How to Break a "Secure" Oblivious Transfer Protocol

Donald Beaver

313 Whitmore Lab, Penn State University, State College, PA 16802, USA, (814) 863-0147, beaver@cs.psu.edu.

Abstract.

We show how to break a protocol for Oblivious Transfer presented at Eurocrypt 90 [11]. Armed with a new set of definitions for proving the security of interactive computations, we found difficulties in proving the protocol secure. These difficulties led us to a simple attack that breaks the OT protocol in a subtle but fundamental way. The error that we found may be present in a wide variety of secure protocols. It reveals a fundamental flaw in the traditional definition of Oblivious Transfer itself.

1 Introduction

Solid proofs are a lacking but essential requirement for cryptography. Whereas a failed claim in complexity theory might mean an algorithm gives errors sometimes, a failed claim to security might provide a huge potential for malicious destruction of data, resources, and dependability.

Proofs are often lacking because clear and simple definitions are hard to come by. The value of good definitions goes beyond the confidence they inspire in proven results. When good definitions are present, a concise proof can usually be found; the lack of one, or the difficulty of finding one, often indicates that a theorem is incorrect. In fact, truly clear definitions and proof techniques often turn up counterexamples when applied to an incorrect conjecture.

In [1, 3], Beaver proposed a concise set of definitions that provide not only a clear, intuitive understanding of security in interactive protocols but that support direct and modular proofs. In this paper, we show how these definitions revealed a subtle flaw in a protocol claimed to be "provably" secure.

We consider the problem of Oblivious Transfer (OT), a fundamental cryptographic problem introduced by Rabin [17]. OT is a two-party protocol in which Alice transmits a bit b to Bob, and Bob receives the bit with probability $\frac{1}{2}$. Alice must not learn whether Bob received the bit. OT forms the basis for a wide variety of cryptographic protocols [18, 15, 6, 8].

Based on assuming that determining quadratic residuosity is hard, Rabin suggested an elegant but inefficient protocol for OT, requiring the generation of a large integer $n = pq$ (with p, q prime) for each bit to be sent [17].

Den Boer suggested an efficient protocol for OT in Eurocrypt '91 [11], relying on the same assumption but allowing the re-use of n and ensuring unconditional security of b. We show that this protocol is insecure.

The flaw is that Alice can send a bit b without knowing what b is — potentially giving her information about the quadratic residuosity of numbers of her choice, without her knowing the factors of n. In fact, after a single transfer she may be able to break all subsequent transfers.

Interestingly, the protocol *is* provably secure according to the traditional list of required security properties, if one confirms "Alice does not learn whether Bob received b" using the approach of of zero-knowledge [14]. As we shall see, zero-knowledge is insufficient as as a measure of the security of OT (and of other protocols). Rabin's protocol (apparently serendipitously) does not suffer from the same problem, but our analysis shows how to break Rabin's protocol in a network of three or more players.

Instead of proposing a new requirement for OT, we define the security of OT according to a unifying property called *resilience*. Resilience captures all known security properties and, we submit, it implies all security properties *a priori*. Unforeseen properties such as transparency (the inability to bluff) arise as newly-observed implications of our unified definition, as this paper exemplifies.

Using resilience, we were able to expose the subtle flaw in the OT protocol within minutes. We fix the flaw and sketch a proof of security, thus salvaging den Boer's brilliant idea and providing an efficient protocol for Oblivious Transfer.

2 Oblivious Transfer: the Traditional Approach

Traditionally, the goal of Oblivious Transfer is to find a protocol satisfying certain properties:

1. Alice sends bit b but does not know whether Bob receives it;
2. Bob receives $(1, b)$ or $(0, 0)$ [*resp.* "received b" or "got nothing"] with equal probability, but receives no additional information;
3. Both players can abort the protocol by deviating from it in a syntactic sense (*eg.* Alice does not send 0/1 or Bob sends "quit").

The usual formalization of "no additional information" or "does not know" uses a zero-knowledge approach: a simulator must demonstrate that Alice (or Bob) could generate an accurate view of the interaction based *only* on given, limited information. But zero-knowledge is not enough to guarantee that Alice learns nothing.

We show that a previously-overlooked property is essential, namely that Alice not be able to bluff her way through the protocol:

4. Alice must know the effective bit b she sends.

Rather than adding to a potentially incomplete list of properties, we examine a single property, *resilience*, which implies (1)-(4) above and, by virtue of its clarity and wide applicability, seems to capture properties as yet unobserved.

A simple and direct fix for protocols lacking property (4) is to require Alice to give a zero-knowledge proof that she knows her effective input b. This turns out to be sufficient to construct a protocol secure under our unified definition. First, however, let us examine why the ability to bluff should be restricted and how it can be used to break den Boer's protocol.

3 Oblivious Transfer in an Ideal World

In the ideal case, Alice would give b to a trusted host (noisy channel) that would then send $(1, b)$ or $(0, 0)$ to Bob with equal probability, without informing Alice which one it sent. In a very real sense, the goal of Oblivious Transfer is to implement this ideal protocol without having a trusted host available.

Imagine for the moment that a trusted host is available - perhaps in the form of a quantum channel [9]. Consider the following scenario: Alice sends b_1 to Bob, and later says, "I think I got some static on the line; can we test it?" Bob agrees, and Alice sends a second bit b_2, and Bob reports his second result $((0,0)$ or $(1, b_2))$.

Certainly, in the ideal case, Alice would not learn anything about the results of the first execution, *eg.* whether Bob received b_1 or not. Any implementation of OT should not allow later executions to compromise earlier ones, even when Bob reveals the later results.

4 Oblivious Transfer: Background

4.1 Notes on Cryptography

A **Blum integer** is a product $n = pq$ of two equally-sized primes of the form $p = 4k+3$, $q = 4l+3$; let BLUM_k be the set of such numbers of size k. A number x is a **quadratic residue** (mod n) iff it has a square root (mod n). The integers modulo Blum integer n having Jacobi symbol $+1$ form a multiplicative group, $\mathbf{Z}_n^+$, of which half are residues. Define $Q_n(x) = 0$ iff $x \in \mathbf{Z}_n^+$ is a residue, or 1 if not; note $Q_n(ab) = Q_n(a) \oplus Q_n(b)$. For Blum integers, $Q_n(-1) = 1$.

The notation $x \leftarrow X$ indicates x is sampled uniformly at random from set X. An **ensemble** is a function mapping a pair (z, k), with $z \in \Sigma^*$ and $k \in \mathbf{N}$, to a distribution on strings of size at most $(|z|k)^c$ for some c. If P and Q are ensembles, and if M is a TM or function, define the **distinguishing power** of M to be

$$\delta_M(z,k) \overset{\text{def}}{=} |\Pr[M(P(z,k)) = 1] - \Pr[M(Q(z,k) = 1)]|.$$

Ensembles P and Q are **statistically indistinguishable**, written $P \approx Q$, if for all functions M, and for all z and c, $\delta_M(z,k) = 0(k^{-c})$. Ensembles P and Q are **computationally indistinguishable**, written $P \overset{c}{\approx} Q$, if for all probabilistic poly-time TM's M, and for all z and c, $\delta_M(z,k) = 0(k^{-c})$.

The **Quadratic Residuosity Assumption (QRA)** states that random residues in BLUM_k are computationally indistinguishable from random non-residues. A more general version states the same for all products of two primes.

RABIN-OT(b, k)

1. Alice chooses $n = pq \leftarrow \text{BLUM}_k$, remembers (p, q), selects $t \leftarrow \mathbf{Z}_n^*$, and sends $(n, (-1)^b t^2)$ to Bob.
2. Bob chooses $r \leftarrow \mathbf{Z}_n^*$ and sends $x \equiv r^2 (\text{mod } n)$ to Alice.
3. Alice chooses s randomly from the four square roots of x and sends s to Bob.
4. If $s \boxminus \pm r$ then Bob concludes $(0, 0)$; otherwise Bob factors n using $\gcd(r + s, n)$, computes $b = Q_n((-1)^b r^2)$, and concludes $(1, b)$.

Fig. 1. Rabin's Oblivious Transfer protocol: Alice sends b to Bob; k is a security parameter.

4.2 An Early Implementation

Figure 1 describes Rabin's protocol for OT, for security parameter k. A flaw noted long ago is that Bob might choose x in a different manner, obtaining illegal information from the protocol: for example, a square root of a number whose roots he does not know. The simple correction is to require Bob to prove in zero-knowledge that he already knows a square root of x. A second flaw is that Alice may cheat by using an n that is not a product of two primes; thus, she too should prove in zero-knowledge that n is the product of two primes. The corrected version satisfies a definition of OT using the property list given above, assuming on the QRA; in fact, it satisfies our definitions given below, if Alice must also prove she *knows* the two factors of n.

Unfortunately, Rabin's protocol is inefficient: it requires Alice to generate a new, large Blum integer for every bit to be transferred. Furthermore, the secrecy of b is not unconditional: it depends on Bob's inability to determine quadratic residuosity.

5 Breaking an Oblivious Transfer Protocol

At Eurocrypt '91, den Boer [11] presented a protocol (see Fig. 2) that requires only *one* generation of a large product of two primes and that ensures the secrecy of bit b unconditionally. In contrast to Rabin's protocol, den Boer's protocol assumes that *Bob* knows the factors of n while Alice does not. Together, Alice and Bob generate a number $J^c a$ (where J is a nonresidue) whose residuosity is random and unknown to Alice. Alice "encodes" b as $x = J^b r^2$; she sends x and $y = J^c a x^{-1}$ to Bob. If $Q_n(J^c a) = 0$, implying that $Q_n(x) = Q_n(y) = b$, then Bob can compute b, else he cannot.

The fundamental flaw in this protocol is that Alice can cheat by selecting x without knowing its residuosity, effectively transmitting some bit b without knowing its value. This apparently innocuous flaw ("Who cares if Alice knows less?") is far more significant than it seems. One main advantage to den Boer's protocol is that it does not require repeated generation of large Blum integers

BOER-OT(b, k)

1. Bob chooses $n \leftarrow pq$, remembers (p, q), selects a nonresidue J, selects $a \leftarrow Z_n^+$, and sends (n, J, a) to Alice. Bob proves in zero-knowledge that n is the product of two primes and J is a nonresidue.
2. Alice selects $c \leftarrow \{0, 1\}$ and $r \leftarrow Z_n^*$ and sets $z = J^c a$. Alice computes $x = J^b r^2 (\text{mod } n)$ and sets $y = zx^{-1} (\text{mod } n)$. Alice sets $(u, v) = (x, y)$ if $x < y$, else $(u, v) = (y, x)$. (This is equivalent to taking (x, y) in random order). She sends (u, v) to Bob.
3. Bob checks that $uv \equiv a$ or $uv \equiv Ja$. Bob computes $Q_n(uv)$; if 0, he outputs $(1, Q_n(u))$ ("received $Q_n(u)$"), else he outputs $(0, 0)$ ("received nothing").

Fig. 2. Den Boer's Oblivious Transfer protocol.

for every bit to be sent; it can also be used directly for string-transfer (by using the same a for many bits). But Alice can break the protocol exactly in those situations where more than one bit must be transferred. Even though the protocol may seem secure when used only once, subtle flaws such as these preclude its use as a black-box subroutine — an essential property for cryptographic protocols.

To be concrete, let us consider the simple scenario described in §3. Alice sends b_1 and later sends b_2. (We use subscripts 1 and 2 to denote the two executions.) By directly asking Bob as described in §3 (or perhaps more realistically, by observing Bob's later behavior) Alice learns whether Bob received b_2 or not — without being told anything directly about the results of sending b_1. Because the sending of b_2 and b_1 should be independent, this should not really be a problem.

But a clever Alice misbehaves during the second execution, setting $x_2 \leftarrow a_1 t_2^2$ (where a_1 was the value used for b_1). If Bob reports he received b_2, then Alice now knows that $Q_n(a_1) = b_2$, so she can calculate precisely whether Bob received bit b_1!

Knowing that Alice can send bits without knowing their value, the interested reader is invited to consider other more and less subtle ways to break the protocol or at least to gain unfair advantage. Alice's ability to bluff her way through is essential to her attack.

The devil's advocate may complain that Bob should never go along with Alice's later requests, to prevent Alice from deducing anything. In this case, Bob's actions must always be completely independent of the results he obtains — this includes cases where he detects cheating, since Alice can derive subtle and compromising information even from an accusation of cheating. This is an extremely stringent handicap to put on a protocol, hardly applicable to any realistic situation. Protocols should be secure enough to be treated as black-boxes, called at will and independently, without interdependencies that compromise security.

6 Why OT Has Always Been Defined Incorrectly

Until now, the requirement that Alice *know* the bit b that she sends has not been made explicit (to the best of our knowledge). Protocol BOER-OT fails exactly for this reason. Rabin's OT protocol survives because Alice knows the factors of n and hence could calculate the effective bit b that she sent, even if she generated numbers illegally. (Let us point out that even this assurance is subject to the assumption that Alice's proof of $n \in \text{BLUM}_k$ demonstrates *knowledge* of p and q).

Intuitively, bluffing allows Alice to observe effects that she could not predict by herself. If we design a protocol that requires Alice's attack to be *transparent* (namely, Alice's effective bit b should be predictable from her *view*), then Alice cannot play subtle games without knowing in advance what their effects will be. Thus she does not gain information to which she is not entitled. This is a situation quite different from zero-knowledge.

We could add "Alice knows b" to the list of properties required by OT, but we would have no guarantee that our human powers of observation have still not overlooked some essential property. Instead, we turn to a single property, *relative resilience*, that defines exactly what it means for a real protocol to implement an *ideal* specification, without having to list separate properties.

7 Defining and Proving Security

We define a general security reduction among protocols that states precisely how one protocol implements another securely and fault-tolerantly. We sketch results developed in [3] for the information-theoretic setting and in [2] for resource-bounded computation.

Information-Theoretic Security. Intuitively, protocol α is as secure as protocol β if the attack of any allowed adversary $\mathcal{A}$ against α wreaks as much havoc on β as on α. Essentially, $\mathcal{A}$ gains the same *information* and wields the same *influence* over correct outputs in β as in α. Of course, $\mathcal{A}$ might not understand β (*eg.*, β might have a different communication format), so we give it an *interface* $\mathcal{I}$. The interface provides a convincing α-environment to $\mathcal{A}$, attempting to bring $\mathcal{A}$ to a final state as though $\mathcal{A}$ had really attacked α. At the same time, $\mathcal{I}$ attacks β, getting the information it needs and attempting to induce the same results in honest processors as when $\mathcal{A}$ attacks α. (These two goals are inseparable; $\mathcal{I}$ must achieve them simultaneously.) Thus the view of $\mathcal{A}$ (its *information*) and the outputs of nonfaulty players (reflecting $\mathcal{A}$'s *influence* on correctness) are the same in both protocols. If $\texttt{ADV}_\alpha$ is the set of allowable adversaries against α and $\texttt{ADV}_\beta$ that against β, $\mathcal{I}(\mathcal{A})$ should of course be in $\texttt{ADV}_\beta$.

An execution of a protocol α with n players on inputs $x_1, \ldots, x_n$ (and auxiliary inputs $a_1, \ldots, a_n$ for the players, a_A for $\mathcal{A}$) induces some distribution on outputs $y_1, \ldots, y_n$ and views $v_1, \ldots, v_n$ of good players and the output/view y_A of $\mathcal{A}$. We let $[\mathcal{A}, \alpha](\mathbf{x} \circ \mathbf{a} \circ a_A, k)$ denote the distribution on $(v_A, y_1, y_2, \ldots, y_n)$, namely on adversary-view and honest-outputs. Ranging over all possible inputs and security parameters, this collection of distributions is an ensemble, $[\mathcal{A}, \alpha]$. When $\mathcal{A}$ is allowed to attack another protocol β through an interface $\mathcal{I}$, the corresponding ensemble is denoted $[\mathcal{I}(\mathcal{A}), \beta]$. The restriction to player outputs (resp., adversary view) is denoted by subscript Y (resp., Y_A).

[Aside: for technical reasons, we consider a slight modification allowing the adversary to perform "post-protocol corruption," namely to elect to compute and corrupt even after the protocol is finished. In this case (and only in this case), $\mathcal{A}$ receives *all player outputs* (but not views) and $\mathcal{I}$ must continue to provide $\mathcal{A}$ with accurate α-views when $\mathcal{A}$ corrupts new players. This tests the ability of $\mathcal{I}$ to create accurate α-views — which, technically, it is otherwise not required to do after the protocol is finished. Without further explanation (but see [3]), such strong interfaces permit us to prove that sequential protocol concatenation is secure. The ensembles $[\mathcal{A}, \alpha]$ and $[\mathcal{I}(\mathcal{A}), \beta]$ are defined to include the cases when $\mathcal{A}$ elects post-protocol corruption.]

An interface is called **parsimonious** if it corrupts the same pattern of players as that listed in the adversary-view v_A it induces.

The following definition is a preliminary formalization of the notion that, if effects of attacks on α match those of interface-assisted attacks on β, then α is as secure as β.

Definition 1. Let $\mathtt{ADV}_\alpha$ denote a class of adversaries allowed to attack α. Protocol α is **(info-theoretically) as resilient as** β if there exists a parsimonious interface $\mathcal{I}$ such that, for all $\mathcal{A} \in \mathtt{ADV}_\alpha$, we have $\mathcal{I}(\mathcal{A}) \in \mathtt{ADV}_\beta$ and

$$[\mathcal{A}, \alpha] \approx [\mathcal{I}(\mathcal{A}), \beta].$$

Ideal Protocols. An **ideal protocol** contains one or more *trusted hosts* that are incorruptible. All desirable security properties are, by definition, observations about an ideal situation. The **ideal protocol** $\mathtt{ID}(F)$ **for function** F consists of a trusted host that accepts inputs, computes F, and returns the outputs. The **ideal OT protocol** contains a trusted host that accepts b from Alice and sends $(1, b)$ or $(0, 0)$ to Bob with equal probability; either player can send a **quit** message to the host to abort the protocol. We shall declare a protocol secure if it achieves what an ideal protocol achieves; but first, we consider some computational issues.

Computational Issues. In the computational setting, we are worried about obtaining information that is not efficiently computable, so we require that $\mathcal{I}$ be poly-time regardless of how it accesses $\mathcal{A}$ (*eg.* as a black-box, resettable black-box, *etc.*). This restricts $\mathcal{A}$'s information to be a feasible function of the information $\mathcal{I}$ gains in β. A subtle but crucial point to note is that in β, $\mathcal{I}$ *knows* explicitly the messages it sends; it cannot learn its effective inputs, because it must send them itself. (We define "effective input" as the collection of messages sent and received by $\mathcal{I}$, generalizing the intuitive but restricted notion.) To make our mapping from α to β accurate, we must require that $\mathcal{A}$ cannot discover its effective inputs; it must know them.

To this end, we require a *translator* that maps progressive stages of $\mathcal{A}$'s view to the messages sent and received by $\mathcal{I}$ in β. This approach to specifying effective inputs from views generalizes the input-committal function introduced in 1989 by Beaver [2]. We consider stages because we must ensure that $\mathcal{A}$ knows its input before receiving a response, *ie.* that the execution of α corresponds temporally to the execution of β. To avoid notational inconvenience and save space, we now restrict our attention to OT, where in the ideal protocol $\beta = \texttt{ID}(OT)$, Alice merely sends one message, and Bob merely receives one message. Thus the issues of timing are simplified, and we require merely that a translator map faulty Alice's view (in α, the implementation) to the bit b that $\mathcal{I}$ sends on her behalf (likewise, the translator maps faulty Bob's α-view to the pair $(0,0)$ or $(1,b)$, a much simpler task).

Let $\mathcal{T}$ be a translator, namely a machine that (synchronously) maps v_A to the messages sent and received by $\mathcal{I}$. The ensemble $[\mathcal{A},\alpha,\mathcal{T}]$ is induced by running α attacked by $\mathcal{A}$, and including the outputs of $\mathcal{T}$ run on the views of the participants ($\mathcal{A}$ and honest players). The ensemble $[\mathcal{A},\beta]$ is now taken to include the conversations (transcripts of messages sent and received) by $\mathcal{A}$ and the honest players. The output by $\mathcal{T}$ should match the conversations.

Definition2. A **transparent interface** is an interface-translator pair $(\mathcal{I},\mathcal{T})$. Protocol α is **computationally as resilient as** β, written $\alpha \overset{\centerdot}{\succeq} \beta$, if there exists a transparent interface $(\mathcal{I},\mathcal{T})$ such that:

(1.) $\mathcal{I}(\mathcal{A}) \in \texttt{ADV}_\beta$;
(2.) $\mathcal{I}(\mathcal{A})$ is probabilistic poly-time;
(3.) $\mathcal{I}$ is parsimonious;
(4.) $\mathcal{T}$ is probabilistic poly-time;
(5.) The effects of attacks on α match those for β:[1]

$$[\mathcal{A},\alpha,\mathcal{T}] \overset{c}{\approx} [\mathcal{I}(\mathcal{A}),\beta] \text{ and } [\mathcal{A},\alpha,\mathcal{T}]_Y \approx [\mathcal{I}(\mathcal{A}),\beta]_Y.$$

We remark specifically that the *entire view* of $\mathcal{A}$ is considered for transparency. An alternate definition based on just the messages that $\mathcal{A}$ sends and receives but not its internal bits (*cf.* "traffic" in [16]) would imply that there exists absolutely no protocol for OT — nor could there be any secure encryption, for that matter.

Definition3. We say α **implements** β if $\alpha \overset{\centerdot}{\succeq} \beta$ and $\beta \overset{\centerdot}{\succeq} \alpha$.

Definition4. Protocol α is a **resilient protocol for** F if it implements the ideal protocol for F. Protocol α is a **resilient OT protocol** if it implements the ideal protocol for OT.

[1] The requirement that $[\mathcal{A},\alpha,\mathcal{T}]_Y \approx [\mathcal{I}(\mathcal{A}),\beta]_Y$ (*ie.* statistical indistinguishability of nonfaulty outputs) addresses an ongoing philosophical debate: should an answer be mathematically correct or just indistinguishable from a correct answer? There are pros and cons, but the important point is that this decision is independent of the rest of the definition. Compare Beaver, Micali, and Rogaway in [8].

We remark for the interested reader that these definitions support *proofs* that resilience is transitive and that sequential (black-box) compositions preserve resilience (see [1, 3]).

Let us also remark that the inclusion of a translator provides a notion of security more stringent than otherwise necessary: the theory stands on its own if the translator is omitted, as long as we continue to require that $\mathcal{I}$ be poly-time. Even if we do not require the translator, the arguments of §8 show that a proof of security — *ie.* a satisfactory interface — cannot be found. But the translator helps make it explicit that the adversary must know the effective inputs it uses, and it provides a useful tool to detect vulnerability to bluffing attacks.

8 Finding and Fixing Security Holes

In an attempt to find a transparent interface that maps attacks on protocol BOER-OT to attacks on the ideal OT protocol, two problems arose. We consider primarily the harder situation, when $\mathcal{A}$ chooses to corrupt Alice.

The first is fixable: how would $\mathcal{I}$ come up with bit b to send to the trusted host? Interface $\mathcal{I}$ could create an "environment" for an adversary $\mathcal{A}$ that manipulates Alice, by playing the role of Bob. The problem is that $\mathcal{I}$ would then obtain bit b from $\mathcal{A}$ only half the time, so it might have to "reset" $\mathcal{A}$ until $\mathcal{I}$ gets the bit b that $\mathcal{A}$ sends. Then $\mathcal{I}$ can send this bit to the trusted host, and in the ideal protocol, Bob receives it with probability $\frac{1}{2}$. Clearly, $\mathcal{I}$ is poly-time.

The second problem is inescapable. Even though $\mathcal{I}$ can send bits to the trusted host with the same probability as $\mathcal{A}$ (hence inducing a correct distribution on final outputs in the ideal scenario), it does not make $\mathcal{A}$'s attack transparent: $\mathcal{A}$ might not be able to compute the bit b it effectively sent. Provably, no polynomial-time machine $\mathcal{M}$ can determine whether $\mathcal{I}$ sends 0 or 1 to the trusted host, based on $\mathcal{A}$'s view. Assume otherwise; consider an adversary $\mathcal{A}$ that makes Alice generate x at random. Then the output of $\mathcal{M}$ must be $Q_n(x)$, which is what the interface passes on to the trusted host. Intuitively, the output of $\mathcal{M}$ is the *effective* bit that corrupt Alice sends. But this would mean that Quadratic Residuosity is not hard. Thus:

- Either the QRA is untrue and protocol BOER-OT is therefore invalid, or
- the QRA is true and the protocol is insecure because Alice's view is not translatable, *ie.* because it fails to have a transparent interface, *ie.* because it permits a "bluffing" attack in which Alice does not know b.

8.1 How to Fix den Boer's Protocol

Clearly, we must require Alice to prove that she knows $Q_n(u)$ or $Q_n(v)$. It is a fairly straightforward exercise to come up with a direct number-theoretic method for Alice to demonstrate such knowledge without revealing whether u is x or y (we may adapt [13] to these purposes, which is especially suitable because the verifier has already generated a suitable n). Note, however, that this significantly increases the round complexity and message complexity of the protocol. Fortunately, many such demonstrations can be done in parallel. Fortunately as well, it seems that the added computations seem less expensive than the alternative: generating many large Blum integers. Our main theorem becomes:

BB-OT(b, k)

1. Bob chooses $n = pq \leftarrow \text{BLUM}_k$, remembers (p, q), selects $a \leftarrow Z_n^+$, and sends (n, a) to Alice. Bob proves in zero-knowledge that $n \in \text{BLUM}_k$.
2. Alice selects $c \leftarrow \{0, 1\}$ and $r \leftarrow Z_n^*$ and sets $z = (-1)^c a$. Alice computes $x = (-1)^b r^2 \pmod n$ and sets $y = zx^{-1} \pmod n$. Alice sets $(u, v) = (x, y)$ if $x < y$, else $(u, v) = (y, x)$. She sends (u, v) to Bob and proves in zero-knowledge that she knows the residuosity of one of u or v.
3. Bob checks that $uv \equiv \pm a \pmod n$. Bob computes $Q_n(uv)$; if 0, he outputs $(1, Q_n(u))$ ("received $Q_n(u)$"), else he outputs $(0, 0)$ ("received nothing").

Fig. 3. Our corrections to den Boer's protocol, along with some modifications. ("BB" represents "Beaver/Boer" or "Boer/Beaver.")

Theorem 5. *There exists an efficient protocol for OT that is computationally resilient.*

Proof Sketch. Figure 3 outlines the corrected protocol. We can now find a transparent interface and prove the modified version secure. If $\mathcal{A}$ corrupts Alice, $\mathcal{I}$ simulates Bob internally and corrupts ideal-Alice. Whenever Alice fails to prove that she knows how (u, v) was constructed (*ie.* that she knows b), $\mathcal{I}$ sends quit to the trusted host, causing ideal-Bob to output ABORT, just as the "real" Bob does. If the proof succeeds, then the interface can in fact derive the bit b from Alice's view (briefly: by resetting a copy of $\mathcal{A}$ and making different challenges, thereby extracting b), and it sends b to the trusted host. Because the proof of knowledge is such that a successful proof indicates b can be efficiently extracted from $\mathcal{A}$'s view, not only $\mathcal{I}$ but $\mathcal{T}$ as well can extract $\mathcal{I}$'s message b in poly-time.

If $\mathcal{A}$ corrupts Bob, $\mathcal{I}$ corrupts ideal-Bob in ID(OT) and obtains $(0, 0)$ or $(1, b)$ from the trusted host. $\mathcal{I}$ plays the part of Alice in BB-OT. If $\mathcal{I}$ got $(0, 0)$, it plays using $b = 0$ and resets $\mathcal{A}$ until it fails to receive the bit. If $\mathcal{I}$ got $(1, b)$, it plays using b and resets $\mathcal{A}$ until it does receive a bit. Ignoring detectable cheating, these require one expected reset. If $\mathcal{A}$ tries to cheat, then with equal probability, $\mathcal{I}$ resets $\mathcal{A}$ or accepts the cheating and sends quit to the trusted host, so that honest-ideal-Alice outputs abort with the same probability as honest-real-Alice. □

Some Remarks. One must be extremely careful in formalizing the notion of "proof of knowledge." It is quite easy to come up with a notion that seems fine in isolation but which fails when protocols are executed in parallel, unless some identification scheme is available.

An interesting flaw in Rabin's protocol comes to light when one applies our definitions to parallel executions. Although provably secure individually or in sequential composition, Rabin's protocol (even with corrections) is insecure when used in parallel, unless some identification scheme is available. One serious problem is that a proof that n is the product of two primes is not exactly a proof that Alice knows those two primes. A full description of the attack and of necessary conditions to ensure the resilience of *parallel* composition exceeds our space bounds here but is forthcoming [4].

It should be clear that deriving b from Alice's *view* is possible when she gives a satisfying proof of knowledge. We emphasize that deriving b from Alice's *conversation* should never be easy, or else Bob could do it himself. In this case, one must define input-committal/transparency with respect to views.

8.2 Yet Another Protocol

We mention an OT protocol developed by the author with Nicol So, which led the author to den Boer's protocol and inspired this paper. Like den Boer's protocol, this OT protocol does not require repeated generation of large Blum integers.[2]

1. Bob chooses $n = pq \leftarrow \text{BLUM}_k$, remembers (p, q), selects $a \leftarrow \mathbf{Z}_n^+$, and sends (n, a) to Alice. Bob proves in zero-knowledge that $n \in \text{BLUM}_k$.
2. Alice chooses $r \leftarrow \mathbf{Z}_n^*$ and random bit d, computes $z = [(-1)^d a]^b r^2 (\text{mod } n)$, and sends (d, z) to Bob. She proves that z was computed properly and that she knows (b, r).
3. Bob concludes $(d \oplus Q_n(a), [d \oplus Q_n(a)] \cdot Q_n(z))$, meaning: if $d = Q_n(a)$, Bob received nothing; else he received $Q_n(z) = b$.

Like the corrected BOER-OT protocol, this protocol requires that Alice demonstrate that she behaved and that she knows b.

9 Conclusion

We have found and fixed a flaw in a recently published protocol for Oblivious Transfer [11]. The flaw was found by applying a new, robust definition for security and fault-tolerance, which we call *resilience*. Resilience expresses the idea that one protocol is as secure as another if the results of attacks on the first are the same as those on the second. Using our definitions, we were able to identify the flaw quickly and even to give a direct fix for it.

A correction for the flawed step introduces a significant amount of communication, but in practical and computational terms it seems less costly than

[2] References to any other appearances of this or similar protocols would be greatly appreciated.

repeatedly generating large products of two primes. We have recently developed a non-generic proof of knowledge optimized specifically for this OT protocol [5], performing much better than using a generic proof of knowledge to correct the protocol. Thus, we believe that the computational advantages of den Boer's elegant OT protocol can be salvaged, and we can provide provably-secure OT at low cost.

Acknowledgements. Thanks to Nicol So for many discussions of OT. Thanks to Claude Crépeau for pointing out den Boer's paper.

References

1. D. Beaver. *Security, Fault Tolerance, and Communication Complexity in Distributed Systems.* PhD Thesis, Harvard University, Cambridge, 1990.
2. D. Beaver. "Formal Definitions for Secure Distributed Protocols." *Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography,* Princeton, NJ, October, 1989, J. Feigenbaum, M. Merritt (eds.).
3. D. Beaver. "Foundations of Secure Interactive Computing." *Proceedings of Crypto 1991,* 377–391.
4. D. Beaver. "The Security of Protocols Executed in Parallel." In preparation.
5. D. Beaver. "Efficient and Provably Secure Oblivious Transfer." Manuscript, 1992.
6. D. Beaver, S. Goldwasser. "Multiparty Computation with Faulty Majority." *Proceedings of the* 30th *FOCS,* IEEE, 1989, 468–473.
7. D. Beaver, S. Haber. "Cryptographic Protocols Provably Secure Against Dynamic Adversaries." Eurocrypt 1992.
8. D. Beaver, S. Micali, P. Rogaway. "The Round Complexity of Secure Protocols." *Proceedings of the* 22st *STOC,* ACM, 1990, 503–513.
9. C. Bennett, G. Brassard, C. Crépeau, M. Skubiszewska. "Practical Quantum Oblivious Transfer." *Proceedings of Crypto 1991,* 351–366.
10. M. Blum. "How to Exchange (Secret) Keys." *ACM Trans. Comput. Sys.* 1:2, May, 1983, 175–193.
11. B. den Boer. "Oblivious Transfer Protecting Secrecy." *Proc. of Eurocrypt 1991,* 31–45.
12. G. Brassard, D. Chaum, C. Crépeau. "Minimum Disclosure Proofs of Knowledge." *J. Comput. System Sci.* **37** (1988), 156–189.
13. G. Brassard, C. Crèpeau. "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond." *Proceedings of the* 27th *FOCS,* IEEE, 1986, 188–195.
14. S. Goldwasser, S. Micali, C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems." *SIAM J. Comput.* 18:1 (1989), 186–208.
15. J. Kilian. "Founding Cryptography on Oblivious Transfer." *Proceedings of the* 20th *STOC,* ACM, 1988, 20–29.
16. S. Micali, P. Rogaway. "Secure Computation." *Proc. of Crypto 1991,* page 9.8 [sic], and incomplete preliminary version distributed at conference.
17. M. Rabin. "How to Exchange Secrets by Oblivious Transfer." TR-81, Harvard, 1981.
18. A. Yao. "Protocols for Secure Computations." *Proceedings of the* 23rd *FOCS,* IEEE, 1982, 160–164.