The IPS Compiler: Optimizations, Variants and Concrete Efficiency*

Yehuda Lindell, Eli Oxman, and Benny Pinkas

Dept. of Computer Science, Bar Ilan University, Ramat Gan, Israel
lindell@cs.biu.ac.il, eli.oxman@gmail.com, benny@pinkas.net

Abstract. In recent work, Ishai, Prabhakaran and Sahai (CRYPTO 2008) presented a new compiler (hereafter the IPS compiler) for constructing protocols that are secure in the presence of malicious adversaries without an honest majority, from protocols that are only secure in the presence of semi-honest adversaries. The IPS compiler has many important properties: it provides a radically different way of obtaining security in the presence of malicious adversaries with no honest majority, it is black-box in the underlying semi-honest protocol, and it has excellent asymptotic efficiency.

In this paper, we study the IPS compiler from a number of different angles. We present an efficiency improvement of the "watchlist setup phase" of the compiler that also facilitates a simpler and tighter analysis of the cheating probability. In addition, we present a conceptually simpler variant that uses protocols that are secure in the presence of covert adversaries as its basic building block. This variant can be used to achieve more efficient asymptotic security, as we show regarding black-box constructions of malicious oblivious transfer from semi-honest oblivious transfer. Finally, we analyze the IPS compiler from a *concrete efficiency* perspective and demonstrate that in some cases it can be competitive with the best efficient protocols currently known.

1 Introduction

In the setting of secure multiparty computation, a set of parties wish to jointly compute some function of their inputs while preserving security properties such as *privacy*, *correctness*, *independence of inputs*, and more. These properties must be preserved in the face of adversarial behavior. In this paper, we consider security in the presence of three types of adversaries. The two classic adversary models are those of *semi-honest* adversaries that follow the protocol specification exactly but attempt to learn more than they should, and *malicious* adversaries that can behave as they wish and as such can arbitrarily deviate from the protocol specification. A more recent model, called security in the presence of *covert*

* Research generously supported by the European Research Council as part of the ERC project LAST. The first author was also supported by the ISRAEL SCIENCE FOUNDATION (grant No. 781/07).

P. Rogaway (Ed.): CRYPTO 2011, LNCS 6841, pp. 259–276, 2011.

adversaries, guarantees that if a malicious adversary behaves in a way that enables it to break the protocol in some way, then it will be caught cheating by the honest parties with some probability ϵ [1].

There are two rather distinct settings for studying the problem of secure multiparty computation. In the first, it is assumed that a majority of the parties are honest. In such a case, it is possible to securely compute any efficient functionality with information-theoretic security [3,4] assuming private channels (and broadcast, for the case of $n/3 \leq t < n/2$ corrupted parties). In the second setting, any number of the parties may be corrupted; this case includes the important two-party setting where one party may be corrupted. In this case of no guaranteed honest majority, information-theoretic security cannot be achieved. Nevertheless, assuming the existence of oblivious transfer, which can be constructed from enhanced trapdoor permutations and homomorphic encryption, it has been shown that any efficient functionality can be securely computed without an honest majority [24,12].

Beyond proving an important theorem stating that any functionality can be securely computed without an honest majority, the construction of [12] shows how to *compile* any protocol that is secure in the presence of semi-honest adversaries into a protocol that is secure in the presence of malicious adversaries, using one-way functions alone. This result is therefore often referred to as the GMW compiler. Recently, a new compiler was presented by Ishai, Prabhakaran and Sahai (IPS) [16]. This compiler works in a completely different way to that of the GMW compiler. First, unlike GMW, it does not compile an m-party protocol for securely computing a functionality f in the presence of semi-honest adversaries into an m-party protocol for securely computing the same f in the presence of malicious adversaries. Rather, the IPS compiler uses m-party protocols that securely compute some basic operations (like addition and multiplication in a finite field) in the presence of semi-honest adversaries with no honest majority, in order to transform a multiparty protocol that securely computes f in the presence of malicious adversaries with an honest majority, to an m-party protocol that securely computes f in the presence of malicious adversaries with *no honest majority*. As a specific example, note that by setting $m = 2$ the IPS compiler can generate a two-party protocol for computing f that is secure against malicious adversaries, from two-party protocols for computing basic functionalities (secure against semi-honest adversaries), and a multiparty protocol for computing f (secure against malicious adversaries which can corrupt only a minority of the parties). Intriguingly, the IPS compiler utilizes the world of information-theoretic secure computation in order to achieve security in the setting of no honest majority, since the basic protocol computing f, to which the compiler is applied, can be defined in an information-theoretic setting.

The IPS compiler has a number of important properties. First, it is black box in the underlying constructions; see [13] for why this is important. Second, it provides a uniform approach to both the two-party and multiparty settings, like the GMW compiler. Third, in some settings, it has excellent asymptotic efficiency. This is due to the fact that in some cases (e.g., when an arithmetic

circuit computing f is significantly smaller than a Boolean circuit computing f), information-theoretic protocols for the setting of an honest majority are much more efficient than computational protocols for the setting of no honest majority. The compiler can be applied to these more efficient protocols. This property has already been utilized to present protocols that have excellent theoretical, asymptotic efficiency [17]. Despite the above, it is unclear as to whether this approach can be used to achieve *concrete* efficiency for functionalities of interest [14].

1.1 The IPS Compiler

The compiler of [16] utilizes the following components in order to achieve the secure m-party computation of a functionality f, with no honest majority:

- A multiparty information-theoretic protocol π computing f with m clients who provide input and $n = O(m^2 k)$ servers who carry out the computation (where k is a security parameter analyzed in our work), which is secure in the presence of a malicious adversary corrupting a minority of the *servers*. This is called the *outer protocol* by [16].
- m-party subprotocols and local client instructions for simulation of the server computation in π, that are secure as long as the adversary behaves in a semi-honest manner. Given the known techniques for information-theoretic secure multiparty computation, it suffices for example to use m-party subprotocols for securely computing additive shares of the product of shares (all other steps can be carried using local client instruction for generating shares, adding shares, and so on). We stress that these subprotocols need only be secure in the presence of semi-honest adversaries. These are called the *inner protocols* by [16].

The way that the compiler works is for the m real parties to run the information-theoretic protocol π by emulating the operations of the n servers in π. This emulation is carried out using the secure m-party subprotocols, run between the m real parties (clients), to compute the next step of all parties in π. Thus, the n servers running π are virtual and are emulated by the m real parties running the protocol. Observe that if the real adversary were to behave in a semi-honest manner in each subprotocol, then the overall computation would clearly be secure. This is due to the fact that the emulation of the n parties in π is carried out using a protocol that is *secure* in the presence of semi-honest adversaries. Thus, f is securely computed, as guaranteed by π. However, the adversary here may be *malicious*.

The magic in the IPS compiler is how to leverage the semi-honest security of the subprotocols in order to achieve security in the presence of malicious adversaries. The central observation is that in order for a malicious adversary to cheat, it must cheat in at least $n/2$ of the subprotocols. This is due to the fact that π is secure unless a *majority* of the servers, namely at least $n/2$ of them, behave maliciously. In order to prevent such cheating, the IPS compiler sets up *watchlists*, which enable the honest parties to verify that malicious parties are not cheating. These watchlists are generated as follows. Each real party (client)

chooses the randomness that it will use when running the semi-honest subprotocol for each virtual server. Then, using oblivious transfer, each other client obtains k of the random strings of all other clients. Now, given the randomness that a party is supposed to use in the semi-honest subprotocol, it is possible to check that it is indeed behaving honestly. Furthermore, since oblivious transfer is used to obtain these strings, no client can know which semi-honest executions are being "watched". It is important to note, however, that it is not possible to raise the number of watchlists too high, because each time the randomness of a client (used with respect to some server) is watched, the internal state of the server is seen and that server is corrupted. It is shown in [16] that $n = O(k^2 m)$ servers are required in order to obtain security with a probability of cheating that is negligible in k. This number is important because it determines the number of servers in the information-theoretic protocol and thus its complexity.

In our presentation, we assume some familiarity with the IPS compiler. See the IPS papers for a description [16,17], or the brief tutorial of the construction in the full version of our paper.

1.2 Optimizations of the IPS Compiler

As will become clear in our concrete analysis of the efficiency of the compiler, see Sections 1.4 and 4, the number of servers n can become very large for some choices of parameters. This can have a severe effect on the efficiency of the protocol in a number of ways, one of these being the watchlist setup phase where $m^2 n$ executions of Rabin oblivious transfers must be carried out (each of these costing $\log n$ regular oblivious transfers; see Section 2 for a detailed explanation). This can therefore quickly become the bottleneck of the protocol. We therefore first devise a method for reducing the required number of servers to $n = O(mk)$ rather than $n = O(m^2 k)$, and second for setting up the watchlists in a way that costs an equivalent of $O(mn)$ regular oblivious transfers (rather than $O(m^2 n \log n)$ oblivious transfers). These optimizations are significant since they enable a better choice of parameters for the outer information-theoretic protocol. (Specifically, the best efficiency is obtained by using a protocol with many servers and a small fraction of corrupted parties; this enables the heavy use of the packed or multi-secret sharing methodology of [10].) Using our method, a smaller number of virtual servers can be corrupted and so a more efficient protocol can be used. We stress that when measuring concrete complexity, our new watchlist setup protocol is substantially more efficient, even for the two-party case where $m = 2$. An additional optimization is described in Section 2.3.

1.3 Variants of the IPS Compiler for Covert Adversaries

A simple compiler from covert to malicious security. We present an analog of the IPS compiler that uses subprotocols that are secure in the presence of covert adversaries instead of protocols that are secure in the presence of semi-honest adversaries. Recall that a protocol is secure in the presence of covert adversaries, with a deterrent parameter ϵ, if any cheating by an adversary is detected by the honest parties with probability at least ϵ [1]. (For our purposes,

it is convenient to assume that $\epsilon = 1/2$.) The use of subprotocols with this level of security fits naturally with the IPS paradigm: the information-theoretic outer protocol is emulated using protocols that are secure in the presence of covert adversaries with deterrent $\epsilon = 1/2$. Then, if the adversary tries to cheat in k of the subprotocol executions, it will be caught except with probability 2^{-k}. Observe that there is no need for any watchlists. In addition, the analysis and proof of security of this compiler are extraordinarily straightforward, since the cheating probability can be measured exactly with ease.

Beyond being a significant conceptual simplification, the usage of covert protocols also enables us to use an information theoretic protocol with just $n = m+2k$ parties (tolerating up to k corruptions), rather than $O(mk)$ parties when using semi-honest security. Relying on the fact that protocols that are secure in the presence of covert adversaries with deterrent $\epsilon = 1/2$ are only about 2–3 times the cost of semi-honest protocols, this results in an *asymptotic* efficiency improvement over the original compiler (we stress, though, that by our concrete analysis, the original compiler of [16] will typically be more efficient for *concrete* parameters since the use of watchlists means that local computation by a party can be checked directly and need not be distributed).

An IPS compiler from semi-honest to covert security. We observe that the IPS compiler with some minor modifications can be used to obtain security in the presence of covert adversaries from protocols that are secure in the presence of semi-honest adversaries, via a black-box reduction. We show that it suffices to use $O(m)$ watchlists and an oblivious transfer protocol that is secure in the presence of covert adversaries to set up those watchlists. (We also show that covert oblivious transfer can be constructed at the cost of only a constant number of semi-honest oblivious transfers.) This answers a major open question left by the work of [6].

IPS compilation and covert adversaries. Based on the above, we have that the IPS paradigm significantly contributes to our understanding of security in the presence of covert adversaries, and enables us to position covert adversaries in their natural place between semi-honest and malicious adversaries with respect to protocol constructions. In addition, as we show, it is possible to obtain quantitative improvements using this methodology. Specifically, we obtain a fully black-box reduction from semi-honest OT and one-way functions to malicious OT, at the cost of just a linear number of semi-honest OT invocations (the previous reduction of this type requires a quadratic number of semi-honest OTs [13]).

1.4 The Concrete Efficiency of the IPS Protocol

On an *abstract level*, the IPS compiler provides an elegant and conceptually simple way of constructing protocols that are secure in the presence of malicious adversaries. However, the actual instantiation of a protocol using the IPS approach depends on many different parameters and choices, all having a significant effect on the concrete efficiency of the result. To start with, appropriate inner and outer protocols must be chosen, and these choices are interdependent.

This is due to the fact that the most efficient information-theoretic outer protocol may require more invocations of the inner protocol for computing multiplications (which may be more expensive than other operations) than a less efficient protocol, when judging efficiency in the standard information-theoretic setting. Thus, the cost of running the inner multiplication protocol must be traded off with the cost of other operations in the outer protocol. In addition, there may be outer protocols that can utilize different inner protocols and obtain higher efficiency.

Another parameter that must be chosen is the exact number of servers $n = O(mk)$. Observe that each corrupted client has effectively corrupted k servers, since the watchlists it obtains from the honest parties reveal the internal state of these servers. In addition, some additional servers may be corrupted by the client cheating in the inner protocols emulating these servers, and hoping that it does not get caught. Based on this, it is clear that $n > 2mk$ since $m - 1$ corrupted parties have already effectively corrupted $(m-1)k$ servers, from the watchlists of the honest parties that they observe. However, how large should n be? A naive approach, which is to take n to be the smallest possible function of k, actually may have the opposite effect. For example, if $m = 2$ and $n = 4k$ then the corrupted party, who corrupts k servers through its watchlists, needs to cheat in k inner protocols in order to cheat in the outer In order to be concrete, let the number of clients m equal 2, and consider an outer protocol that tolerates any $t < n/2$ corruptions. We briefly analyze the difference between setting $n = 4k$ and $n = 3k$, where k is the number of watchlists viewed. In the case of $n = 4k$, the adversary needs to corrupt an additional k servers in order to have a dishonest majority of $2k$ servers, and this requires cheating in the simulation of at least k servers in the inner semi-honest protocols. The honest party does not detect this if its watchlists all fall in the other $3k$ (out of $4k$) servers. A rough analysis gives that the probability that the adversary succeeds in this case is $(3/4)^k$. In contrast, if $n = 3k$ then the adversary needs to corrupt an addition $k/2$ servers and so it successfully cheats with probability only $(2.5/3)^k = (5/6)^k$. Setting a fixed error of 2^{-40}, we have that when $n = 4k$ we need to set $k = 97$ and so $n = 388$, and when $n = 3k$ we need to set $k = 152$ and so $n = 456$. We therefore conclude that it is better to take $n = 4k$ than $n = 3k$ since this results in a lower number of servers n relative to the same cheating probability of 2^{-40}.

The above analysis relates to an outer protocol that tolerates any $t < n/2$ corruptions. However, the best protocols for this setting [7] use the packed secret sharing methodology of [10]. This methodology enables the effective multiplication of an entire block of shares using a single multiplication protocol as well as other efficiency improvements. Thus, large blocks can significantly lower the complexity of the protocol. However, the outer protocol must have the property that the number of corrupted parties that can be tolerated is upper bounded by the difference between the secret sharing threshold (which for [7] must be less than $n/4$) and the block size. This demonstrate the complexity of choosing good parameters since they are all interdependent. Observe that in our concrete analysis, we use k as the number of watchlists and not an independent security parameter; in the above asymptotic treatment these were the same.

2 Optimizations of the IPS Compiler

2.1 Efficient Watchlist Setup

As we have mentioned, the first step in the IPS compiler involves the setup of watchlists. Recall that the number of real parties is m and the number of virtual servers is n. Denote the real parties by $P_1, \ldots, P_m$. Technically, each party P_i chooses n random strings $r_i^1, \ldots, r_i^n$ (say, of length the security parameter k) and runs a two-party protocol with every other party P_j in which P_j receives k of the strings without P_i knowing which. The value r_i^ℓ is the random tape used by P_i in the semi-honest protocol simulating the ℓth server. Thus, any party knowing r_i^j can verify that P_i is running the protocol honestly.

The IPS setup. The method proposed in [16] is for each pair of parties P_i, P_j to run n executions of Rabin oblivious transfer [22]; in the ℓth execution the sender P_i inputs r_i^ℓ and the receiver P_j obtains this string with probability k/n, and obtains nothing otherwise. The result of this procedure is that the expected number of strings obtained by the receiver is $n \cdot k/n = k$, as required. As described in [16] it is possible to construct a single Rabin oblivious transfer with receipt probability k/n by running a single 1-out-of-n oblivious transfer, which in turn can be constructed using $\log n$ regular 1-out-of-2 oblivious transfers [19]. Thus, the cost of this setup phase is $n \log n$ oblivious transfers for P_i as sender and each P_j as receiver, with an overall of $m(m-1)n \log n$ oblivious transfers between all parties. With $n = O(m^2 k)$ as stated in [16] we have that the cost is approximately $O(m^4 k \log(m^2 k))$ oblivious transfers.

Concretely, this step can be carried out in two ways; one is to use the efficient extending of oblivious transfers of [15]. However, in the case of malicious oblivious transfer the oblivious transfer extension is not so efficient since it is based on the cut-and-choose methodology; in addition, it requires the use of the less standard assumption of correlation-robust hash functions. Alternatively, one can use a highly efficient OT protocol with $O(1)$ exponentiations per transfer [21] (the cost here is 11 exponentiations per transfer, with UC and stand-alone variants at essentially the same cost).

A new watchlist setup. We propose a new watchlist setup with the following properties. First, the method guarantees that each malicious party views the *same* k server watchlists for every honest party. This means that an adversary can examine the computation of k, rather than $(m-1)k$, servers. As we will see, this means that it suffices to set $n = O(mk)$ instead of $O(m^2 k)$. Second, it guarantees that each party views *exactly* k watchlists; this enables a tighter and more straightforward analysis of the probability that an adversary can cheat. Finally, we present a concrete protocol that gives a considerable efficiency improvement over the IPS setup, for both aforementioned implementation options.

Recall that the aim of the watchlist setup procedure is for each party to obtain k watchlists that it can view. Rather than achieving this by having each pair of parties run a separate procedure, we have the parties run a protocol for what we call multi-sender k-out-of-n oblivious transfer. This m-party functionality enables a receiver P_m to obtain k-out-of-n strings from $m-1$ different senders

$P_1, \ldots, P_{m-1}$ (each with a set of n strings). The functionality is formally defined in Figure 1.

Figure 1 (Multi-Sender k-out-of-n Oblivious Transfer $\mathcal{F}_n^k$).
Inputs: For every $j = 1, \ldots, n-1$, party P_j inputs a vector of n strings $(x_1^j, \ldots, x_n^j)$. The receiver P_m inputs a set of indices $I \subset [n]$ of size exactly k.
Outputs: If $|I| \neq k$ then all parties receive $\perp$ as output. Otherwise, for every $j = 1, \ldots, m-1$, party P_m receives the set $\{(i, x_i^j)\}_{i \in I}$ of k strings. Parties $P_1, \ldots, P_{m-1}$ receive no output.

We stress that the receiver is forced to use the same index set I for all sender parties. In the context of the IPS compiler, this means that each party can choose k servers for which it watches *all* parties. Now, let t be the number of corrupted real parties in the protocol. Then, the parties can together view $t \cdot k$ servers, which is equivalent to these servers being corrupted. In addition, the probability that the corrupted real parties can cheat in the semi-honest subprotocols for more than L servers equals the probability that none of the honest parties have any of these servers in their watchlists. A single honest party chooses k out of n watchlists and so the probability that it does *not* detect such cheating equals $\binom{n-L}{k} / \binom{n}{k}$.

Let $m = 2$ and let $n = 4k$. We have that the corrupted real party views k of the servers in its watchlist and needs to cheat in $L = k$ more in order to have corrupted at least half of the servers. The probability that it can do this without being detected is therefore $\binom{3k}{k} / \binom{4k}{k} < (3/4)^k$. Now consider the general case of m real parties and $n = 4mk$. We first analyze the case that $m-1$ parties are corrupted. Then, the adversary can view $(m-1)k$ different servers, and needs to corrupt $(m+1)k$ additional servers in order to have corrupted half of the servers. Thus, the probability that it goes undetected is $\binom{(3m-1)k}{k} / \binom{4mk}{k} < (3/4)^k$.

Thus, we conclude that it suffices to use $n = 4mk$ virtual servers rather than $O(m^2 k)$. In addition, as is evident, the proof of this fact is straightforward. (We stress that as shown in Section 4, this is not necessarily the best way to choose the parameters for concrete efficiency. However, here we are dealing with asymptotic efficiency.)

We conclude with the following somewhat informal claim; it's proof follows from the analysis above and the proof of security of [16].

Claim 2. *The IPS compiler with the watchlist setup phase using the multi-sender k-out-of-n oblivious transfer functionality is secure with $n = O(mk)$ servers.*

2.2 Securely Realizing Multi-sender k-out-of-n Oblivious Transfer

A general protocol from oblivious transfer. It is possible to securely realize the multi-sender k-out-of-n oblivious transfer functionality in a straightforward way using committed oblivious transfer, which in turn can be constructed in a

black-box way from any 1-out-of-2 oblivious transfer [5]. This construction has the advantage of preserving the IPS general structure of working under general assumptions and using any oblivious transfer protocol that is secure in the presence of malicious adversaries. Thus, we obtain the efficiency improvement regarding the number of servers and the simpler analysis, while remaining within the same framework. We note, however, that this strategy will not yield a concretely efficient watchlist setup.

A concrete protocol with greater efficiency. One of the aims in this paper, which is dealt with in detail in Section 4, is to consider the concrete efficiency of the IPS compiler. As such, in this section we present a highly-efficient protocol for securely computing the multi-sender k-out-of-n oblivious transfer functionality in the presence of malicious adversaries. The security of the protocol is based on the DDH assumption and requires just $O(n)$ exponentiations. Below, we compare the concrete efficiency of our watchlist setup method based on this protocol to the best-known concrete instantiations of the original IPS watchlist setup, and show that the efficiency improvement is dramatic. As we will see, the use of this protocol enables the use of significantly more servers than otherwise (in Section 4 it will become apparent why this is so important).

The protocol uses ideas from the cut-and-choose oblivious transfer protocol of [18]. The idea is for the receiver P_m to choose n pairs of group elements (a_i, b_i) so that relative to two fixed group elements g, h it holds that at most k out of the n tuples (g, h, a_i, b_i) are Diffie-Hellman tuples (and all the others are non-DH tuples). The receiver then broadcasts all of these pairs to the sending parties $P_1, \ldots, P_{m-1}$ and proves that at most k are DH tuples, as required. Following this, all of the sending parties send values with the property that P_m can obtain the ith string if and only if (g, h, a_i, b_i) is a DH tuple. This ensures that P_m receives k-out-of-n of the strings and no more. Furthermore, since the pairs are broadcast to all $P_1, \ldots, P_{m-1}$, it is guaranteed that P_m receives the ith string from every $P_1, \ldots, P_{m-1}$ if and only if (g, h, a_i, b_i) is a DH tuple. Thus, intuitively, the protocol securely computes the multi-sender k-out-of-n oblivious transfer functionality. See the full version of our paper for a full description and proof of security. The overall number of exponentiations in that protocol is $4n + (11n+k)(m-1)$. (For large m, this is approximately $11mn+km$ exponentiations. For the special case of $m = 2$ this comes to $15n + k$.)

A comparison of concrete efficiency. We now compare the concrete cost of running our watchlist setup protocol to the method of [16]. Recall that the IPS setup requires $m(m-1)n \log n$ oblivious transfers and this can be implemented using [21] at the cost of $11 \cdot m(m-1)n \log n$ exponentiations, or using the method of extending oblivious transfers of [15]. In contrast, our protocol requires $4n + (11n + k)(m - 1)$ exponentiations. (All exponentiations here are in any group in which the DDH assumption holds.)

For the sake of comparison and simplicity, assume that the same level of security is obtained using both setups and so the same values of k and n can be used. We compare this for two sets of concrete parameters given in Section 4.3, optimizing the performance of the inner protocol for the case of $m = 2$ parties.

For the AES-type circuit, we have $k = 207$ and $n = 1752$. Then, the cost of the original IPS setup is $m(m-1)n \log n = 2 \times 1752 \times 11 = 38544$ oblivious transfers. This can be implemented using [21] at a cost of 11 exponentiations per oblivious transfer, resulting in $423,984$ exponentiations. Alternatively, using the method of extending oblivious transfers of [15], the cost is about $6,000$ oblivious transfers (requiring $66,000$ exponentiations) and approximately $2,500,000$ hash function computations.[1] In contrast, our new setup costs $15n+k = 15 \cdot 1752+207 = 26,487$ exponentiations, which is much less (even than the solution using [15]). An even more illustrative example relates to the two settings of parameters given for the case of a circuit of size $30,000$ in Section 4.3; for this we just directly use the extending oblivious transfer alternative. With the first choice of parameters, $n = 19554$ and $k = 729$, we have that the IPS setup costs $66,000$ exponentiations and $38,700,000$ hash operations, versus $294,039$ exponentiations for our protocol. The other choice of parameters, of $n = 3362$ and $k = 292$, requires $66,000$ exponentiations and $5,300,000$ hash operations, versus $50,772$ exponentiations for our protocol. Thus, using our setup protocol, it is still feasible to choose either of the optimal choices of parameters and tradeoff the cost for the rest of the protocol. In contrast, using the IPS setup, only the latter setting of k and n can be reasonably used.

2.3 Flexibility of the Outer Protocol

In the original analysis carried out by [16] and that discussed above, the outer protocol chosen is secure in the presence of a malicious adversary that can adaptively corrupt up to t servers, for an appropriately chosen t. However, the corruptions of the servers are actually of two distinct types. The up to $(m-1)k$ corruptions that are due to the fact that the adversary sees the watchlists of the honest parties, are actually *semi-honest* corruptions, meaning that the adversary sees the internal state of these servers but does not cause them to deviate from the protocol specification. In contrast, the corruptions that are due to the corrupted real parties cheating in the semi-honest server simulation (without being caught) are *malicious* corruptions. Thus, it is possible to use an outer protocol that provides hybrid security in the presence of t_1 malicious corruptions and t_2 semi-honest corruptions, for appropriate t_1 and t_2. This model has been studied, and it has been demonstrated that better resilience can be achieved [9].

In order to be concrete, we demonstrate this on the parameters discussed above in Section 2.1. When $m = 2$ and $n = 4k$, we have that the corrupted party views k of the servers and needs to actively corrupt k more. Thus, the outer protocol needs to be secure in the presence of k malicious servers and k semi-honest servers, rather than $2k$ malicious servers. This can significantly simplify the outer protocol and yield higher efficiency. (For example, the simple and efficient multiplication protocol of BGW [3] for the case of $t < n/4$ malicious corruptions can also be used in the case of $t_1 < n/6$ malicious corruptions together

[1] This calculation is based on a 128-bit seed, and setting the parameter σ of the extending OT scheme to be of size 44 in order to obtain an error of 2^{-40} in the oblivious transfer extension protocol; see [15, Figure 2].

with $t_2 < n/6$ semi-honest corruptions. Thus, although the overall number of corruptions is $t < n/3$, the simpler and more efficient multiplication protocol can be used. This can be heavily utilized in the setting of IPS compilation.)

3 IPS Variants Using Covert Adversaries

In this section we present variants of the IPS compiler in order to obtain security in the presence of malicious adversaries from security in the presence of covert adversaries, and security in the presence of covert adversaries from security in the presence of semi-honest adversaries. In addition, we show that this approach has a quantitative advantage regarding the black-box construction of malicious oblivious transfer from semi-honest oblivious transfer.

3.1 Secure Computation for Malicious from Covert Adversaries

We show an extraordinarily simple analog of the IPS compiler when the starting point is a protocol for secure computation in the presence of covert adversaries. As we have already discussed the idea behind our construction in Section 1, we proceed directly to the construction. We construct a protocol for computing a function f for m parties, where any number of them can be corrupt. The protocol is secure against malicious adversaries. The security parameter is denoted by k. The protocol uses the following tools.

Outer protocol: Let π be a multiparty (outer) protocol for m clients and $n = 2k$ servers, which is secure for any number of corrupted clients and as long as less than k servers are corrupted by an adaptive malicious adversary. π computes the function f where parties $P_1, \ldots, P_m$ provide input and receive output. For simplicity, the protocol π is such that all messages are sent over a broadcast channel, and every party broadcasts in every round

Server protocols: Let $\pi_1, \ldots, \pi_{m+n}$ be the instructions for the different parties in π. That is, the clients $P_1, \ldots, P_m$ run $\pi_1, \ldots, \pi_m$ and the ith server runs π_{m+i}. For the servers, namely for $i = 1, \ldots, n$, let $\mathcal{F}_{m+i}$ be the *reactive* ideal functionality computing π_{m+i}. Loosely speaking, $\mathcal{F}_i$ is a functionality that receives $n + m - 1$ inputs in each round and generates a single output; the $m + n - 1$ inputs are the values broadcast by all parties P_j for $j \neq i$ in the previous round and the output is the value that P_i should broadcast in this round. The exact description of the functionality $\mathcal{F}_i$ is more involved. This functionality is actually run by the m real parties. Thus, each party inputs a vector of length $m + n$ with the values broadcast in the previous round. The functionality then verifies that *all* vectors input by the m clients are identical. If not, it outputs $\perp$. If yes, it computes the next message that P_i would send and hands it to all the m clients.

Covert model functionality: We denote by $\mathcal{F}^{\epsilon}_{m+i}$ the functionality $\mathcal{F}_{m+i}$ in the covert model with deterrent ϵ. This means that we consider an ideal functionality that computes $\mathcal{F}_i$ with the additional instructions of the trusted party of the ideal model of the definition of covert adversaries; see [1].

Protocol 3 uses these tools to obtain security in the presence of a malicious adversary controlling an arbitrary number of the m parties. The protocol is defined in a hybrid model where the functionalities $\mathcal{F}^{\epsilon}_{m+1}, \ldots, \mathcal{F}^{\epsilon}_{m+n}$ are executed by a trusted party. Security is derived when these functionalities are instantiated by real protocols via standard composition theorems.

Protocol 3 (Security for Malicious in the Covert $\mathcal{F}^{\epsilon}_{m+i}$-Hybrid Model).

Inputs: Real parties $P_1, \ldots, P_m$ hold respective inputs $x_1, \ldots, x_m$

The protocol: For every round of protocol π, the parties $P_1, \ldots, P_m$ do:

1. Each party P_j $(1 \leq j \leq m)$ broadcasts the message that π instructs the client P_j to send in this round (using π_j), based on the messages from the last round.
2. For every $i = 1, \ldots, n$, each party P_j $(1 \leq j \leq m)$ sends to the ideal functionality $\mathcal{F}^{\epsilon}_{m+i}$ the vector of all messages broadcast in the previous round. (In the first round, the vector contains $m + n$ empty values λ.)
3. For every $i = 1, \ldots, n$, each party P_j $(1 \leq j \leq m)$ receives an output from $\mathcal{F}^{\epsilon}_{m+i}$. If the output is $\mathsf{corrupted}_\ell$ or abort_ℓ (see [1]), then P_j halts and outputs abort_ℓ. Otherwise, it records the output as the message "broadcast" by server P_{m+i} in this round.

Output: Each party P_j $(1 \leq j \leq m)$ outputs the value that π instructs client P_j to output.

We now state the security of Protocol 3. The proof appears in the full version of the paper; it is very straightforward, and this highlights the conceptual advantage of this alternative IPS compiler.

Theorem 4. *Let π be a protocol for m clients and $n = 2k$ servers that securely computes the m-party functionality f with abort, in the presence of an adaptive malicious adversary corrupting any number of clients and less than k of the servers, and let $\epsilon > 0$ be any constant. Then, Protocol 3 securely computes f with abort in the $\mathcal{F}^{\epsilon}_{m+1}, \ldots, \mathcal{F}^{\epsilon}_{m+n}$ hybrid model, in the presence of an adaptive malicious adversary corrupting any number of parties.*

3.2 Secure Computation for Covert from Semi-Honest Adversaries

We describe a black-box transformation from semi-honest protocols to covert protocols, using a covert oblivious transfer protocol. This result answers an open question left by the work of [6], which showed a similar transformation in the information-theoretic setting with an honest majority, but did not cover the case of a majority of corrupted parties. The construction is similar to the original construction of IPS [16] with two exceptions. First, only a small number of watchlists are used. Second, it suffices for us to use an oblivious transfer protocol with security for covert adversaries, rather than security for malicious adversaries, in order to set up the watchlists. (Oblivious transfer protocols with security for covert adversaries with constant ϵ can be constructed using $O(1)$ black-box invocations of semi-honest OT, as described in the full version of our paper.)

The protocol computes a function f for m parties, where any number of them can be corrupt. The security parameter is denoted by k.

Tools:

- Let π be a multiparty protocol for m clients and $n = 4m$ servers, which is secure for any number of corrupted clients and less than $n/2$ corrupted servers. As in Protocol 3, all messages of π are sent over a broadcast channel and every party broadcasts in every round. Furthermore, the clients $P_1, \ldots, P_m$ are the only ones who provide input and receive output.
- Let $\pi_1, \ldots, \pi_{m+n}$ be the instructions for the parties in π, and let $\mathcal{F}_{m+i}$ be the *reactive* ideal functionality computing π_{m+i}, for $i = 1, \ldots, n$. The functionality is as defined for Protocol 3.
- Let $\rho_{m+1}, \ldots, \rho_{m+n}$ be m-party protocols such that ρ_{m+i} securely computes $\mathcal{F}_{m+i}$ in the presence of semi-honest adversaries. Without loss of generality we assume that the random-tape of each party in each ρ_i is of length exactly k (a pseudorandom generator can be used if it is longer).

The compiler is described in Protocol 5.

Protocol 5 (Security for Covert from Semi-Honest).

Inputs: Real parties $P_1, \ldots, P_m$ hold respective inputs $x_1, \ldots, x_m$

The protocol:

1. *Phase 1 – set up watchlists:*
   (a) For every $j = 1, \ldots, m$, party P_j chooses a vector of $n = 4m$ random seeds $s_1^j, \ldots, s_{4m}^j \in \{0,1\}^k$. The parties all then run m multi-sender 1-out-of-n oblivious transfers that are secure in the presence of covert adversaries, so that each party P_j receives $\{s_{r_j}^i\}_{i=1}^m$ for m random indices $r_j \in_R \{1, \ldots, n\}$.
   (b) At the conclusion of this phase, each client P_j holds the following:
      i. A vector $\overline{s}_j = (s_1^j, \ldots, s_n^j)$ of random seeds chosen by P_j
      ii. A set of strings $\{s_{r_j}^i\}_{i=1}^m$ received by P_j from others
2. *Phase 2 – emulate π:* For every round of the $n = 4m$-party protocol π, the parties $P_1, \ldots, P_m$ work as follows:
   (a) Each party P_j $(1 \le j \le m)$ broadcasts the message that π instructs the client P_j to send in this round, based on the previous messages.
   (b) For every $i = 1, \ldots, n$, each party P_j $(1 \le j \le m)$ runs ρ_{m+i} with input the vector of all messages broadcast in the previous round, and using random-tape s_i^j.
   (c) Each party P_j checks its watchlists for the executions run in the previous step. Specifically, for every $\ell = 1, \ldots, m$ $(\ell \neq j)$, party P_j verifies that party P_ℓ computed the next message according to ρ_{m+r_j} using the random tape $s_{r_j}^\ell$ and the messages broadcast. If no, then P_j outputs $\mathsf{corrupted}_\ell$ (signifying that P_ℓ cheated) and halts.
   (d) For every $i = 1, \ldots, n$, each P_j $(1 \le j \le m)$ receives from ρ_{m+i} an output and records it as the message "broadcast" by P_i in this round.

Output: Each party P_j $(1 \le j \le m)$ outputs the value that π instructs client P_j to output.

Security is stated by the following theorem (proved in the full version).

Theorem 6. *Let π be a protocol for m clients and $n = 4m$ servers that securely computes the m-party functionality f with abort, in the presence of an adaptive malicious adversary corrupting any number of clients and a minority of servers. Then, Protocol 5 securely computes f in the $\mathcal{F}_{m+1}, \ldots, \mathcal{F}_{m+n}$ (semi-honest) hybrid model, where $n = 4m$, in the presence of an adaptive covert adversary corrupting any number of corrupted parties, with ϵ-deterrence for $\epsilon > 1 - e^{-0.25}$.*

3.3 The Semi-honest Cost of Malicious Oblivious Transfer

In the full version of this paper, we use our methodology of IPS compilation via covert adversaries to prove the following theorem:

Theorem 7. *There exists a black-box reduction from bit oblivious transfer that is secure in the presence of malicious adversaries to one-way functions and $O(k)$ invocations of bit oblivious transfer that is secure for semi-honest adversaries.*

Previously, the best known such reduction required $O(k^2)$ invocations of semi-honest oblivious transfer [13].

4 The Concrete Efficiency of IPS

In this section, we describe our analysis of the *concrete* efficiency of the best IPS-type protocols. Due to the high level of abstraction in the IPS construction, its concrete complexity was completely unknown.

The protocol that we examined is based on sharing values using block secret sharing as in [10], in which ℓ values are encoded in a single polynomial. Thus, given blocks $a = (a_1, ..., a_\ell), b = (b_1, ..., b_\ell)$ which are shared using two polynomials of degree δ, addition results in a sharing of a polynomial of degree δ that hides the block $a+b = (a_1+b_1, ..., a_\ell+b_\ell)$, while multiplication results in sharing a polynomial of degree 2δ which hides the block $ab = (a_1 b_1, ..., a_\ell b_\ell)$; as usual, a protocol is used to reduce the degree of the polynomial to δ.

An in-depth analysis of the protocol, described in the full version of the paper, reveals that the overall complexity of the protocol is dominated by the number of multiplications and the number of OTs (both the communication complexity and other computational operations are negligible in comparison to these). Our efficiency analysis will therefore present those two factors. The OTs are only needed for the inner semi-honest multiplication protocols,[2] and so the other building blocks will be analyzed only in terms of the number of multiplications. We emphasize that these OTs must only be secure against a semi-honest adversary, and not a malicious one.

[2] We remark that the OTs needed for setting up the watchlists (which must be secure against malicious adversaries) are also a factor. However, they depend only on the number of servers n and so can be considered at the end.

4.1 An Analysis of the Building Blocks

Secret sharing for blocks: This secret sharing scheme is a variant of Shamir's secret sharing [23], presented in [10]. Each polynomial encodes a block of ℓ values. The cost of sharing w elements among n servers, using blocks of size ℓ and polynomials of degree δ, is $(w/\ell)(\delta^2 + n\delta)$ multiplications.

Proving that shares lie on δ-degree polynomials: After sharing the secrets it must be proved that the shares are indeed encoded by z polynomials, each of degree δ (each polynomial is used to hide ℓ field elements; depending on the number of inputs, multiple polynomials must be used for the sharing). A protocol for such a proof is presented in [17]. It requires $\delta(z + n + k)$ multiplications.

Proving a replication pattern of shared blocks: The protocol requires parties to prove that certain shared blocks follow some replication pattern (namely that a certain output value is used as an input value for the next layer). The protocol we used is mentioned in the computation complexity analysis of [17]. The cost is $(4\delta)^2 + 4\delta n + 2((2\delta n + (2\delta)^2)u + (\delta n + \delta^2)v) + 2(n + k)(u + v)$ multiplications, where v is the number of input blocks (represented by polynomials of degree δ) and u is the number of output blocks (represented by polynomials of degree 2δ).

Semi-honest inner multiplication: For multiplication gates the parties run a semi-honest protocol for the functionality $(x_1, x_2) \mapsto (x_1 x_2 - r, r)$ for a random $r \in_R \mathcal{F}$. Six different protocols are presented for this functionality in [17]. We present the analysis for the most efficient protocols only (based on our concrete analysis for all options). The first protocol is based on packed Reed-Solomon encoding and is black-box in the field [17], and the second protocol is due to Glboa [11] and makes nonblack-box usage of the field and assumes standard bit representation of elements.

As is detailed in the full version of the paper, for a security parameter $s = 40$ giving error 2^{-40}, the packed Reed-Solomon encoding protocol costs 2734 multiplications and 16 1-out-of-2 OTs per inner multiplication, while the protocol of [11] uses 40 multiplications and 40 1-out-of-2 OTs. Namely, one protocol is more efficient in terms of OTs and the other is more efficient in terms of multiplications. For concrete numbers this phenomenon might present implementers with a real dilemma.

4.2 Instantiating the Parameters

In order to count concrete efficiency, the values of the different parameters must be set. We do not claim to have found the absolute optimal parameters, as the analysis of their effect on the overhead is very complex. We do present for each parameter the different considerations affecting the choice of its value, and eventually show that the protocol is comparable in its efficiency to other protocols from the literature and may be competitive in some settings.

The four main parameters that must be set are the degree of the polynomials δ, the block size ℓ, the number of corrupted parties tolerated t, and the number of servers n. Three out of the four different parameters, the degree, the block

size and the corruption threshold, are tightly interconnected in that setting any two of them determines the third one. In addition, these three parameters are all chosen as a function of the number of servers n, and given their descriptions the actual concrete value of n is chosen (independently of the circuit). As we will see below, the determination of the degree δ is a straightforward choice. We then determine the block size based on the actual circuit being computed, thereby essentially setting the threshold.

The degree δ: Due to the replication proof protocol, it must hold that $\delta < n/4$. Other than that δ should be maximized, and we therefore set $\delta = n/4 - 1$ (for simplicity of notation, from here on we write $\delta = n/4$).

The block size ℓ: The block size has to be strictly smaller than the degree δ, but otherwise the larger the block size, the more efficient the outer protocol gets. This is because more multiplications are carried out together (note that there is no use in having a block size larger than the width of a layer in the circuit since this already upper bounds the number of multiplications that can be carried out together). However, the number of corrupted servers that the outer protocol can tolerate is $\delta - \ell$, thus the closer ℓ is to δ, the smaller the fraction of corrupt servers that can be tolerated. As a result, more servers are required in order to ensure that the probability of catching the adversary cheating in the server simulation does not go down.

The corruption threshold t: As shown in [10], up to $t < \delta - \ell$ corrupted servers receive no information about the secret block when using block secret sharing with degree δ and block size ℓ. Thus t is set to $\delta - \ell - 1$.

The number of servers n: Define $\tau = n/t$, and so $1/\tau$ is the ratio of servers that the adversary needs to corrupt in order to successfully cheat. Denote $n = O(mk) = a \cdot mk$ for some parameter a which should be chosen to minimize n. We have already observed in Section 1.4, however, that the naive approach of minimizing a is not optimal. An analysis reveals that for the two-party case the best choice is to take $a = \tau$, and thus to use $n = 2\tau k$ servers in the outer protocol.

4.3 Setting Concrete Values

We now show the concrete cost of IPS based on the results of our analysis regarding parameter instantiation. The choices of parameters are demonstrated for two different circuits, which clarify the dilemmas that arise in practice. As stated above the only parameter which we did not set independently of the circuit, but rather want to optimize for the concrete circuit, is the block size. We therefore calculated the number of operations required for a large range of block sizes. Our calculations are based on a combination of an analytic and numerical analysis of the parameters that yield a cheating probability of at most 2^{-40}.

The first example uses circuit parameters similar to the AES circuit of [8], assuming 2400 multiplication gates split over 100 layers. In this case minimal values for the number of OTs and the number of multiplications occur for the same block size of $n/73$. Remembering that $\delta = n/4$, the threshold ratio is $\tau = \frac{1}{(\delta-\ell)/n} \approx 4.231$. Setting $a = \tau$ as suggested above results in $n = 4.231 \cdot 2k$.

In order to get a cheating probability of 2^{-40}, each client must check $k = 207$ watchlists, and the number of servers is $n \approx 1752$.[3]

The number of OTs and multiplications which are required in this setting depends on the inner multiplication protocol that is used (based on Reed-Solomon codes, or on the protocol of [11]). The first choice results in approximately $5.5 \cdot 10^6$ OTs and $5.5 \cdot 10^9$ multiplications, while the latter choice requires approximately $13.8 \cdot 10^6$ OTs and $4.5 \cdot 10^9$ multiplications. The choice of the inner protocol is therefore not trivial, and depends on the properties of concrete implementations of the OT and multiplication primitives. Recall that multiplications are in a finite field of size 2^{40} and therefore elements fit in a single word of a modern 64-bit architecture, and can be done very efficiently. The OTs need only be secure against semi-honest adversaries, and so can be efficiently implemented using methods of extending OT as in [15]. Given these two observations, the run time of the protocol seems reasonable in comparison to that of other protocols providing security against malicious adversaries.

The second example is of a circuit of 30000 multiplication gates split over 10 layers, and results in another optimization dilemma. Setting the block size $\ell = n/5.7$ results in the minimal number of OTs, but minimizing the number of multiplications requires setting $\ell = n/13.1$. The actual numbers of operations are described below.

	n	k	RS OT	Gilboa OT	RS mult	Gilboa mult
$\boldsymbol{\ell = n/5.7}$	19554	729	$5.6 \cdot 10^6$	$11.1 \cdot 10^6$	$54 \cdot 10^9$	$53 \cdot 10^9$
$\boldsymbol{\ell = n/13.1}$	3362	292	$12 \cdot 10^6$	$31 \cdot 10^6$	$13 \cdot 10^9$	$11 \cdot 10^9$

The reason for this tradeoff is that the number of OTs is minimized when the block size can accommodate an entire layer in a single block, but this setting requires more servers (compared to a smaller block size), and so multi-point evaluation and interpolation become more expensive (as they depend on the number of servers), which results in an increased amount of multiplications. Observe also that setting $\ell = n/5.7$ results in a much larger number of servers which in turn affects the cost of the watchlist setup protocol. Plugging in the cost of our watchlist setup ($15n + k$ exponentiations), we have that when $\ell = n/5.7$ the setup cost is $294,039$ exponentiations, versus just $50,722$ when $\ell = n/13.1$. This cost may also weigh in as a factor.

We conclude that the IPS protocol may be competitive in some settings. We are currently implementing the protocol in order to empirically verify our analysis and conclusions.

References

1. Aumann, Y., Lindell, Y.: Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. J. of Cryptology 23(2), 281–343 (2010)
2. Beaver, D.: Correlated Pseudorandomness and the Complexity of Private Computations. In: The 28th STOC, pp. 479–488 (1996)

[3] Note that the block size, of $n/73 = 24$, is equal to its maximum reasonable value, namely to the width of a layer of the circuit, which is also 24.

3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In: 20th STOC, pp. 1–10 (1988)
4. Chaum, D., Crépeau, C., Damgård, I.: Multi-party Unconditionally Secure Protocols. In: 20th STOC, pp. 11–19 (1988)
5. Crépeau, C., van de Graaf, J., Tapp, A.: Committed Oblivious Transfer and Private Multi-Party Computation. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 110–123. Springer, Heidelberg (1995)
6. Damgård, I., Geisler, M., Nielsen, J.B.: From Passive to Covert Security at Low Cost. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 128–145. Springer, Heidelberg (2010)
7. Damgård, I., Ishai, Y.: Scalable Secure Multiparty Computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006)
8. Damgård, I., Keller, M.: Secure Multiparty AES. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 367–374. Springer, Heidelberg (2010)
9. Fitzi, M., Hirt, M., Maurer, U.M.: Trading Correctness for Privacy in Unconditional Multi-Party Computation. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 121–136. Springer, Heidelberg (1998)
10. Franklin, M.K., Yung, M.: Communication Complexity of Secure Computation (Extended Abstract). In: The 24th STOC, pp. 699–710 (1992)
11. Gilboa, N.: Two Party RSA Key Generation (Extended Abstract). In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 116–129. Springer, Heidelberg (1999)
12. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In: 19th STOC, pp. 218–229 (1987)
13. Haitner, I., Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-Box Constructions of Protocols for Secure Computation. SIAM Journal on Computing 40(2), 225–266 (2011)
14. Ishai, Y.: Personal Communication (2011)
15. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
16. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding Cryptography on Oblivious Transfer – Efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
17. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure Arithmetic Computation with No Honest Majority. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 294–314. Springer, Heidelberg (2009)
18. Lindell, Y., Pinkas, B.: Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)
19. Naor, M., Pinkas, B.: Oblivious Transfer with Adaptive Queries. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 573–590. Springer, Heidelberg (1999)
20. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
21. Peikert, C., Vaikuntanathan, V., Waters, B.: A Framework for Efficient and Composable Oblivious Transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008)
22. Rabin, M.: How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U. (1981)
23. Shamir, A.: How to Share a Secret. Communications of the ACM 22(11), 612–613 (1979)
24. Yao, A.: How to Generate and Exchange Secrets. In: 27th FOCS, pp. 162–167 (1986)