,c^i)$

Protocol 4: Naor-Pinkas01 ROM protocol for doing m $\binom{n}{1}$-OT-s ($i\in[m]$).

Similarly to the Chou-Orlandi15 protocol, the first message of the sender can be reused for all the m OT-s. The authors note that the random bit-string R^i should be sufficiently long so that the same R does not occur in any pair of these m $\binom{n}{1}$-OT invocations. They suggest a length of $2\log m$ to make this probability negligible.

5.3 $\binom{n}{1}$-OT from black-box $\binom{2}{1}$-OT

We briefly mention a general method for using a secure $\binom{2}{1}$-OT to build a secure $\binom{n}{1}$-OT protocol [28]. The construction relies on *pseudo-random function* and as such, can give only computational security for both corrupted sender and receiver cases. In fact, the construction uses pseudo-random permutations (PRP). Efficient PRP-s can be instantiated as block ciphers (e.g AES) or using keyed hash-functions (e.g HMAC).

The construction itself can be found in [28]. We only note here that to get a $\binom{n}{1}$-OT, it requires doing $\lceil\log_2 n\rceil$ of $\binom{2}{1}$-OT-s in parallel with an additional communication overhead of nl bits, where l is the message length. We use this technique to compare Bellare-Micali89 with the other protocols for doing m $\binom{n}{1}$-OT-s.

Note that this construction only maintains computational one-sided simulatable security. In other words, the receiver's security is defined via computational indistinguishability of corrupted sender's view, and sender's security is defined via ideal vs real world simulatability (also computational).

6 Conclusion

OT prot.	**$\mathcal{S}$ Comp.**		**$\mathcal{R}$ Comp.**		**Comm. (bytes)**
	$\mathbb{G}$ exp.	**$\mathbb{G}$ mult.**	**$\mathbb{G}$ exp.**	**$\mathbb{G}$ mult.**	
Bellare-Micali89	$4m\lceil\log_2 n\rceil$	$m\lceil\log_2 n\rceil$	$2m\lceil\log_2 n\rceil$	$m\lceil\log_2 n\rceil$	$32m\,(6\lceil\log_2 n\rceil + 4n)$
Naor-Pinkas01 (ROM)	$n+m$	$m\,(n-1)$	$2m$	m	$32m\left(n+\frac{\log_2 m}{16}+1\right)+ 32n$
Chou-Orlandi15	$2+m$	mn	$2m$	$m+n-1$	$32m(n+1)+32$
Chou-Orlandi15 Random-OT	$2+m$	mn	$2m$	$m+n-1$	$32(m+1)$

Table 1: Comparison of performance of OT protocols for $m\ \binom{n}{1}$-OT-s

We conclude with an overview of the theoretical computational performance and estimated communication complexity of different protocols in Table 1. We list computation for both $\mathcal{S}$ and $\mathcal{R}$ and total communication for doing $m\ \binom{n}{1}$-OT-s. For computation, we report the number of group operations that need to be done based on the theoretical protocol description. The group operations are named assuming a multiplicative group, which means for the Chou-Orlandi15 protocol, exponentiations denote multiplications with scalar and multiplications denote group addition. We do not other computational complexity of symmetric primitives used, such as hash functions, since these are in practice much faster than group operations.

Note that Chou-Orlandi15 report very different computation complexity in the paper abstract [14]. We believe our calculations to be more exact and do not understand how the authors obtained the reported numbers in the article. We specifically minimize the number of exponentiation, as for example, jT for $j \in [n]$ and $T \in \mathbb{G}$ can be computed with $n-1$ group additions, given T.

We calculate the communication cost assuming 128-bit security and 128-bit messages. In the Chou-Orlandi15 protocol, the length of the hash output is 256 bits, since Keccak is used. The ciphertext size is then also 256 bits, which fits 128-bit security, since it is equal to the message size plus the security parameter of the encryption scheme. The group elements are also transferred using bit-strings of length 256. For comparability, we assume other protocols also use a group that has an efficient 256-bit element representation.

However, the ciphertext sizes are different. In Bellare-Micali89, the ciphertext consists of a group element and the message masked with the hash output. Therefore, we get a 384-bit ciphertext. In contrast, Naor-Pinkas01 protocol only masks the message, and the ciphertext size is therefore 128 bits. Also note that we use the general $\binom{n}{1}$-OT construction from [28] for estimating Bellare-Micali89 performance for doing m $\binom{n}{1}$-OT-s. This might not be the most efficient way, and certainly there are possible trade-offs in terms of computation and communication complexity that can be done. This also holds for the Naor-Pinkas01 protocol, since they also describe possible trade-offs in their paper.

Concerning the Chou-Orlandi15 protocol, it seems to provide currently one of the best performance for fully-simulable OT protocols secure against malicious adversaries. It also *somewhat* achieves UC-security, however, without formal justifications, it is reasonable to assume that the non-local use of the random oracle cannot guarantee security in concurrent composition with *completely arbitrary* protocols, in particular those that use the same hash function.

As always, there is the trade-off between performance and committing to the random oracle model, which might be a subjective matter based on the actual context where the protocol is applied.

The highly optimized reference implementation, which includes many low-level details of efficient elliptic curve operations, is very useful from a practical point of view. The code is highly non-portable though, and as such, it would take some engineering effort for implementing the protocol in practice. A reasonably-optimized implementation however, should not be too difficult, as the concepts of elliptic curve representations are well discussed in the paper.

Acknowledgment

The author of this report has received the Skype and Study IT in Estonia Master's Scholarship for the academic year 2015/16, funded by Estonian Information Technology Foundation and Skype Technologies OÜ.

References

[1] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 535–548. ACM, 2013.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, volume 9056 of *Lecture Notes in Computer Science*, pages 673–701. Springer, 2015.

[3] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Formal certification of elgamal encryption. In P. Degano, J. D. Guttman, and F. Martinelli, editors, *Formal Aspects in Security and Trust, 5th International Workshop, FAST 2008, Malaga, Spain, October 9-10, 2008, Revised Selected Papers*, volume 5491 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008.

[4] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557. Springer, 1989.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73. ACM, 1993.

[6] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang. High-speed high-security signatures. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2011.

[7] E. Bresson, O. Chevassut, and D. Pointcheval. New security results on encrypted key exchange. In F. Bao, R. H. Deng, and J. Zhou, editors, *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, March 1-4, 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 145–158. Springer, 2004.

[8] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[9] R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[10] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[11] R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 597–608, 2014.

[12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In J. H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 494–503. ACM, 2002.

[13] R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

[14] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2015.

[15] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[16] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[17] M. P. G. V. A. G. Bertoni, J. Daemen. The keccak reference, January 2011.

[18] M. Geisler. *Cryptographic protocols: theory and implementation.* PhD thesis, Aarhus University, February 2010.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[20] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions.* Information Security and Cryptography. Springer, 2010.

[21] M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741. Springer, 2015.

[22] J. Kilian. Founding cryptography on oblivious transfer. In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31. ACM, 1988.

[23] N. Koblitz and A. J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptography*, 77(2-3):587–610, 2015.

[24] G. Leurent and P. Q. Nguyen. How risky is the random-oracle model? In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2009.

[25] Y. Lindell. Efficient fully-simulatable oblivious transfer. *Chicago J. Theor. Comput. Sci.*, 2008, 2008.

[26] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[27] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In S. R. Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 448–457. ACM/SIAM, 2001.

[28] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.

[29] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[30] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[31] A. C. Yao. Protocols for secure computations. In *Proc. of SFCS'82*, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.