Efficient Oblivious Transfer Protocols

Moni Naor* Benny Pinkas†

1 Introduction

Oblivious Transfer (OT) protocols allow one party, the sender, to transmit part of its inputs to another party, the chooser, in a manner that protects both of them: the sender is assured that the chooser does not receive more information than it is entitled, while the chooser is assured that the sender does not learn which part of the inputs it received. OT is used as a key component in many applications of cryptography. Its computational requirements are quite demanding and they are likely to be the bottleneck in many applications that invoke it.

1.1 Contributions. This paper presents several significant improvements to oblivious transfer (OT) protocols of strings, and in particular: (i) Improving the efficiency of applications which many invocations of oblivious transfer. (ii) Providing the first two-round OT protocol whose security analysis does not invoke the random oracle model. In more detail, the paper discusses the following issues:

- Oblivious transfer protocols with low amortized overhead, and a bandwidth/computation tradeoff for oblivious transfer, are discussed in Section 3. In particular we are able to break the "one exponentiation for each OT" barrier and decrease the number of exponentiations required by any desired factor, at the cost of increasing the communication overhead. For example, in Section 3.2.1 we show a specific example requiring a single exponentiation per *eight* invocations of 1-out-of-2 oblivious transfer (OT_1^2), using online communication of 400 bytes per OT_1^2. Another application (Section 3.1) is a OT_1^N protocol whose amortized overhead is just a single exponentiation (regardless of the size of the sender's input), and $O(N)$ communication (as in all previous OT_1^N protocols).

  The analysis of this construction relies on modeling hash functions as truly random functions (the random oracle assumption) as well as the computational Diffie-Hellman Assumption.

- A two round 1-out-N OT protocol which requires a fixed number of exponentiations on the chooser's side and O(N) exponentiations on the sender's side is described in Section 4. The analysis of the construction relies on the Decisional Diffie-Hellman Assumption. This is the first *two-round* OT protocol whose security analysis does *not* rely on the random oracle assumption.

1.2 Motivation. All known oblivious transfer protocols require "public key" operations (e.g. trapdoor functions), that are typically implemented using modular exponentiations, which are computationally intensive tasks[1]. The result of Impagliazzo and Rudich [21] implies that it is unlikely that oblivious transfer could be based on more efficient one-way functions or other private-key cryptographic primitives. Our working assumption is that such private-key functions are substantially more efficient than oblivious transfer, and, therefore, we wish to minimize the use of oblivious transfer.

Our work is motivated by applications which require two or more parties to perform *many* oblivious transfers of strings. Such applications include for example Yao's protocol for secure two-party computation [34, 20], and protocols for OT_1^N and for oblivious evaluation of polynomials [25], which use reductions to many invocations of OT_1^2. The computation overhead of oblivious transfer is often more demanding than its communication overhead. For example, the analysis of a recent privacy preserving protocol for computing the ID3 data mining algorithm [23] shows that the communication overhead can be done in a matter of seconds, while the computation takes several minutes. This is a good case for providing a bandwidth/computation tradeoff. As a specific example consider a recent construction of privacy-preserving protocols for auctions [26]. We examine a representative scenario of this protocol in Section 3.2.1 and demonstrate that using the tradeoff we can reduce the time that it takes to complete the protocol by a factor of 32 (e.g., for some reasonable choice of the parameters the time is reduced from half an hour to less than a minute). Another application is a protocol for 1-out-of-N oblivious transfer protocol which requires only a single exponentiation.

*Dept. of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot, Israel. Currently on sabbatical in IBM Almaden Research Lab and Stanford University. Email: naor@wisdom.weizmann.ac.il.

†STAR Lab, Intertrust Technologies. Most of this work was done while the author was at the Weizmann Institute of Science and the Hebrew University of Jerusalem, and was supported by an Eshkol grant of the Israeli Ministry of Science. Email: bpinkas@intertrust.com, benny@pinkas.net.

[1] For example, about 50 exponentiations of 1024 bit numbers can be computed in a single second, on a Pentium III PC.

The bandwidth/computation tradeoff is ideal in scenarios which require the use of *proxy oblivious transfer* – a three party variant of oblivious transfer defined in [26]. In this protocol the role of the chooser is divided between two parties: a *chooser* which chooses which of the sender inputs is learned, and a *proxy* which learns this item. The tradeoff is particularly useful if the party playing the chooser in the proxy oblivious transfer protocol has limited computational resources (for example, if it is a handheld device), since the computational overhead of the chooser is greatly reduced, and she can actually compute all the exponentiations in a preprocessing step. The increase in communication affects only the sender and the proxy.

Two round oblivious transfer with no random oracle assumptions: Previous oblivious transfer protocols of strings require at least three rounds of interaction, or depend on a security analysis which models some function as a *random oracle*. In section 4 we present the first two-round oblivious transfer protocol whose security does not depend on the random oracle assumption. Recent discussions demonstrate that modeling specific functions as random might not be justified [7, 8, 16], and therefore motivate our construction.

1.3 Related Work.

Reducing computational work via amortization has been done in several contexts in cryptography. The best results are in the area of signatures: it is possible to *verify* several modular exponential equations (and in particular signatures) with a number of exponentiations proportional to the security parameter (See [2, 24, 29]). The only example we are aware of where amortization reduces the cost to below an exponentiation per signature *generation* is Batch RSA [19].

In this paper we are concerned with 1-out-of-2 Oblivious Transfer (OT_1^2). This primitive was suggested by Even, Goldreich and Lempel [18], as a generalization of Rabin's "oblivious transfer" [28]. There have been many suggestions for implementing OTs; the one that is our starting point is by Bellare and Micali [3]. This protocol can be viewed as an instance of the EGL paradigm [18] for designing oblivious transfer protocols based on public-key cryptosystems, where the cryptosystem in question is El Gamal. The two-round Bellare-Micali protocol as presented in [3] is not known to be secure[2]. To the best of our knowledge no analysis of it using random oracles has appeared previously.

Regarding 1-out-of-N OT protocols, Naor and Pinkas [25] have shown how to implement them using $\log N$ parallel invocations of OT_1^2's protocols. This construction requires $\log N$ exponentiations, compared to the amortized overhead of a single exponentiation of the protocol of Section 3.1 (the communication overhead of both protocols is $O(N)$). We apply this algorithm to obtain some variants of the protocols presented in this paper (Remark 3.1 and Section 4.1).

Oblivious transfer is related to Private Information Retrieval (PIR). A discussion of this relation appears in Remark 3.4.

2 Preliminaries

2.1 Definitions of Security - Oblivious Transfer

In this paper we are concerned with 1-out-of-2 Oblivious Transfer (OT_1^2) where one party, the *sender*, has input composed of two strings (M_0, M_1), and the input of a second party, the *chooser*, is a bit σ. The chooser should learn M_σ and nothing regarding $M_{1-\sigma}$ while the sender should gain no information about σ. The definition of 1-out-of-N oblivious transfer is a straightforward generalization.

In order to define security we discuss separately protecting the sender and the chooser. Since we cannot offer both the sender and the chooser unconditional (information-theoretic) protection, we offer only computational protection for (at least) one of them. We start by defining the security for the protocols presented in Section 3, where the sender is protected computationally and the chooser information-theoretically:

The Chooser's Security: Given that under normal operation the sender gets no output from the protocol the definition of the chooser's security is simple: for any $\sigma, \tau \in \{0, 1\}$ and for any adversary $\mathcal{B}'$ executing the sender's part, the views that $\mathcal{B}'$ sees in case she tries to obtain M_σ and in case the chooser tries to obtain M_τ are statistically indistinguishable given M_0 and M_1.

The Sender's Security: We make the comparison to the *ideal implementation* using a trusted third party that receives M_0 and M_1 from the sender's inputs, and receives σ from the receiver, and tells the chooser M_σ. Our requirement is that for every distribution on the inputs (M_0, M_1) and any *adversarial* probabilistic polynomial-time machine $\mathcal{A}$ substituting the chooser, there exists a simulator – a probabilistic polynomial-time machine $\mathcal{A}'$ – that gets to play the chooser's role in the ideal model and receives the same a priori information about M_0 and M_1 as $\mathcal{A}$, such that the outputs of $\mathcal{A}$ and $\mathcal{A}'$ are computationally indistinguishable to a polynomial time distinguisher that is given M_0 and M_1.

Since we are interested in simultaneous executions of *many* 1-out-of-2 OT protocols we should clarify the security in this case. If n simultaneous protocols are executed, then the chooser's security is that for any $\vec{\sigma}, \vec{\tau} \in \{0, 1\}^n$, the distribution the sender sees when the input to the chooser is $\vec{\sigma}$, or is $\vec{\tau}$, is statistically indistinguishable. As for the sender's security, we should take into account the fact that the adversary controls a subset of the choosers. The definition requires that for any distribution on the inputs $\vec{M}_0$ and $\vec{M}_1$, any subset $S \subset \{1, \ldots n\}$, and any adversarial (probabilistic polynomial time) machine $\mathcal{A}$ that controls the choosers

[2] However if the two strings of the sender are random, then the protocol does not leak both of them. There are also provable alternatives using more rounds for a proof of knowledge

in a subset S of the protocols, there exists a probabilistic polynomial-time machine $\mathcal{A}'$ that gets to play the chooser's role for the subset S in the ideal model and receives the same a priori information about $\vec{M}_0$ and $\vec{M}_1$ as $\mathcal{A}$, such that the outputs of $\mathcal{A}$ and $\mathcal{A}'$ are computationally indistinguishable to a polynomial time distinguisher that is given $\vec{M}_0$ and $\vec{M}_1$.

2.1.1 Definitions for the construction which does not rely on random oracles

In contrast to the above definition, the protocols in Section 4 offer the sender information-theoretic security and the chooser only computational security, as defined below.

The Chooser's Security: For any $\sigma, \tau \in \{0,1\}$ and for any *probabilistic polynomial time adversary* $\mathcal{B}'$ executing the sender's part, the views that $\mathcal{B}'$ sees in case the chooser tries to obtain M_σ and in case the chooser tries to obtain M_τ are *computationally* indistinguishable given M_0 and M_1.

The Sender's Security: We make the comparison to the *ideal implementation*, using a trusted third party that receives M_0 and M_1 from the sender's inputs, and receives σ from the receiver, and tells the chooser M_σ. Our requirement is that for every distribution on the inputs (M_0, M_1) and any *adversarial* (not necessarily polynomial-time) $\mathcal{A}$ substituting the chooser, there exists a simulator – again not necessarily polynomial-time - $\mathcal{A}'$ that gets to play the chooser's role in the ideal model and the same a priori information about M_0 and M_1 as $\mathcal{A}$, such that the outputs of $\mathcal{A}$ and $\mathcal{A}'$ are statistically indistinguishable given M_0 and M_1.

2.1.2 Other issues

The above definitions do not address the issue of whether the sender is committed to his input prior to and independently of the chooser's inputs, i.e. the definition does not disallow the possibility that the values $\{M\}_i$ might depend on σ'_j (this can happen for instance in Protocol 3.2, see Remark 3.3). Although this requirement is significant, we do not require it since it seems that it is best to handle this issue at the level of the application above the OT, as can be done in the auctions architecture of [26]. Similarly, another issue not handled is whether $\mathcal{A}$ "knows" which input it has chosen, i.e. whether σ (for which $\mathcal{A}$ learns M_σ) is extractable. The major help of the random oracles is for this extraction to be possible. Once σ has been extracted the simulation is straightforward.

2.2 Assumptions and Models

The analysis of the security of the constructions in this paper is based on the Computational Diffie-Hellman Assumption, the Decisional Diffie-Hellman Assumption and the random oracle model.

The Diffie-Hellman assumptions: We assume that we have some probabilistic method to generate a group Z_q and a generator g. The *Computational* version of the assumption states that any probabilistic polynomial time machine that is given random $g^a, g^b \in Z_q$ has only negligible probability to correctly compute g^{ab}, where the probability is over the choice of the group and g, choices of g^a and g^b and the internal coin-flips of the machine. Various boosting algorithms are known for this problem [31]. In particular, the assumption implies that extracting discrete logarithms is hard.

The *Decisional* version of the Diffie-Hellman assumption is that it is difficult to differentiate between (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) for randomly chosen a, b and c. For this version a reduction from the worst case hardness to the average case hardness is known [5, 33, 27].

The Random Oracle model: Some of the constructions of this paper (Section 3) use a function H that is modeled *in the analysis* as a random oracle, i.e. chosen as a truly random function available to any participant (of course, in the application itself H is implemented as a hash function, such as SHA). The proof of security is based on the fact that the adversary can evaluate in limited time only a small number of the values of the function.

The major way we utilize the random oracle is for allowing extraction (by the simulator) without any interaction. I.e. the simulator $\mathcal{A}'$ monitors the queries to H by the adversary $\mathcal{A}$ and answers them in an appropriate manner. The security is measured in terms of the number of queries to H.

While the random oracle model is often used in the analysis of many cryptographic protocols (e.g. [4, 32]), it best not to assume this behavior from the concrete functions actually used in the construction. (See [7, 8, 16] for recent discussions on the subject.) We manage to achieve this in Section 4: we show a two round 1-out-N protocol that requires two exponentiations on the chooser side (for any N) and $O(N)$ exponentiations from sender's side. It is an interesting open problem whether it is possible to get efficient constructions on the sender's side as well (without relying on random oracles assumptions).

Efficiency of operations: the operation we consider the most expensive is modular exponentiation; in contrast private-key encryptions and calls to the function H are considered cheaper. We minimize the number of exponentiations, but at the same time we do increase the number of calls to the other primitives significantly.

2.3 The basic oblivious transfer protocol

Our starting point is the Bellare-Micali [3] protocol for oblivious transfer, which we have amended with random oracles. The revised construction is described below.

PROTOCOL 2.1. (OBLIVIOUS TRANSFER USING A RANDOM ORACLE)

Input: *The chooser's input is* $\sigma \in \{0,1\}$, *and the sender's input is two strings* M_0, M_1.

Output: *The chooser's output is* M_σ.

Preliminaries: *The protocol operates over a group* Z_q *of prime order. More specifically,* G_q *can be a sub-*

group of order q of Z_p^*, where p is prime and $q|p-1$. Let g be a generator group, for which the computational Diffie-Hellman assumption holds. The protocol uses a function H which is assumed to be a random oracle.

Initialization: *The sender chooses a random element $C \in Z_q$ and publishes it. (It is only important that the* chooser *will not know the discrete logarithm of C to the base g. It is irrelevant whether the sender knows the discrete log of C.)*

Protocol:

- *The chooser picks a random $1 \leq k \leq q$, sets public keys $PK_\sigma = g^k$ and $PK_{1-\sigma} = C/PK_\sigma$, and sends PK_0 to the sender.*
- *The sender computes $PK_1 = C/PK_0$ and chooses random $r_0, r_1 \in Z_q$. It encrypts M_0 by $E_0 = \langle g^{r_0}, H(PK_0^{r_0}) \oplus M_0 \rangle$, and encrypts M_1 by $E_1 = \langle g^{r_1}, H(PK_1^{r_1}) \oplus M_1 \rangle$, and sends the encryptions E_0, E_1 to the chooser.*
- *The chooser computes $H((g^{r_\sigma})^k) = H(PK_\sigma^{r_\sigma})$ and uses it to decrypt M_σ.*

Overhead: This protocol requires the sender to compute four exponentiations (two of them can be pre-computed before the protocol begins). The chooser should compute two exponentiations (one of which can be pre-computed).

When discussing the communication overhead it is important to distinguish between the length of the input elements M_0, M_1, and the length of group elements (which can typically be 768 or 1024 bits long). The communication from the chooser to the sender is composed of one group element, and the communication from the sender to the chooser is composed of two group elements and two elements in the size of the inputs.

Security: (we only give an intuitive discussion, see more details in further protocols). The *chooser's privacy* is preserved since the value that she sends to the sender, PK_0, is uniformly random (and independent of σ). As for *the sender's security*, the chooser cannot know the discrete logarithms of both PK_0 and PK_1, since this would reveal to her the discrete logarithm of C. The Diffie-Hellman assumption implies that she cannot compute both $(PK_0)^{r_0}$ and $(PK_1)^{r_1}$. Together with the random oracle assumption this ensures that she cannot distinguish either $H((PK_0)^{r_0})$ or $H((PK_1)^{r_1})$ from random.

3 OT with Low Amortized Overhead and Bandwidth/Computation Tradeoff

The basic oblivious transfer protocol (protocol 2.1) requires the sender to compute four exponentiations. This section describes two methods for reducing this overhead: (1) A 1-out-of-N oblivious transfer construction with low amortized overhead, in which the sender and chooser compute only a single exponentiation per transfer, while the communication overhead is not affected. (2) A tradeoff between the bandwidth and the communication requirements of 1-out-of-2 oblivious transfer protocols.

A single 1-out-of-2 oblivious transfer. As a warmup, consider protocol 2.1 with the modification that the sender uses the same random value r for both encryptions. Namely, $r = r_0 = r_1$. Since it holds that $PK_0 \cdot PK_1 = C$, it also holds that $(PK_0)^r \cdot (PK_1)^r = C^r$. Therefore, the operation of the sender can be defined as follows:

- **Initialization:** Choose a random r, compute C^r and g^r.
- **Transfer:** After receiving PK_0, compute $(PK_0)^r$, and $(PK_1)^r = C^r/(PK_0)^r$. Send g^r and the two encryptions, $H((PK_0)^r, 0) \oplus M_0$ and $H((PK_1)^r, 1) \oplus M_1$, to the chooser.

The *online* overhead of the sender is reduced to a single exponentiation, while the precomputation overhead is two exponentiations, as before.

Security:

The *chooser's privacy* is preserved, since for any C the distribution of PK_0 is the same whether it was chosen directly at random or as C/PK_1 for random PK_1.

As for the *sender's security*, consider an adversary $\mathcal{A}$. We describe the simulator $\mathcal{A}'$ operating in the *ideal model* and achieving results which are indistinguishable from those of the adversary $\mathcal{A}$. The simulator simulates both the sender (at least externally, without knowing M_0 and M_1) and $\mathcal{A}$. It chooses C and r and gives $\mathcal{A}$ the value C. When $\mathcal{A}$ sends PK_0 the simulator selects two random strings α_0 and α_1 (of the appropriate length) and sends them and g^r in response (as if they were the sender's answers). It then monitors all of $\mathcal{A}$ queries to H. All queries that do not contain $(PK_0)^r$ or $(PK_1)^r$ are answered at random. If $\mathcal{A}$ asks H about (PK_σ^r, σ), then $\mathcal{A}'$ asks the trusted party in the ideal model to receive M_σ. It then sets $H(PK_\sigma^r, \sigma)$ to be $\alpha_\sigma \oplus M_\sigma$. Then whatever $\mathcal{A}$ outputs so does $\mathcal{A}'$ (in particular, they have the same transcript).

It is easy to verify that in its interaction with $\mathcal{A}'$, $\mathcal{A}$ witnesses exactly the same distribution as it would under normal operation (and hence the output of the simulated one cannot be distinguished from the real one) - except for when $\mathcal{A}$ queries after $H(PK_\sigma^r, \sigma)$ or $H(PK_{1-\sigma}^r, 1-\sigma)$ (since $\mathcal{A}'$ cannot supply it with $M_{1-\sigma}$). However, if this occurs with probability ϵ then we can use $\mathcal{A}$ in order to break the computational Diffie-Hellman problem in time proportional to ϵ/m^2 where m is the number of queries to the random oracle. Given g^a and g^b, fix $C = g^a$ and $g^r = g^b$. Perform the previous simulation and guess that queries i and j correspond to PK_0^r and PK_1^r. Their product is $C^r = g^{ab}$. The probability of success is at least ϵ/m^2. (Using the boosting algorithm in [31] one can get an algorithm that is almost always right, at the expense of increased time.)

To conclude, if ϵ (which is an upper bound on the difference between the two distributions) is non-negligible, then we have an efficient algorithm breaking the CDH.

Note that adding the index σ as a suffix to PK_σ^r in the call for H assures that the answers for PK_0 and PK_1 are independent, even if PK_0 is chosen (maliciously) to be equal to PK_1 by setting it to be $\sqrt{C}$. (If the protocol should be run several times with the same constant C, then the input to H should include a random element, R, as explained for protocol 3.1.

It is also worth noting that it is possible to reduce the above dependency on m (the number of oracle queries) from m^2 to m by assuming the hardness of the Decisional Diffie-Hellman problem. Given a candidate y for g^{ab}, check and see whether any two calls to H of the form $(x_0, 0)$ and $(x_1, 1)$ are such that $x_0 \cdot x_1 = y$. This can be done in linear time by computing for all x_0's the value y/x_0 and checking for intersection with the x_1's.

3.1 OT_1^N with amortized complexity of a single exponentiation

3.1 OT_1^N with amortized complexity of a single exponentiation. The idea is to use the *same* value g^r for *all* transfers between the sender and chooser, and expand from 1-out-2 OT to 1-out-N OT by using N public-keys, so that the chooser can know the corresponding secret-key of at most one of them. This is achieved using $N-1$ public values. The computational overhead of the sender is consequently reduced to a single exponentiation per transfer regardless of the number of inputs.

PROTOCOL 3.1. (MANY SIMULTANEOUS 1-OUT-OF-N OT'S WITH THE SAME g^r)

Initialization: *The sender chooses $N-1$ random constants $C_1, C_2, \ldots, C_{N-1}$ (it will hold that $PK_0 \cdot PK_i = C_i$). It also chooses a random r and computes g^r. The values $C_1, \ldots, C_{N-1}$ and g^r are sent to the chooser and play the role of the public-key of the sender. The same values will be used for all transfers.*

The sender precomputes for every $1 \leq i \leq N-1$ the value $(C_i)^r$.

Transfer: *The sender's input is $M_0, M_1, \ldots M_{N-1}$. The chooser's input is $\sigma \in \{0, \ldots N-1\}$ (she should learn M_σ).*

- *The chooser selects a random k and sets $PK_\sigma = g^k$. If $\sigma \neq 0$ she computes $PK_0 = C_\sigma / PK_\sigma$. She sends PK_0 to the sender and can already compute a decryption key $(g^r)^k = (PK_\sigma)^r$.*
- *The sender computes $(PK_0)^r$ and then for every $1 \leq i \leq N-1$ computes (without doing any additional exponentiations)*

$$(PK_i)^r = (C_i)^r / (PK_0)^r.$$

The sender chooses a random string[3] R. He then encrypts each M_i by computing $H((PK_i)^r, R, i) \oplus M_i$, and sends these encryptions and R to the chooser.

- *The chooser uses $H((PK_\sigma)^r, R, \sigma)$ to decrypt M_σ.*

Overhead: The initialization phase consists of N exponentiations on the sender's side. Following the initialization (that can be amortized on all subsequent transfers), in each transfer the sender performs only a *single* exponentiation per transfer, plus $N-1$ multiplications and N invocations of H. The chooser can precompute the decryption key before receiving the encrypted elements from the sender. The communication overhead from the sender to the chooser corresponds to the size of the N strings.

REMARK 3.1. *Suppose that the cost of N exponentiations for the initialization, and a public key consisting of N values, are considered too high. We can combine Protocol 3.1 with the* 1*-out-N protocol of [25] in order to design the following: let K be a parameter such that a public-key of size K and initialization of K exponentiations is acceptable. Then if we express the index i in base K (i.e. as a vector of length $\log N/\log K$ of numbers in $\{0, \ldots K-1\}$) we can perform the [25] algorithm using $\log N/\log K$ invocations of* 1*-out-of-K OT (instead of OT_1^2). The* 1*-out-of-K OT is performed using Protocol 3.1 (with a single exponentiation). The result is a protocol that requires $\log N/\log K$ exponentiations from the sender and $2\log N/\log K$ exponentiations from the chooser, for every transfer. Note that setting $K = \log N/\log\log N$ implies that without* any *amortization of the initialization cost the complexity is reduced to $2\log N/\log\log N$, including the initialization (compared to $\log N$ of the plain [25] protocol).*

Security: The Chooser's security is, as before, information theoretic – the distribution on PK_0 is independent of the values of $C_0, \ldots C_{N-1}$ and σ. The sender's security is based on the same argument as the sender's security for the single oblivious transfer which uses a single g^r. The first point to note is that if the chooser knows for more than a single public key (say for PK_{i_1} and PK_{i_2}) the values $(PK_{i_1})^r$ and $(PK_{i_2})^r$, then $(PK_{i_1})^r/(PK_{i_2})^r = (C_{i_1}/C_{i_2})^r$. This in turn can be used to deduce the Diffie-Hellman value of a random C and random g^r: given inputs $A = g^a$ and $B = g^b$ (where the goal is to compute g^{ab}), simulate $\mathcal{A}$ by generating the constants $C_i = A^{r_i}$ for a random r_i and $g^r = g^b$. If $\mathcal{A}$ is successful on i_1 and i_2, then $(C_{i_1}/C_{i_2})^r = g^{abr_{i_1}}/g^{abr_{i_2}}$. Raising the latter to $1/(r_{i_1} - r_{i_2})$ yields g^{ab}.

[3] The string should be long enough so that there are no two invocations of the protocol in which R obtains the same value. Namely, the length should be larger than $2\log(n)$ bits, where n is the number of invocations of the protocol. Alternatively, the sender could use a counter to set the value of R, and then the length could be reduced to $\log n$.

Since we envision using Protocol 3.1 many times with the same public-key $C_1, C_2, \ldots C_{N-1}$ and g^r, we should describe the actions of the simulator $\mathcal{A}'$ against an adversary that controls some subset of the users. Therefore, the input of H contains a random element R which ensures that the inputs in different invocations of the protocol will be different. As before the goal is to extract the values in which $\mathcal{A}$ is interested (and obtain them in the ideal model). $\mathcal{A}'$ chooses $C_1, C_2, \ldots C_{L-1}$ at random as well as g^r. When $\mathcal{A}$ sends in the message of the tth user $\mathcal{A}'$ responds with random $(\alpha_0, \alpha_2, \ldots \alpha_{N-1})$ for the encryptions of the M_i's and a random R_t. Whenever $\mathcal{A}$ queries H on a point (x, R', j), $\mathcal{A}'$ checks whether $R' = R_t$ for some t. If not, a random answer is given. If $R' = R_t$ and $x = (PK_j)^r$ then $\mathcal{A}'$ asks in the ideal model for the values corresponding to j and obtain M_j. It then has to set H appropriately - set $H((PK_j)^r, R_t, j) = \alpha_j \oplus M_j$. The only difference between the distribution the simulated $\mathcal{A}$ sees and the real one occurs if (i) $\mathcal{A}$ queries on another $PK^r_{j'}$ – which we have argued can be used for breaking the CDH, so by assumption it happens with negligible probability. (ii) One of the $H(x, R_t, j)$ is queried in advance - but this happens with low probability, at most (number of queries / size of group), for each query.

REMARK 3.2. *The protocol would have been insecure if it were using plain El Gamal encryption instead of a random oracle H: Suppose that the transfer was of the form $\langle (PK_{i,0})^r \cdot m_{i,0}, (PK_{i,1})^r \cdot m_{i,1} \rangle$. Then if the chooser has prior knowledge of one of the transferred elements (say, $m_{i,0}$), she could compute the corresponding encryption key $(PK_{i,0})^r$ from the encrypted message even if she knows the private key of the other element. Therefore she can compute both $(PK_{i,0})^r$ and $(PK_{i,1})^r$, multiply them and get $(C_i)^r$. This value enables her to decrypt both elements in every other transfer.*

3.2 Bandwidth/Computation tradeoff for OT^2_1.

The computation overhead of OT^2_1 can be reduced by performing ℓ OT^2_1's at the same transfer using protocol 3.1. The idea is to translate ℓ calls to OT^2_1 into a *single* 1-out-L OT for $L = 2^\ell$. (This can be seen as the opposite of what [25] do: they translate a 1-out-L OT into ℓ OT^2_1's.) While this reduces the computation it increases the communication, so we obtain a computation/communication tradeoff.

Let ℓ be a parameter which denotes the number of strings that are transferred in each batch, i.e. if the number of OT^2_1's is large, partition them into blocks of ℓ. Consider one block, in which the sender should transfer to the chooser one string out of each of the pairs $\{(m_{i,0}, m_{i,1})\}^\ell_{i=1}$. Our approach is to perform all ℓ transfers simultaneously. The sender defines $L = 2^\ell$ strings, $M_0, \ldots, M_{L-1}$, corresponding to all combinations of ℓ strings, one from each pair. Namely, $M_j = \langle m_{1,j_1}, m_{2,j_2}, \ldots, m_{\ell,j_\ell} \rangle$, where j_i is the ith bit of j, $0 \le j \le L-1$. Instead of engaging in ℓ OT^2_1's, the parties can engage in a single 1-out-of-L oblivious transfer of one of these strings.

PROTOCOL 3.2. (SIMULTANEOUS TRANSFER OF ℓ STRINGS) *The parties run protocol 3.1 for 1-out-of-L oblivious transfer. The sender's input contains the L strings $M_0, \ldots, M_{L-1}$. The chooser's input is $\overrightarrow{\sigma} = \sigma_1, \ldots, \sigma_\ell$, where σ_i is her choice in the ith oblivious transfer.*

The protocol obtains a general tradeoff between reducing the computational overhead of the sender, and exponentially increasing the communication overhead (although the exponential blowup in the communication seems very limiting, the optimal tradeoff seems to happen for a rather large value of ℓ, as is described in Section 3.2.1).

THEOREM 3.1. (BANDWIDTH/COMPUTATION TRADEOFF) *In protocol 3.2 the amortized computational overhead* per transfer *(OT^2_1) is $1/\ell$ exponentiations, for both the sender and the chooser. In addition the sender performs $2^\ell/\ell$ multiplications. The offline communication overhead is 2^ℓ keys per transfer, while the online communication overhead is $2^\ell/\ell$ keys per transfer.*

Computation overhead: During the transfer phase the sender has to perform a *single* exponentiation and $L = 2^\ell$ multiplications. The chooser performs one exponentiation when she sends the request and one when she receives it (note that both of them can be done offline).

The initialization phase requires L exponentiations from the sender, but these are amortized over all the blocks ever sent by the sender.

Communication overhead: It may seem as if the communication is $\ell \cdot L$ times the size of an input element, since there are L messages M_i of length ℓ each. However a finer analysis is called for: First we distinguish between group elements (which might be long, say 1000 bits) and private-keys which can be as short as 100 bits. Furthermore, another distinction is between *online* and *offline* communication, where the latter refers to messages that can be sent *independently* of the inputs. Therefore we can think of them as being sent in a preprocessing phase (say when the communication network is idle, or stored in a DVD).

From the chooser to the sender only a *single* group element is sent. The *online* communication from the sender to the chooser can be reduced from $O(\ell \cdot L)$ to $O(L)$ by encrypting each m_i with a different key, and running the oblivious transfer with the keys as inputs. That is , the following protocol is run:

1. The sender chooses for each $1 \le i \le \ell$ and $\sigma \in \{0, 1\}$ a random key $k_{i,\sigma}$ (a total of 2ℓ keys). For each $1 \le j \le L$ let $M'_j = \langle k_{1,j_1}, k_{2,j_2}, \ldots, k_{\ell,j_\ell} \rangle$ be the concatenation of the keys of the values transferred when j is chosen.

2. For each $1 \le j \le L$ the sender chooses a random key K_j and encrypts M'_j with $H(K_j, R)$, i.e. gen-

erates $M'_j \oplus H(K_j, R)$. These encryptions are sent offline.

3. In the transfer phase of the protocol the sender first sends, for all $1 \leq i \leq \ell$ and $\sigma \in \{0,1\}$, the values $E_{k_{i,\sigma}}(m_{i,\sigma})$. The values that are actually transferred in the oblivious transfer are $(K_1, K_2, \ldots K_L)$. Each K_j is encrypted with $H((PK_j)^r, R, j)$.

4. The chooser uses the key K_j to decrypt the corresponding string M'_j. It then uses the keys $k_{i,j}$ in M'_j to decrypt the elements $m_{i,j}$.

Offline communication: The total length of the *offline communication* in step 2 is $L\ell|k_{i,\sigma}|$ bits. The length of the keys $|k_{i,\sigma}|$ can be set to be 100 bits, and does not increase if the inputs are longer (however, if the inputs $m_{i,\sigma}$ are shorter, they can be encrypted by xoring them with the corresponding keys $k_{i,\sigma}$, and then the keys can be as short as the inputs).

Online communication: In the transfer step itself the sender sends a message of length $L|K_j| + 2\ell|m_{i,\sigma}|$ bits to the receiver (the keys K_j can also be 100 bits long).

Note that in comparison to the alternative of running the oblivious transfer protocol ℓ times independently, there is an $L = 2^\ell$ multiplicative increase in the offline communication, and an L/ℓ increase in the online communication. The increase in the communication compared to the basic Bellare-Micali protocol (Protocol 2.1) is smaller, since all messages in that protocol include elements in the group in which the Diffie-Hellman assumption holds.

Security: The chooser is protected information theoretically, as before. As for the sender's security, the major point is preventing the chooser from learning more than a single string in each transfer. We should describe the actions of the simulator $\mathcal{A}'$ against an adversary that controls some subset of the choosers. As before the goal is to extract the values in which $\mathcal{A}$ is interested and obtain them in the ideal model. $\mathcal{A}'$ chooses $C_1, C_2, \ldots C_{L-1}$ at random as well as g^r. In the preprocessing phase $\mathcal{A}'$ generates for each chooser t a random vector $\{\vec{\beta_j} = \langle \beta_{1,j_1}, \beta_{2,j_2}, \ldots, \beta_{\ell,j_\ell}\rangle | 1 \leq j \leq L\}$. When $\mathcal{A}$ sends in the message of the tth chooser $\mathcal{A}'$ responds with random $(\gamma_{1,0}, \gamma_{1,1}, \gamma_{2,0}, \gamma_{2,1}, \ldots \gamma_{\ell,1})$ for the encrypted values, random $(\alpha_1, \alpha_2, \ldots \alpha_L)$ for the keys and random R_t. Whenever $\mathcal{A}$ queries H on a point (x, R', j), $\mathcal{A}'$ checks whether $R' = R_t$ for some t. If not, a random answer is given. If $R' = R_t$ and $x = (PK_j)^r$ then $\mathcal{A}'$ asks in the ideal model for the values corresponding to $(j_1, j_2, \ldots j_\ell)$ and obtains M_j. It then has to set H appropriately - choose random key K_j and set $H(PK_j^r, R_t, j) = \alpha_j \oplus K_j$. Set $H(K_j, R_t) = \vec{\beta_j} \oplus \langle \gamma_{1,j_1}, \gamma_{2,j_2}, \ldots \gamma_{\ell,j_\ell}\rangle \oplus M_j$. The only difference between the distribution the simulated $\mathcal{A}$ sees and the real one occurs if (i) $\mathcal{A}$ queries on another $(PK_{j'})^r$ – which we have argued can be used for breaking the CDH, so by assumption it happens with negligible probability. (ii) One of the $H(K_j, R_t)$ is queried in advance - this happens with low probability.

REMARK 3.3. *The protocol allows a malicious sender a new possible abuse not handled by our definition of security in Section 2.1: he can encode $k_{i,\sigma}$ inconsistently in the different strings that should include $k_{i,\sigma}$. Therefore the chooser gets different values of $m_{i,\sigma}$ depending on (i', σ'). Fortunately, there is a simple solution in a framework like [26]: add periodic checks, by requesting the sender to open all hidden strings and show consistency.*

REMARK 3.4. *It is interesting to note that the approach we have taken in investigating the bandwidth/computation tradeoff is somewhat opposite to the approach of Private Information Retrieval (PIR) (see e.g., [22, 6]). The goal of PIR protocols is to minimize the communication overhead, and this requires (in the case of single server solutions) to increase the computation overhead, up to computing an exponentiation for every bit of the database. Our goal, on the other hand, is to reduce the computation overhead of oblivious transfer protocols by increasing the communication overhead. As we describe below, this approach makes sense in many applications, since existing oblivious transfer protocols often have lesser utilization of bandwidth than of computation.*

3.2.1 Which tradeoff to choose?

The optimal balance between communication and computation depends on specific considerations for each case, depending on the available bandwidth and computational resources. A very realistic assumption is, however, that the goal is to minimize the *latency* of the oblivious transfer protocol. This latency is affected by both the communication and the computation times. Typical bandwidth between the chooser and the sender could be 1.5Mb/sec for a T1 line or 35Mb/sec for a T3 line. On the other hand, an exponentiation of a 1000 bit number takes about 20 msec on a state of the art computer Pentium III or Celeron at 500Mhz (see [13] for benchmark results). (This is where the exponent is 200 bits long.)

For a quick estimate of the optimal value of ℓ, consider the tasks of the chooser (the analysis for the sender is almost identical). Most of her overhead is incurred by computing a single exponentiation for every ℓ transfers, and receiving on-line 2^ℓ keys (say, $100 \cdot 2^\ell$ bits for 100 bit keys) from the sender (offline $2^\ell \ell$ keys should be sent). Optimal online performance is achieved when the same amount of time is invested in the computation and in the communication since in this case there are no idle times for either the processor or the network.

Suppose that the bandwidth is B bits per second, the keys are k bits long, and the processor is capable of performing E exponentiations per second. Then, considering only the online communication, the latency per OT_1^2 is $\max(\frac{2^\ell k}{\ell B}, \frac{1}{\ell E})$ (this is assuming pipelining of the communication and computation; otherwise the two quantities should be added .) Minimal latency is achieved when

$$\frac{2^\ell k}{B} = \frac{1}{E}. \quad \text{Namely,} \quad \ell = \log \frac{B}{kE}.$$

The optimal tradeoff is achieved (perhaps surprisingly) using a rather large ℓ. For example a server which can perform $E = 50$ exponentiations per second, and uses a T1 line (1.5Mbps), should set $\ell = 8$ to obtain optimal performance. Assuming that the server invests more in its resources, distributes the exponentiations between 5 processors (achieving $E = 250$), and uses a T3 line, $\ell = 10$ is optimal. Figure 1 describes the throughput, i.e. the number of OT_1^2's per second, that is achieved for different values of the bandwidth (when E and k are kept constant). Figure 2 describes the gain in speed as a function of ℓ (namely, the throughput divided by that of a protocol with no optimizations), for representative values of B, E and k.

An example. As a concrete example for the improvement achieved by the techniques presented here, and in particular by the bandwidth/communication tradeoff, consider the auction protocol of [26]. The main computational overhead of the protocol is the execution of many oblivious transfers. More accurately, for N bidders with m bit inputs, the protocol requires Nm oblivious transfers[4].

The overhead of the sender in a naive application of the basic protocol (protocol 2.1) is

Computing $4Nm$ exponentiations, and sending $2Nm \cdot (1024 + 100)$ bits.

(Assuming a 1024 bit modulus and 100 bit keys.) For an auction with $N = 1000$ bidders, and $m = 24$ bit bids, this results in 96,000 exponentiations, and a message of 6.7 Mbytes. Each receiver should compute $2m = 48$ exponentiations.

If protocol 3.2 with $\ell = 8$ is used, then the overhead of the sender is

Computing $Nm/8$ exponentiations, sending $\frac{Nm}{8} \cdot (1024 + 2^8 \cdot 100)$ bits online, and $\frac{Nm}{8} \cdot (2^8 \cdot 8 \cdot 100)$ bits offline.

For the same values of N and m, the sender computes only 3000 exponentiations (an improvement by a factor of 32). Each receiver computes 6 exponentiations (3 offline and 3 online). The sender sends an online message of length 10 Mbytes, and an offline message of length 76 Mbytes. Using a T1 communication line, and a computer which computes 50 exponentiations per second, both the computation and the online communication take about 60 seconds for all bidders (compared to approximately half an hour for the original protocol).

[4] More accurately, the protocol uses a proxy oblivious transfer protocol which involves three parties. However, the protocols we defined in this paper can be translated in a straightforward manner to the proxy OT setting.

4 How to avoid random oracles

This section discusses methods for designing protocols whose analysis does *not* rely on random oracles. The first issue is whether it is possible to design a two-round (this is called "non-interactive" by Bellare and Micali) OT protocol which does not rely on the random-oracle assumption. The only previous protocol that (almost[5]) fits the two-round requirement was recently proposed by Dwork and Naor [17] and is quite inefficient, as it relies on the general protocol for non-interactive zero-knowledge protocols for languages in NP. We note that independently of this work Aiello, Ishai and Reingold [1] suggested a protocol with a similar structure for a related problem.

The structure of the protocol is *chooser to sender* and then *sender to chooser*. Unlike the protocols of [3, 17] there is no need for the sender to have a public-key or public-value. It is based on the DDH assumption and offers information-theoretic protection for the sender and computational indistinguishability for the chooser (the opposite of the case in the Bellare-Micali scheme and the other schemes in this paper). The idea is for the chooser to create two encryption keys so that at most one of them is legitimate. It uses the following idea: if a key is badly formed, then the corresponding ciphertext has a random value. This is done similarly to the randomized self-reduction of the DDH problem (see [33] and [27]). The idea of randomizing bad ciphertexts was previously suggested by Canetti and Goldwasser [9] in order to design a threshold variant of the Cramer-Shoup cryptosystem [11] – there if a ciphertext is *not* valid, then it decrypts to a random value.

For the protocol we assume some El Gamal like method that is semantically secure. The simplest case is to assume that the messages are in the subgroup in which we operate. Otherwise we have to use hashing (probabilistic, not random-oracle type, see e.g. [27]).

PROTOCOL 4.1. (BASIC PROTOCOL)

1. *The chooser generates random g^a, g^b and c_0, c_1 such that $c_\sigma = ab$ and $c_{1-\sigma}$ is random. Chooser sends $x = g^a, y = g^b, z_0 = g^{c_0}, z_1 = g^{c_1}$.*
2. *The sender verifies that $z_0 \neq z_1$. It then generates random (r_0, s_0) and (r_1, s_1) and*
   - *(a) Computes $w_0 = x^{s_0} \cdot g^{r_0}$ and encrypts M_0 using the key $z_0^{s_0} \cdot y^{r_0}$. The value w_0 and the encryption are sent to the chooser.*
   - *(b) Computes $w_1 = x^{s_1} \cdot g^{r_1}$ and encrypts M_1 using the key $z_1^{s_1} \cdot y^{r_1}$. The value w_1 and the encryption are sent to the chooser.*
3. *The chooser computes $(w_\sigma)^b$ and decrypts M_σ.*

[5] That protocol requires either a public-string (not key) from the sender or relies on non-uniformity.

Overhead: The chooser is required to compute three exponentiations when preparing g^a, g^b and $g^{c_\sigma} = g^{ab}$ (as well as selecting $g^{c_{1-\sigma}}$ at random). Note that one of $\{g^a, g^b\}$ can remain fixed and there is no need to change it from transfer to transfer. Therefore the chooser can compute (offline) two exponentiations per transfer, and compute an additional exponentiation (online) when it receives the message from the sender. The sender has to compute two double exponentiations (of two different values) per transfer.

Security: The Chooser's security is based on the indistinguishability of c_σ and $c_{1-\sigma}$ which in turn is based on the DDH assumption. The sender's security is information-theoretic and is based on the following claim.

CLAIM 4.1. *If $z_\tau \neq g^{ab}$, then $(x_\tau^{s_\tau} \cdot g_\tau^{r_\tau}, z_\tau^{s_\tau} \cdot y_\tau^{r_\tau})$ is uniformly distributed.*

The proof follows from the randomized reduction of DDH of [27] and [33].

Therefore if $z_\tau \neq g^{ab}$ no information about m_τ is transferred. Since at most one of $\{z_0, z_1\}$ equals g^{ab}, the chooser's gets information on at most one of M_0 and M_1.

4.1 The 1-out-of-N Protocol.

It is straightforward to generalize the above scheme to 1-out-of-N OT. The interesting point is that it can done be without increasing the chooser's complexity. The point is that the chooser does not have to choose and send the g^{c_i}'s explicitly. Suppose that the chooser is interested in M_i. Then she chooses $c_i = a \cdot b$ and sets $z_i = g^{c_i}$. This in turns defines all the z_j's by setting $z_j = z_i \cdot g^{j-i}$ and in particular $z_0 = g^{c_i - i}$. Therefore what the chooser sends is g^a, g^b and $z_0 = g^{c_i}/g^i$. The sender on the other hand still has to perform all the work he performed previously.

Security: The sender's security is information theoretic, as before. As for the chooser's security, under the DDH assumption no probabilistic polynomial-time machine can distinguish for any fixed i between a triple of the form (g^a, g^b, g^{ab-i}) and a random triple. Therefore triples of the form (g^a, g^b, g^{ab-i}) and $(g^a, g^b, g^{ab-i'})$ are computationally indistinguishable.

Overhead: The chooser is required to compute three exponentiations when preparing g^a, g^b and g^{ab} (and as before one of $\{g^a, g^b\}$ can remain fixed). The sender has to compute $2N$ double exponentiations. As for communication, the chooser is sending a fixed number of elements and receives from the sender $2N$ elements. Hence the chooser performs roughly as much work as she does in the random oracle based OT_1^2 construction of Protocol 3.1.

REMARK 4.1. *Note that the protocol in [25] is more egalitarian in that each side performs $\log N$ exponentiations. As in Remark 3.1 we can combine the two approaches. Expressing the index i in base K (i.e. as a vector of length $\log N/\log K$ of numbers in $\{0, \ldots K-1\}$) we perform the [25] algorithm using 1-out-of-K OT (instead of 1-out-2); the 1-out-of-K OT is performed using the above protocol. The result is a protocol that requires $K \log N/\log K$ exponentiations from the sender and $2\log N/\log K$ exponentiations from the chooser.*

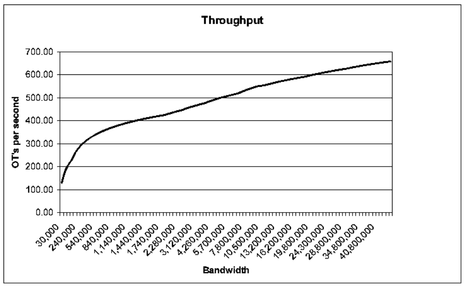

Figure 1: The throughput (number of OT_1^2 per second) as a function of the bandwidth, using the optimal ℓ, for $E = 50$ and keys of $k = 100$ bits.

4.2 Two Applications and a Caveat.

One scenario where this protocol is applicable is the architecture for performing auctions and mechanism design of [26]. The bidders should participate in many OT protocols as choosers. The method for reducing the computational load is very relevant there - it can save significant work on both the bidder and auctioneer side, but not for the auction issuer.

Another application is turning PIR into SPIR *without increasing the number of rounds* (A SPIR protocol also protects the owner of the data from leaking information about more than one entry.) In [25] it was suggested to construct SPIRs based on OT_1^2 without increasing the number of rounds compared to the OT_1^2 protocol, so one can either apply Protocol 4.1 or can do it directly using the OT_1^N protocol described in Section 4.1. Namely, the sender creates a PIR database for the $2N$ elements it sends in response to the chooser's query, and runs a PIR protocol for selecting the ith entry. Since in all known single-server PIR protocols [6, 22] the sender performs a huge number of operations anyway, the latter approach is most appropriate.

Finally, note that in the protocol we cannot rule out the possibility of *malleability* from the Sender's side: the information the sender transmits might be a function of the bit(s) σ the chooser is requesting (without the sender really knowing what he is sending). This is not relevant in case the sender is committed to its value prior to the protocol, as in the [26] scenario.

References

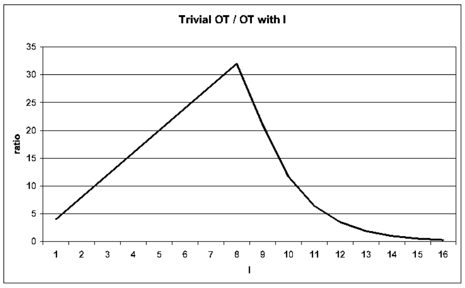

Figure 2: The gain in speed as a function of ℓ, for a bandwidth of $B = 1,500,000$Mbps, computation power of $E = 50$ exponentiations per second, and $k = 100$ bit keys.

[1] W. Aiello, Y. Ishai and O. Reingold, *Oblivious Commerce: How to Sell Digital Goods*, Manuscript, 2000.

[2] M. Bellare, J. Garay and T. Rabin. "Fast Batch Verification for Modular Exponentiation and Digital Signatures." Proc. Advances in Cryptology–Eurocrypt '98, LNCS (1403), Springer-Verlag, pp. 236-250, 1998.

[3] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications", *Proc. Adv. in Cryptology - Crypto '89*, Springer-Verlag LNCS 435 (1990), 547-557.

[4] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", *1st ACM Conference on Computer and Communications Security*, ACM Press, November 1993.

[5] D. Boneh, "The Decision Diffie-Hellman Problem", *Proc. of the Third Algorithmic Number Theory Symposium*, Springer-Verlag LNCS 1423 (1998), 48–63.

[6] C. Cachin, S. Micali and M. Stadler, *Computationally Private Information Retrieval With Polylogarithmic Communication*, Advances in Cryptology – Eurocrypt '99, LNCS 1592, Springer-Verlag, 1999.

[7] R. Canetti, *Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information*, Advances in Cryptology – Crypto '97, pp. 455-469.

[8] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, Proc. 30th ACM Symposium on Theory of Computing, 1998, pp. 209-218.

[9] R. Canetti and S. Goldwasser: *An Efficient Threshold Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attack*, Advances in Cryptology – EUROCRYPT '99, Springer-Verlag, 1999, pp. 90-106.

[10] R. Cramer, I. Damgard and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols", *Proc. Advances in Cryptology – Crypto '94*, Springer-Verlag LNCS 839 (1994), 174–187.

[11] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attacks", *Proc. Advances in Cryptology - Crypto '98*, Springer-Verlag LNCS 1462 (1998), 13–25.

[12] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung, *On Monotone Formula Closure of SZK*, Proc. of 35th IEEE Symposium on Foundations of Computer Science (FOCS '94), Santa Fe, New Mexico, USA, November 20-22, 1994, pp. 454-465.

[13] W. Dai, Crypto++ 3.1 Benchmarks, available at `http://www.eskimo.com/~weidai/benchmarks.html`

[14] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Tra. Inform. Theory, 22 (6), 644-654, 1976.

[15] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography", *Proc. 23th ACM Symp. on Theory of Computing*, 1991. Full version: to appear Siam J. on Computing. Available at `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`

[16] C. Dwork, M. Naor, O. Reingold and L. Stockmeyer, Magic Functions, FOCS'99, pp. 523-534.

[17] C. Dwork, M. Naor, *Zaps and their applications*, 41st IEEE Symp. on Foundations of Comp. Science, 2000.

[18] S. Even, O. Goldreich and A. Lempel, "A Randomized Protocol for Signing Contracts", *Communications of the ACM* **28**, 1985, pp. 637–647.

[19] A. Fiat, *Batch RSA*, J. of Crypt. 10(2): 75-88 (1997).

[20] O. Goldreich, M. Micali and A. Wigderson, "How to play any mental game", *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 218–229.

[21] R. Impagliazzo and S. Rudich, "Limits on the Provable Consequences of One-Way Permutations", *20th ACM Symp. on the Theory of Computing*, 1989, 44–61.

[22] E. Kushilevitz and R. Ostrovsky, *Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval*, 38th FOCS, pp. 364-373, 1997.

[23] Y. Lindell and B. Pinkas, *Privacy Preseving Data Mining*, *Proc. Advances in Cryptology - Crypto '2000*, Springer-Verlag LNCS 1880, pp. 36–54, 2000.

[24] J. Naor and M. Naor, *Small-Bias Probability Spaces: Efficient Constructions and Applications*, SIAM J. Comput. 22(4): 838-856 (1993).

[25] M. Naor and B. Pinkas, *Oblivious Transfer and Polynomial Evaluation*, Proc. 31st Symp. on Theory of Computer Science (STOC), pp. 245-254, May 1-4, 1999.

[26] M. Naor, B. Pinkas and R. Sumner, *Privacy Preserving Auctions and Mechanism Design*, Proc. of the 1st ACM conference on Electronic Commerce, November 1999.

[27] M. Naor and O. Reingold, Number-Theoretic constructions of efficient pseudo-random functions, *38th IEEE Symp. on Foundations of Comp. Sci.*, 1997, 458-467.

[28] M. O. Rabin, "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, 1981.

[29] K. Sako, J. Kilian, Secure Voting Using Partially Compatible Homomorphisms. CRYPTO 1994: 411-424

[30] C. P. Schnorr, "Efficient Signature Generation by Smart Cards", *J. of Crypt.*, 4(3), pp. 161-174, 1991.

[31] V. Shoup *Lower bounds for discrete logarithms and related problems*, in Proc. Eurocrypt '97, Springer Verlag LNCS 1233, pp. 256-266, 1997.

[32] V. Shoup and R. Gennaro , *Securing threshold cryptosystems against chosen ciphertext attack*, Proc. Advances in Cryptology - Eurocrypt'98, Springer-Verlag LNCS 1403, 1998, pp. 1–16.

[33] M. Stadler, Publicly verifiable secret sharing, *Proc. Advances in Cryptology - EUROCRYPT '96*, LNCS, vol. 1070, Springer, 1996, pp. 190-199.

[34] A.C. Yao, "How to generate and exchange secrets", *Proc. of the 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 162–167.