Secure Two-Party Computation is Practical

(Full Version)*

Benny Pinkas[1], Thomas Schneider[2], Nigel P. Smart[3], and Stephen C. Williams[3]

[1] Dept. of Computer Science,
University of Haifa
Haifa 31905, Israel
benny@pinkas.net
[2] Horst Görtz Institute for IT-Security,
Ruhr-University Bochum,
D-44780 Bochum, Germany,
thomas.schneider@trust.rub.de
[3] Dept. Computer Science,
University of Bristol,
Woodland Road,
Bristol, BS8 1UB, United Kingdom,
{nigel,williams}@cs.bris.ac.uk

Abstract. Secure multi-party computation has been considered by the cryptographic community for a number of years. Until recently it has been a purely theoretical area, with few implementations with which to test various ideas. This has led to a number of optimisations being proposed which are quite restricted in their application. In this paper we describe an implementation of the two-party case, using Yao's garbled circuits, and present various algorithmic protocol improvements. These optimisations are analysed both theoretically and empirically, using experiments of various adversarial situations. Our experimental data is provided for reasonably large circuits, including one which performs an AES encryption, a problem which we discuss in the context of various possible applications.

1 Introduction

That secure multi-party computation can be executed at all is considered one of the main results of the theory of cryptography. Starting with Yao's seminal work [39] many authors have looked at various optimisations and extensions to the basic concept, for both the two-party and the multi-party settings, see for example [8, 11, 12, 20, 23, 26, 38]. Until recently all work on secure multi-party computation has been essentially of a theoretical nature, focusing on feasibility results. However in the last few years a number of practical implementations have appeared [6, 7, 25, 27, 4].

There are many different protocols for secure multi-party computation. Our work focuses on implementation of secure computation and therefore we only mention protocols which have been previously implemented. Secure multi-party computation essentially comes in two flavours. The first approach is typically based upon secret sharing and operates on an arithmetic circuit representation of the computed function, such as in the BGW (Ben-Or, Goldwasser and Wigderson) or CCD (Chaum, Crepeau and Damgård) protocols [5, 9]. This approach is usually applied when there is an honest majority among the participants (which can only exist if more than two parties participate in the protocol). An alternative approach represents the function as a binary circuit. This approach was used in the original two-party garbled circuit construction of Yao [39], and in the GMW (Goldreich, Micali and Wigderson) multi-party protocol [12].

The arithmetic circuit method is better at representing addition and multiplication operations, where parties have additive shares of secret values, but cannot be used to compute comparisons unless the shares are converted to shares of the binary representation of the values. This approach has been used to great effect in the SIMAP project [7], which has resulted in a "real-life" application of secure multi-party computation to the Danish sugar beet industry [6].

* Please cite the conference version of this paper published at ASIACRYPT'09 [34].

The binary circuit approach handles arithmetic operations, especially multiplications, less efficiently, but can easily compute binary operations such as comparisons. This second approach, which forms the basis of Yao's construction for the two party case, has been implemented by Malkhi et al. in the Fairplay system [27]. That system also provides a method to compile a given functionality from a representation in a high-level language into a circuit, which is then interpreted by a run-time environment that performs the secure evaluation of this functionality. FairplayMP, an extension of Fairplay to the case of more than two parties using a modified version of the protocol of Beaver et al. [3] has recently been released [4]. All these implementations provide security against semi-honest adversaries only. A major advantage of the binary circuit based systems (Fairplay and FairplayMP) is that they run in a constant number of communication rounds, whereas the SIMAP system has the advantage of being able to process arithmetic operations very efficiently.

Efficient extensions of Yao's construction to more relevant adversarial models have been a topic of research interest in the last few years. There are several constructions which aim to secure the protocol against malicious adversaries without using generic zero-knowledge protocols. We will focus on the construction of Lindell and Pinkas [23] which is efficient and provides fully simulatable security according to the definition of Canetti [8][4]. A definition of a weaker class of corruption, "covert adversaries", and a protocol secure against this type of behavior, was provided by Aumann and Lindell [1]. In [25] an implementation of the basic Lindell–Pinkas protocol was reported upon and experimental data in various security models was provided.

In this paper we improve on the implementation of [25] in a number of ways. The resulting set of quantitative improvements results in qualitative conclusions: (1) We demonstrate that two-party computation, secure against malicious adversaries, is truly practical, and we experimentally identify the performance bottlenecks which remain after our optimisations. This result should direct further research to the issues which have the largest effect on performance. (2) We experiment with a secure computation of the AES standard, and show that it is indeed feasible, even with security against malicious adversaries. There are a number of applications of such an implementation, some of which we describe below. (3) We provide the first implementation of a protocol with security against covert adversaries and we compare the performance of all 3 types of protocols: malicious, covert and semi-honest.

A more detailed summary of our main results is as follows:

- We improve the communication cost for transmitting the circuits between the parties. In the case when we model the underlying key derivation functions (KDFs) as correlation robust (see discussion below), using the technique of [21] we are able to transmit no information for the XOR gates within the circuit. In this situation we are also able to reduce the data which needs to be sent by 25% for the other gates. When we are not willing to model the KDFs as correlation robust, and we only assume they are psuedo-random functions, we are unable to perform the free XOR optimisation. However we are able to reduce the communication cost for all gates by 50%. Unlike other methods used to improve communication, like [14], our improvement makes a marginal impact on computational costs. We will return to this in a later section.
- In addition to the theoretical analysis we provide experimental data for evaluating "real life" circuits, in both the honest-but-curious, covert and malicious adversary cases; also for the two different methods in the literature that construct the auxillary circuits in the covert and malicious cases (see [25] and Appendix sec:extension). The implementation for the malicious setting is based on the construction of Lindell and Pinkas [23] which provides security in the sense of full simulatability. Therefore the resulting construction can be used as a black-box primitive in more complex applications. The use of our optimisations results in a considerable performance boost compared to previous experimental results published in [25].

[4] This construction may be preferable over other two-party protocols with security against malicious adversaries. The construction of Mohassel and Franklin [26] only protects privacy and is not fully simulatable. The construction of Jarecki and Shmatikov [20] requires the use of public-key operations, rather than symmetric key operations, for any gate of the circuit. The construction of Nielsen and Orlandi [29], too, uses public key operations, or rather public-key based commitments, for each key of every wire of the circuit. A precise practical comparison between the different approaches is beyond the scope of the current paper

Our optimisations change the performance bottleneck to a different part of the computation; namely, the verification of garbled circuits generated by the circuit constructor. This observation is important for focusing future research on the issues that affect the overhead the most.
- We experiment with secure evaluation of a circuit which computes an AES encryption of a single block. The secure computation of AES involves one party which knows the key, and a different party which has an input block. The second party learns the encryption of the block, while the first party learns nothing. We demonstrate the feasibility of computing this function in the semi-honest, covert and malicious settings.

1.1 Secure Computation of AES

As aluded to above, we present experimental results for a circuit which evaluates the AES encryption of a single block. The secure computation of AES involves one party which knows the key, and a different party which has an input block. The second party learns the encryption of the block, while the first party learns nothing.

The fact that a secure computation of AES is feasible, and can run in a matter of seconds, is quite surprising. This function is much more complex than say, the millionaires problem, which merely compares two numbers. The circuit used to implement AES is composed of over 30,000 gates. Smaller hardware implementations are known, but they typically reuse gates by performing different parts of the computation with the same gate in different clock cycles; this is something which one cannot do with Yao circuits.

Application 1, OPRF: A secure computation of a pseudo-random function, denoted OPRF for "oblivious prf", has been defined in [10] and used there for secure keyword based searches. It had actually been used even earlier for implementing oblivious transfer with adaptive queries [30]. OPRF was subsequently used in different constructions, such as protocols for set intersection and pattern matching secure against malicious and covert adversaries [16], or protocols for private itemset support counting [22]. It was also used in a recent system for secure collaborative large-scale data aggregation for identifying denial-of-service attacks [36].

Earlier OPRF constructions were described based on a circuit construction and Yao's protocol, for example in [2], but were not considered a task that one would like to implement. A recent OPRF protocol is based on the Naor-Reingold pseudo-random function [32], which is a number theoretic construction whose security depends on the DDH assumption or on the hardness of factoring. That protocol is secure against semi-honest adversaries (whereas we show security against malicious adversaries). In terms of overhead, a secure protocol applying the NR OPRF to ℓ bit inputs requires about ℓ public-key operations, which is about the same number of public-key operations as in our protocol. There are, however, several advantages for using a symmetric key based OPRF, such as our AES based construction:

Multiple plain invocations of the function. An OPRF is typically used in protocols where the PRF is evaluated many times by a certain party, and then the other party runs an OPRF protocol to learn the value of the pseudo-random function applied to a certain value. For example, the first party might compute many encryptions and publish a list of the ciphertexts. The second party uses OPRF to obtain the encryption of a specific value, and checks whether it appears in the list of ciphertexts. If the Naor-Reingold function is used as the PRF, then the first party must perform exponentiations for each of the many invocations of the PRF that it must perform. If AES is used then these encryptions can be computed very efficiently.

Range. The range of the outputs of the NR PRF is a group in which the DDH assumption holds, e.g., a subgroup of Z_p^*. The output of AES is the set of all 128 bit long strings. The fact that each bit string is a potential output of the function might be required for some applications.

Permutation. AES is a pseudo-random *permutation*, while the NR function is a function. Furthermore, the range of the inputs of the NR function is different than the range of its outputs. There might be applications which require the usage of a pseudo-random permutation rather than a pseudo-random function.

Assumptions. The Naor-Reingold PRF construction is based on number theoretic assumptions. This is uncommon for symmetric cryptographic functions, and can even be considered a weakness. It might

be that the mathematical elegance of number theoretic assumptions makes it easier to break them, compared to breaking AES.

It is easy to add security against malicious or covert adversaries for our secure computation of AES. In fact, we show below experimental results for these implementations. This is not unique for our OPRF implementation, as it was shown in [16] that similar security can be achieved for the NR based OPRF by using OT protocols with corresponding security guarantee.

Application 2, Side Channel Protection: In [13] the authors introduce "one-time programs", which are programs that can only be executed once and then "self-destruct". This type of programs can be used for different cryptographic applications, and in particular can replace some of the applications of software obfuscation, and can support one-time proofs which can only be verified once. An important advantage of this construction is that the execution of the program reveals no side-channel information, and therefore provides a strong security guarantee against side-channel attacks. The construction of one-time programs must use a simple memory device which essentially implements a one-time oblivious transfer. The rest of the computation is essentially done using a garbled Yao circuit, where the one-time program is executed on inputs of the device's choosing using the data held in the one-time-memory.

One of the main applications of smart cards is to compute symmetric encryptions, and therefore the ability to compute AES encryptions by Yao circuits has immediate application in the above scenario. It enables smart cards to perform a one-time computation, secure against side-channel attacks, of AES. This is particularly true since in the one-time program setting the Yao circuit evaluation need only be secure against semi-honest adversaries. We show below that for semi-honest adversaries the AES circuit can be evaluated very efficiently, in around two seconds, making the method of [13] eminently practical.

Application 3, Blind MACs and Blind Encryption: One can think of the operation of obtaining the AES encryption of a message, under the other party's secret key, as a blind MAC or a blind encryption. Currently one-time encryption or MACs are used in the setting of secure two-party computation in order to hide and authenticate an output of a circuit, where this output is sent to a party other than the circuit evaluator, or to a different circuit performing another computation (in particular, while implementing reactive computation). In that setting it must be decided in advance which circuit receives the output of each other circuit, in order to plug the same key into a circuit C and a circuit C' which receives encrypted or authenticated information from C. Computing the encryption or authentication with a key that can be used multiple times, rather than a one-time primitive, enables to arbitrarily route the output of circuits to other circuits, even in an adaptive order which is decided while the protocol is executed.

Application 4, Third Party Operations on Encrypted Data: We essentially show that secure evaluation of encryption and decryption can be implemented using circuits. This enables secure computation of homomorphic operations on encrypted data. This is done by a circuit which receives two ciphertexts from one party and a key from the other party, decrypts the ciphertexts, applies some arbitrary mathematical operation to the plaintexts, and then encrypts the result.

1.2 Paper Organization

The paper is organized as follows: In Section 2 we present the basics behind the Yao protocol that we will require. This section is deliberately compact and provides a sketch since the ideas presented here are covered better elsewhere. However we fix some notation and concepts which we will use later on.

In Section 3 we present a number of optimisations which reduce the effective size of the circuits. In Section 4 we describe an optimisation (due to [21]) which enables the evaluation of XOR gates with no communication and no computation overhead, assuming the underlying KDF used in the Yao construction is correlation robust. In this case we present an additional new optimisation which reduces the communication sent for all other gates by 25%. In Section 5 we describe another new technique based on secret sharing that can reduce the size of the garbled tables by 50 percent, but which cannot be applied together with the "free XOR" optimisation. This optimisation can be applied when one is unwilling to model the KDFs as correlation robust, or when the proportion of XOR gates within the circuit is not particularly high.

Finally in Section 6 we present our experimental results. We feel that the results we present are the most extensive results obtained so far in the case of secure two-party computation. In particular our experimental results are for a size of problem which is indicative of a magnitude which could have practical relevance, as described above Our experimental results point to a number of places in which further research needs to be carried out.

The appendices deal with a number of issues which are for the interested reader, but which would detract from the main thrust of our presentation.

2 Yao's Garbled Circuit Construction

Two-party secure function evaluation makes use of the famous garbled circuit construction of Yao [39] which we briefly overview in this section. The basic idea is to encode the function to be computed via a binary circuit and then to securely evaluate the circuit on the players' inputs.

2.1 Garbled Circuits

We consider two parties, denoted as P_1 and P_2, who wish to compute a function securely which is represented as a simple binary circuit. First assume the circuit consists of only a single gate with two input wires and one output wire. We denote the input wires by w_1 and w_2, and the output wire by w_3. The input to w_1 is denoted by b_1 and is known to P_1, similarly P_2 knows the input to w_2 and this is given by b_2. Each gate has a unique identifier Gid; this enables a circuit fan out of greater than one, i.e., it enables the output wire of one gate to be used in more than one other gate. We require that P_2 evaluates the gate on the two inputs, without P_1 learning anything, and without P_2 determining the value b_1, bar what it can deduce from the output of the gate and its own input. We define the output of the gate by the function $G(b_1, b_2) \in \{0, 1\}$.

The construction of Yao works as follows. P_1 encodes, or garbles, each wire w_i by selecting two different cryptographic keys k_i^0 and k_i^1 of length t. Here t is a computational security parameter which suffices for the length of a symmetric encryption scheme. A random permutation π_i of $\{0, 1\}$ is associated to each wire. The garbled value of wire w_i is then represented by $k_i^{b_i} \| c_i$, where $c_i = \pi_i(b_i)$. We call the value c_i the "external value" of the wire, note that this value is completely independent of the actual value of the wire b_i.

An encryption function $E^s_{k_1,k_2}(m)$ is selected which has as input two keys of length t, a message m, and some additional information s. The additional information s must be unique per invocation of the encryption function, i.e., it is used only once for any choice of keys. The gate itself is then replaced by a four entry table indexed by the values of c_1 and c_2, and given by

$$c_1, c_2 : E^{\mathrm{Gid}\|c_1\|c_2}_{k_1^{b_1}, k_2^{b_2}} \left(k_3^{G(b_1,b_2)} \| c_3 \right),$$

where $c_1 = \pi_1(b_1)$, $c_2 = \pi_2(b_2)$, and $c_3 = \pi_3(G(b_1, b_2))$. Each entry in the table corresponds to a combination of the values of the input wires and contains the encryption of the corresponding garbled output value. The resulting look up table, or set of look up tables in general, is called the "garbled circuit".

Player P_1 then sends to P_2 the garbled circuit, the key corresponding to its input value $k_1^{b_1}$, the value $c_1 = \pi_1(b_1)$, and the permutation π_3. The parties engage in an oblivious transfer (OT) protocol so that P_2 learns the value of $k_2^{b_2} \| c_2$, where $c_2 = \pi_2(b_2)$. Player P_2 can then decrypt the entry in the look up table indexed by (c_1, c_2) using $k_1^{b_1}$ and $k_2^{b_2}$; revealing the value of $k_3^{G(b_1,b_2)} \| c_3$. P_2 determines the value of $G(b_1, b_2)$ by using the mapping π_3^{-1} from c_3 to $\{0, 1\}$.

In the general case the circuit consists of multiple gates. Player P_1 chooses random garbled values for all wires and uses them for constructing tables for all gates. It sends these tables, i.e., the garbled circuit, to P_2 and in addition provides P_2 with the garbled values and the c values of P_1's inputs, and with the permutations π used to encode the *output* wires of the circuit. Player P_2 uses invocations of oblivious transfer to learn the garbled values and c values of its own inputs to the circuit. Given these values, P_2

can evaluate the gates in the first level of the circuit, compute the garbled values and the c values of their output wires. Player P_2 can then continue with this process and compute the garbled values of all wires in the circuit. Finally P_2 uses the π permutations of the output wires of the circuit to compute the real output values of the circuit. If P_1 additionally requires some output from the circuit then this can be dealt with by standard mechanisms, as described in Appendix C.

One could use more general gates than 2-to-1 gates, such as n-to-m gates with 2^n entries. However the optimisations we shall present in this paper are most effective when applied to 2-to-1 gates. While we found that more general gates can improve the performance of a naive Yao circuit protocol, they actually decrease the performance of the optimisations. Hence the rest of this paper is restricted to 2-to-1 gates.

2.2 Required Implementation Details

Having described the basic theoretical description of Yao's protocol and its extensions, we now present a number of implementation details which are needed to understand some of our optimisations. The basic implementation choice of the underlying encryption scheme to be used is the same as the implementation described in [25].

Oblivious transfer: Unlike [25] we do not use the OT scheme of Hazay and Lindell (HL) [17]. Instead we use the OT scheme of Peikert et al. (PVW)[35]. This scheme is UC-secure and hence requires the setup of a Common Reference String (CRS) of a few hundred bits. For our experiments we assume that this is given to the parties. (Alternatively, the parties can run a coin-tossing protocol to generate the CRS, which is possible due to the nature of the CRS used in the PVW scheme.) The batched method of PVW is more efficient per OT than the batched method of HL, especially on the receiver's side. In particular the CRS can be used for any number of invocations of the OT, whereas the method in HL requires the maximum number of OT's being executed to be known before the setup is performed. (The setup in HL also requires two ZK-proofs as opposed to a CRS being created in PVW.) The OT stage is not our computational bottleneck, and is unlikely to be, unless one is in the rare situation of having a circuit with a large number of inputs for P_2 and yet a relatively small number of gates. Thus we do not consider optimisations of OT schemes which are secure against only semi-honest or covert adversaries, since the fully secure OT is efficient enough.

Encryption scheme: The only implementation detail we will need from [25] is that the encryption scheme is implemented via

$$E^s_{k_1,k_2}(m) = m \oplus \mathrm{KDF}^{|m|}(k_1, k_2, s)$$

where KDF is a key deriviation function, whose $|m|$ bits of output are independent of the two input keys in isolation, and which depends on the value of s. We will instantiate this function as follows[5]

$$\mathrm{KDF}^{\ell}(k_1, k_2, s) = H(k_1 \| s)_{1\ldots\ell} \oplus H(k_2 \| s)_{1\ldots\ell}.$$

As subsequently pointed out in the attack and fixes in [33], for this instantiation of the KDF it is very important that the left and right call to H differ, either by requiring that $k_1 \neq k_2$ (by ensuring that the two input wires to a gate are different) or appending a 0 to the left call and a 1 to the right call.

Even if H is a Merkle–Damgård type hash function this will be secure (with the associated issues of length extension), since we are only applying the function to fixed length inputs. Indeed, in our experiments we implement H using SHA-256.

[5] In [25] two instantiations were presented, depending on whether we are working in the random oracle model (ROM) or standard model, via truncating, or extending, the output of a suitable hash function H in the standard way as follows

$$\mathrm{KDF}^{\ell}(k_1, k_2, s) = \begin{cases} H(k_1 \| k_2 \| s)_{1\ldots\ell} & H \text{ is modeled as an RO,} \\ H(k_1 \| s)_{1\ldots\ell} \oplus H(k_2 \| s)_{1\ldots\ell} & H \text{ is modeled as a PRF.} \end{cases}$$

The difference is that the security analysis in the ROM works even if we feed related keys to different invocations of the function. Namely, it is possible to compute, say, $H(k_1\|k_2), H(k_1\|k_2'), H(k_1'\|k_2)$ and $H(k_1'\|k_2')$ and claim that knowledge of k_1, k_2 does not disclose information about any of the values except $H(k_1\|k_2)$. This is impossible in the standard model. Therefore if $H()$ is modeled as a prf it must be invoked separately with each key.

Modeling the hash function, and correlation robustness: In this paper we need to model the underlying hash function H in two ways. In the first we make the usual assumption that it behaves as a pseudo-random function, namely that $H(k\|s)$ is an invocation of a pseudo-random function keyed by k, with the input s. However one of our optimisations requires that we make a stronger assumption on the hash function, namely that it is correlation robust. This later property can be stated formaly as follows:

Definition 1 (Correlation robustness [18]). *An efficiently computable function $H : \{0,1\}^* \to \{0,1\}^\ell$ is correlation robust if the following distribution is pseudo-random: $(t_1, \ldots, t_m, H(t_1 \oplus r), \ldots, H(t_m \oplus r))$, where $t_1, \ldots, t_m$ and r are chosen at random, and m is polynomial in the security parameter.*

This can also be stated by saying that the function $f_r(x) = H(x \oplus r)$ is a weak pseudo-random function. The definition also implies that the distribution of $(H(t_1), \ldots, H(t_m), H(t_1 \oplus r), \ldots, H(t_m \oplus r))$ is pseudo-random.

The correlation-robustness assumption is satisfied by a random oracle (or rather by a very weak form of it: a non-programmable, non-extractable random oracle). However, assuming correlation robustness seems as a much weaker requirement than assuming the existance of random oracles. In particular, this assumption is falsifiable according to the definition of Naor [28], meaning that given a candidate function H it is possible to design a challenge which can be broken if H is not correlation robust, whereas it is impossible to design a similar challenge with regards to the modeling of H as a random oracle. This assumption has been introduced in [18] and was used there for providing security against malicious adversaries for a method of extending oblivious transfer. The correlation robustness assumption has been recently used in the context of oblivious transfer [15, 19] and in the context of secure computation [21, 29].

For our construction, as we deal with circuits with arbitrary fan out, we require a slightly modified definition. Namely that for any set $S = \{s_1, \ldots, s_{|S|}\}$ of size which is of the same order as the number gates, the distribution of $(t_1, \ldots, t_m, \langle H((t_1 \oplus r)\|s_1), \ldots, H((t_m \oplus r)\|s_1))\rangle, \langle H((t_1 \oplus r)\|s_2), \ldots, H((t_m \oplus r)\|s_2))\rangle, \ldots, \langle H((t_1 \oplus r)\|s_{|S|}), \ldots, H((t_m \oplus r)\|s_{|S|}))\rangle$ is pseudo-random, where $t_1 \ldots, t_m$ and r are chosen at random. In other words, all the pads that are used for encrypting table entries are pseudo-random. If one is willing to assume this then our optimisations provide highly efficient protocols. We also provide optimisations for when the user is unwilling to make such an assumption.

3 Structural Optimisations of the Circuit

Yao's protocol operates on functions which are described as a boolean circuit, and its overhead depends on the size of the circuit. A convenient way of generating a representation of a function in this form is to use a compiler which translates a description of a function in a high-level language to a description as a binary circuit. The Fairplay system provides a compiler for this task which operates on functions described in a high-level language called Secure Function Description Language (SFDL) [27, 4]. We use that compiler as the basis of our experiments, but use our own run-time environment to execute the protocol.

There are a number of general circuit simplifications which can be performed to the output of the Fairplay compiler. We have implemented a number of these, based on two basic ideas: (1) identifying component circuits which can be replaced by simpler combinations of gates, and (2) identifying complicated components whose output must always be zero, or one; this allows for the component to be removed and other subsequent components to be further simplified. A combination of these techniques is surprisingly effective, and allows us to produce circuits which are often 60 percent more efficient than the circuit produced by the Fairplay compiler.

Many of the techniques used are ad-hoc, but the following technique is particularly effective. First, by a technique akin to common sub-expression elimination, we identify sets of gates which can be replaced by a single 3-to-1 gate, and then replace the 3-to-1 gate with a set of 2-to-1 gates which was chosen to minimize the number of non-XOR gates. This is particularly effective when combined with our later technique of Section 4, in the case of correlation robust KDFs, to remove the cost of any XOR gates;

however the technique is also successful in the more general case as well. We call a gate even if its truth table has an even number of '1' entries (for example, a XOR gate is even), otherwise it is called odd (an OR gate, for example, is odd). We show in Appendix A that it is possible to replace any 3-to-1 even gate with at most a single 2-to-1 non-XOR gate and at most three XOR gates. The optimal transformation rules, which we found by exhaustive search, are listed in Appendix A.

4 Optimisations with Free XORs, when the KDF is Correlation Robust

In [21] Kolesnikov and Schneider present an optimisation based on the correlation robustness assumption, which allows XOR gates to be evaluated for free, thus doing away with the need to evaluate or transmit the garbled tables for such gates. The optimisation requires that there is a global random value R of bit length t, known only to P_1, such that for all garbled wires w_i it holds that $k_i^1 = k_i^0 \oplus R$. In other words, the garbling of the 1 value of a wire, is determined purely from XOR-ing the garbled 0 value with the value R. Note that a similar property holds for the external values of the wire: $\pi_i(1) = \pi_i(0) \oplus 1$. With this convention we have that a XOR gate can be implemented by simply XOR-ing together the two garbled input values, and the two external values. Namely, for a XOR gate mapping wires w_1 and w_2 to wire w_3, it holds that $k_3 = k_1 \oplus k_2$ and $c_3 = c_1 \oplus c_2$. For a full proof of this optimisation see [21]. Note that [21] states the proof in the random oracle model, but it can be easily seen, as noted in [21], that the proof can be based on the correlation robustness assumption.

Garbled Row Reduction – GRR: The above solution is ideal for XOR gates, but in addition we would like to reduce the size of the tables of the non-XOR gates as well. The following simple optimisation (which was pointed out in [31]) provides a 25 percent reduction in the sizes of the tables needed to represent two-input gates. We can do this in a way which still allows the use of the above trick for free XOR gates. (In general, this method provides a $1/2^n$ reduction in the size of n-to-1 gates, but we will only describe it in detail for the two input case.)

The observation is that instead of defining the two garbled values of the output wires randomly, we can define one of them as a function of garbled values of the two input wires which result in this output value. In other words, we choose an input pair $(b_1, b_2) \in \{0,1\}^2$, and define the garbled output value of $G(b_1, b_2)$ to be a function of the garbled values of b_1 and b_2. The gate table therefore need not store an entry for the input combination (b_1, b_2). In the evaluation phase, if the evaluator has the garbled values of the pair (b_1, b_2) it can compute the corresponding garbled output directly, without consulting the gate table.

Suppose the gate maps wire w_1 and wire w_2 to wire w_3. As before we let k_i^0 and k_i^1 denote the garbled wire values, $G(b_1, b_2)$ denote the function being implemented by the gate, and we set the external value of the wire to be $c_i = \pi_i(b_i)$. We then define the garbled output value corresponding to the output resulting from the external input values $(c_0, c_1) = (0, 0)$ as

$$k_3^{G(\pi_1^{-1}(0),\pi_2^{-1}(0))} \| c_3 = \mathrm{KDF}^{t+1}\left(k_1^{\pi_1^{-1}(0)}, k_2^{\pi_2^{-1}(0)}, \mathrm{Gid} \| 0 \| 0\right).$$

In other words, the garbled value is exactly equal to the pseudo-random mask that was used to hide it in the basic protocol. Note that this operation also defines the external value c_3 of this output value. We therefore define π_3 such that $c_3 = \pi_3(G(\pi_1^{-1}(0), \pi_2^{-1}(0)))$. The other garbled value of the output wire, $k_3^{1-G(\pi_1^{-1}(0),\pi_2^{-1}(0))}$ is then chosen as in the free XOR method above, to enable the evaluation of XOR gates for free. The table is then constructed in the standard way except that we do not store, or transmit, its first entry.

On evaluating the garbled gate the evaluator proceeds as in the standard algorithm except when it wishes to access the first entry of the table, i.e., when the external values of both input wires are 0, namely $c_1 = c_2 = 0$. In that case it possesses the garbled values $k_1^{b_1}$ and $k_2^{b_2}$, where $b_1 = \pi_1^{-1}(0)$ and $b_2 = \pi_2^{-1}(0)$. It uses them to compute $k_3^{G(b_1,b_2)}$ and $c_3 = \pi_3(G(b_1, b_2))$, by computing $\mathrm{KDF}^{t+1}\left(k_1^{b_1}, k_2^{b_2}, 0 \| 0 \| \mathrm{Gid}\right)$ as defined in the equation above.

We will denote this optimisation as Garbled Row Reduction, GRR for short, in our future discussions.

Security: We sketch why the above optimisation maintains security. Recall that the proof of security for Yao's protocol given in [24] shows security against a corrupt P_2 based on a hybrid argument, and on a claim that for each gate it is infeasible to distinguish between a correct garbled table of this gate and a table which encrypts the same value in all four entries. In order for this argument to apply to the GRR optimisation, it is required to show that it is infeasible to find out if the garbled value assigned to the first table entry, $k_3^{G(\pi_1^{-1}(0),\pi_2^{-1}(0))}\|c_3$ is equal to the values encrypted in the other entries. However this value is equal to the mask that is used to encrypt the first entry in Yao's original protocol, and we know that if a polynomial adversary is given only a single pair of garbled input values then the masks that are used for encrypting the other entries of the table are pseudo-random. Therefore the claim follows.

5 Optimisations without Free Xors, when the KDF is not Correlation Robust

One may not want to assume the KDF is correlation robust, or perhaps the proportion of XOR gates in the circuit is so low that making this assumption is not as effective. In these situations, too, we would like to reduce the overhead required by the Yao circuit. This section describes an optimisation which reduces the size of every two-input gate by 50%, but which, unfortunately, cannot be combined with the free XOR method of Section 4.

The underlying idea is that if we are not using the free XOR trick then the two values of the output wire can be chosen independently.[6] The 50% reduction in the size of the gate tables is based on Shamir secret sharing [37]. It makes use of a finite field $\mathbb{F}_{2^t}$. Recall that t is the bit length of the keys used to represent the garbled values of the wires. We can therefore interpret keys as elements of $\mathbb{F}_{2^t}$ and vice versa. We also interpret small integers such as 1, 2, 3 etc. as elements in $\mathbb{F}_{2^t}$. For example if we think of $\mathbb{F}_{2^t}$ as $\mathbb{F}_2[X]/(f(X))$, for some polynomial of degree t, then the integer 3 can be interpreted as $x+1$.

As before we assume a garbled table indexed by the external values, c_1 and c_2, and each entry corresponds to the value being output, on input of the values $k_1^{b_1}$ and $k_2^{b_2}$ where $b_i = \pi_i^{-1}(c_i)$. We set the rows of the gate table to be numbered $1, \ldots, 4$, and therefore set $r = 2c_1 + c_2 + 1$ to be the row number of table entry (c_1, c_2). We define the value used to mask this entry as

$$K_r||M_r = \mathrm{KDF}^{t+1}(k_1^{b_1}, k_2^{b_2}, s) \tag{1}$$

where $s = \mathrm{Gid}\|c_1\|c_2$, K_r is a bit string of length t bits and M_r is a single bit used to mask the external value of the output. We use a different method for optimising odd and even gates. The truth table of each gate, and therefore also the information whether the gate is odd or even, is known to the circuit evaluator. Therefore it can compute each gate according to the right method. (The only information hidden from the evaluator is the values passing on intermediate wires of the circuit.)

5.1 Odd 2-to-1 Gates

Suppose we are implementing an OR-gate, where the external values of $c_1 = 0$ and $c_2 = 0$ correspond to the real input values $(0, 0)$, the other cases will follow immediately from the following. This means that the values $r = 2, 3$ and 4 should evaluate to the same output value k_3^1, whilst $r = 1$ should evaluate to the output value k_3^0. We first define over $\mathbb{F}_{2^t}$ a polynomial $P(X)$ of degree two, by interpolating the polynomial which intersects the three points $(2, K_2)$, $(3, K_3)$ and $(4, K_4)$, where each K_r value was defined according to equation (1). (This is the value which in the other constructions was used to mask entry r of the table.) The garbled output value k_3^1 is defined to be $k_3^1 = P(0)$. We also compute $K_5 = P(5)$ and $K_6 = P(6)$. We then define a second polynomial $Q(X)$, also of degree two, by interpolating the polynomial which intersects the three points $(1, K_1)$, $(5, K_5)$ and $(6, K_6)$, where K_1 was defined according to equation (1). The garbled output value k_3^0 is now defined by $k_3^0 = Q(0)$. The garbled table is replaced by the two values (K_5, K_6). In addition, for each of the four original rows, the external value for the output wire in

[6] This allows for possible extensions of the GRR method, and in Appendix B we detail another optimisation method, which we call Garbled Table Reduction (GTR), which reduces the size for the garbled tables needed to represent odd 2-to-1 gates by 1/3, and the size of tables of even 2-to-1 gates by 1/2.

the rth row is encrypted using the bit M_r, defined in equation (1). The total amount of data sent for the gate is therefore $2t + 4$ bits.

Player P_2 then, given two key values $k_1^{b_1}$ and $k_2^{b_2}$ plus two external values c_1 and c_2, computes, using equation (1) the value of K_r and M_r for $r = 2c_1 + c_2 + 1$. Recall that the evaluator knows r but not b_1 or b_2. It then uses the two supplied values of K_5 and K_6 to interpolate the polynomial passing through the points $(r, K_r), (5, K_5)$ and $(6, K_6)$. The result is either $Q(X)$ or $P(X)$, depending on whether $r = 1$ or not. Player P_2 then recovers the associated secret value $k_3^{b_3}$, by evaluating the polynomial at the point $X = 0$. Using M_r the evaluator can also decrypt the encryption of the external value of the output wire and so obtains c_3. Hence the evaluator recovers the correct value of the output wire.

5.2 Even 2-to-1 Gates

The only non-trivial even 2-to-1 gates are the XOR and NXOR gate, since all other gates can be replaced by wires. Again let us assume the external input values $c_1 = 0$ and $c_2 = 0$ correspond to the real input values $(0, 0)$, and assume we are dealing with a XOR gate. Then the entries 1 and 4 in the standard garbled table will correspond to the same output key, namely $k_3^{\pi_3^{-1}(0)}$. Any other case will follow from the following description.

Player P_1 first creates a linear polynomial $P(X)$ over $\mathbb{F}_{2^t}$ which interpolates the two points $(1, K_1)$ and $(4, K_4)$. The value of $k_3^{\pi_3^{-1}(0)}$ is defined to be equal to $P(0)$. If the external value of this output value is 0 then we store $P(5)$ into the first row of the new table of this gate, otherwise we store $P(5)$ as the second entry. Then P_1 creates another linear polynomial $Q(X)$ which interpolates the two points $(2, K_2)$ and $(3, K_3)$. The value of $k_3^{\pi_3^{-1}(1)}$ is then defined to be $Q(0)$, and the value $Q(5)$ is stored in the remaining row of our new table. The external values of the output wires are now encrypted and stored, using the M_r values as before as a seperate sub-table of 4 bits in length. Thus, the total amount of data required to represent the gate is $2t + 4$ bits.

Player P_2 given two key values $k_1^{b_1}$ and $k_2^{b_2}$ plus two external values c_1 and c_2, computes the value of K_r and M_r. Using M_r it can determine the external value of the output wire. If this external value is zero then using the first entry of our garbled table and the value of K_r, the evaluator recovers $P(X)$ and hence $P(0) = k_3^{\pi_3^{-1}(0)}$. If the external value is one then using the second entry of the table and the value K_r, the evaluator recovers $Q(X)$ and hence $Q(0) = k_3^{\pi_3^{-1}(1)}$.

Security: We sketch why the above optimisations maintain security. Given a pair of garbled values of the input wires, P_2 can compute a garbled output value, but cannot distinguish the other garbled output value from random. This is because that other garbled value is defined using a linear combination with a value which is unknown to P_2. This fact can be used in a, somewhat modified, security proof in the spirit of the proof of Yao's protocol in [24].

6 Some Experimental Results

We now present some experimental results. In our results we separate out precomputation time, i.e., generating the required garbled circuits, from the rest of the computation. This is because it depends on the application whether one should consider this time as part of the computation time or not.

There are two major conclusions of our experiments. Firstly, assuming the KDF is correlation robust then the GRR optimisation produces the most efficient implementation. Secondly we conclude that rather large circuits can be practically evaluated using the methods described. Thus secure two-party computation has become more of a reality than one might previously have thought.

Example 1 – Evaluation a Simple Circuit: First we present results for a simple circuit, where we took the circuit for which each of P_1 and P_2's input is a 32-bit integer. The output for P_2 should be the single bit resulting from the application of the comparison operator on the inputs. The output for P_1 will

be a six bit integer resulting from the scalar product of the bits of the two inputs, i.e. the number of ones in the string obtained from forming the bit-wise "and" of the two strings.

Applying the Fairplay compiler to this functionality we obtain a circuit with 689 gates. We produce two circuits from this output; the first, denoted $C_{2,3}$, is to allow comparison with the existing state of the art, namely the methods of [25]. This is a circuit which uses 2-to-1 and 3-to-1 gates and has 245 gates. The second circuit we use, denoted C_{xor}, replaces, via the techniques of Section 3, all complex gates with 2-to-1 gates, and tries to minimise the number of non-XOR gates in the circuit. This circut has 531 gates, 240 of which are non-XOR gates. An extra six gates are needed in each circuit so as to encode P_1's for tranmission back to P_1, without P_2 learning the value.

The above circuit sizes are purely to implement the functionality, they do not include the extra wires and gates required to transmit P_1's output back to P_1 (for details of how this is done see Appendix C), nor do they include the extension of the circuit to cope with P_2's input in the case of Covert and Malicious adversaries. (We refer to the two methods for encoding P_2's input as the *independent inputs* and the *random combinations* methods. For the details of these methods see [23] or Appendix D.1). The sizes of the extended circuits, and the resulting run-times are given in Table 1, which measures the total elapsed wall times in seconds for the various cases.

The calculations were performed on two machines with Intel Core 2 Duo's running at 3.0 GHz, with 4GB of RAM connected by a 1GB ethernet. The hash function $H()$ used in the protocol was implemented as SHA-256.

Table 1. Experimental Results For Example 1 (Times are in seconds)

Adv.	Input Enc.	Method	No. Gates	% XOR Gates	Precomp Time	Send Time	OT Time	Calc Time	Total Time	Total KBytes
Semi-Honest		Base	251	11	0	0	2	0	2	46
		PRF-SS	537	55	0	0	1	0	1	34
		CoR-GRR	537	55	0	0	1	0	1	22
		ROM-GRR	537	55	0	0	1	0	1	22
Covert	Indep. Inputs	Base	419	38	7	1	4	6	18	1188
		PRF-SS	705	61	8	0	2	7	17	969
		CoR-GRR	705	61	6	1	3	5	15	682
		ROM-GRR	705	61	1	1	2	0	4	629
Covert	Random Comb.	Base	1247	79	9	2	4	7	22	2275
		PRF-SS	1535	82	9	1	3	7	20	1646
		CoR-GRR	1555	82	7	1	3	5	16	682
		ROM-GRR	1555	82	1	1	3	0	5	629
Malic.	Indep. Inputs	Base	1571	83	171	80	47	54	352	180599
		PRF-SS	1857	85	175	79	39	67	360	173942
		CoR-GRR	1857	85	147	78	37	39	301	164323
		ROM-GRR	1857	85	141	71	37	38	287	161741
Malic.	Random Comb.	Base	3029	89	163	75	19	64	321	167276
		PRF-SS	2799	90	161	74	16	69	320	158904
		CoR-GRR	2781	90	117	75	16	39	247	140265
		ROM-GRR	2802	90	117	69	16	37	239	137609

The column of "Total KBytes" contains the total number of kilobytes of data which were transferred during the run of the protocol. The column "Method" details the type of computation used, as follows:

- Base: Denotes the optimisations proposed in [25], extended to the case of Covert and Honest adversaries, which we use for comparison purposes, as our baseline implementation. This uses the $C_{2,3}$ circuit mentioned above, the KDF which is secure in the standard model, and the OT of Hazay-Lindell [17] as opposed to that of Peikert et al. [35].
- PRF-SS: This denotes using the secret sharing based method of Section 5, to reduce the size of the garbled tables. For this the KDF is assumed to be a PRF, but not correlation robust.

– CoR-GRR: This denotes an implementation which is only secure assuming the KDF is correlation robust. It uses the free XOR trick and the method of Garbled Row Reduction, from Section 4, to reduce the size of the remaining garbled tables.
– ROM-GRR: As above for CoR-GRR but all hash functions used are modelled as random oracles. This means we can implement our KDF via a single hash function call, based on the method described in Footnote 5.

The column denoted "No. of gates" describes the number of gates, and the percentage of XOR gates, in the extended circuit (which transfers P_1's outputs and applies the extension described in Appendix D.1, encoding P_2's input).

For the Covert and Malicious cases the "Input Enc." column denotes whether we use the Independent Inputs technique or the Random Combinations technique for the extended circuit construction. See Appendix D for details, as well as for explanation of the following choice of parameters. In the covert case we selected $s_1 = 16$ and $s_2 = 4$ since we require $s_2 \approx \log_2 s_1$, so as to balance the probabilities of cheating for the two ways in which player one can cheat. For a similar reason we use $s_1 = 160$ and $s_2 = 40$ in the case of malicious adversaries, where we require, conjecturally, $s_2 = s_1/4$. From the table we can deduce the following conclusions:

– The running time in the semi-honest setting is about 10-20 times faster than in the covert setting, which is in turn about 15-20 times faster than in the malicious setting.
– For the Covert case the choice of the Independent Inputs technique for the extension of the circuit is slightly better due to the small value of s_2, whilst for the Malicious case the technique of Random Combinations is clearly better.
– A lot of the extra data needed to be transmitted in the Malicious case is related to the large number of commitments and decommitments which need to be transmitted. Thus our optimisation techniques are less effective in the Malicious case. This points to a clear direction for future research in optimising the Malicious case.
– If one is not willing to assume that the KDF is correlation robust we see that using our technique based on secret sharing can reduce the amount of data being transmitted, compared to the base scheme, without increasing the computational cost.
– In all cases we see that the correlation robust variant using Garbled-Row-Reduction is the most efficient variant. The extra efficiency comes from the free XOR's which reduce both the number of encryption/decryptions which need to be performed and also the amount of data needing to be transmitted.
– Note that if we assume the random oracle model, and so could implement our KDF via a single hash function call then for Covert adversaries the protocols run significantly faster. That this does not apply as much to the Malicious case is due to the fact that most of the time in the Malicious case is spent with creating, sending and verifying the various commitments.

We pause to compare our two optimisations with the optimisation in bandwidth suggested in [14]. In our system P_1, the circuit constructor, sends commitments to all circuits that it constructs and to its own inputs, and a random subset of these committed values are checked by P_2. In [14] it is suggested that P_1 commits to a random seed, and uses this to generate the circuit. Then only the commitment to this seed, and eventually its decommitment, need to be transmitted. This means that P_2 needs to compute the circuit given the seed. Whilst this optimisation clearly significantly reduces the consumed bandwidth, it actually leads to a significant increase in the time needed to perform the protocol. To see this consider our Covert experiments in Table 1. The optimisation in [14] would reduce practically to zero, the entry for the "Send Time" column, but P_2 would now need to recompute almost all of the calculations in the "Precomp Time" column. Thus the technique of [14] is only to be compared to ours in the situation where bandwidth is very expensive and CPU time is very cheap.

Before passing onto our larger example we note the following. If we let p denote the proportion of XOR gates within a circuit, and we let N denote the amount of data needed to be sent per circuit in the standard Yao construction, then the average amount of data needed to be sent per circuit gate when

using the free XOR gates and GRR methods is $3/4 \cdot (1-p) \cdot N$. Whereas if we do not use the free XOR gate method and instead use the method based on secret sharing, this value becomes $N/2$. Hence, if we are willing to assume correlation robust KDFs, then the method which uses secret sharing and does not use the free XOR method, will be more efficient as long as the fraction of XOR gates, p, is smaller than 1/3. However as can be seen from the column entitled "% XOR Gates", this proportion is generally much larger than 1/3, especially in the case of Covert and Malicious adversaries where we have had to extend the circuit by a large linear component. This expansion is performed to cope with possible adversarial behaviour related to P_2's input, see the full version for details. One should note that these theoretical estimates of bandwidth are never achieved fully in practice due to overheads in the underlying data transmission mechanism and the fact that they assume a bit-oriented communication mechanism, whereas practical communication is performed in bytes. Hence the saving we achieve in gate transmission is about 5-10% less than one would predict purely by theory.

Example 2 - Evaluating AES: As our second example we created a circuit which computes an AES encryption of a single 128-bit block with respect to a 128-bit key. Here P_1's input is the secret key, and P_2's input is the message block. We require that P_2 learns the encryption of its message under P_1's secret key, and that P_1 learns nothing. Compiling such a circuit using the Fairplay compiler, and applying various optimisations, resulted in a circuit, which we denote by $C_2^{(1)}$, with 33880 gates, where each gate is a 2-to-1 gate. This circuit was derived in a way to try to minimize the number of non-XOR gates. Again, we stress, the above circuit size purely implements the AES functionality, it does not include the extension of the circuit to cope with P_2's input in the case of Covert and Malicious adversaries. Note that the key schedule takes up only about 15% of the circuit, hence encrypting a sequence of message blocks as in CBC-Mode encryption will scale almost linearly with respect to our data.

We repeated our experiments from above, but in Table 2 we only present the times for the most efficient choice for the input encoding. Thus we only look at the Independent Inputs technique for Covert adversaries, and the Random Combinations technique for Malicious adversaries.

Table 2. Experimental Results for Example 2 (Again times are in seconds)

Adv.	Input Enc.	Method	No. Gates	% XOR Gates	Precomp Time	Send Time	OT Time	Calc Time	Total Time	Total KBytes
Semi-Honest		Base	28216	56	5	2	4	3	14	3162
		PRF-SS	33880	66	5	1	3	3	12	1752
		CoR-GRR	33880	66	2	1	2	2	**7**	503
		ROM-GRR	33880	66	1	1	3	2	7	503
Covert	Indep. Inputs	Base	28600	56	96	47	18	45	206	51899
		PRF-SS	34264	67	92	36	13	50	191	29380
		CoR-GRR	34264	67	40	21	11	23	**95**	9078
		ROM-GRR	34264	67	22	21	11	6	60	8942
Malic.	Random Comb.	Base	40253	69	1250	448	39	887	2624	987442
		PRF-SS	45944	75	1184	392	34	829	2439	711729
		CoR-GRR	45960	75	483	270	34	361	**1148**	406010
		ROM-GRR	45881	75	453	276	35	350	1114	417907

We conclude that performing the Yao protocol is certainly feasible on complicated functionalities such as AES encryption. For the case of honest and covert adversaries we again see that the computation and bandwidth consumed, when we use correlation robust KDFs and the GRR method, greatly reduces in comparison to the base case. If one is not willing to assume correlation robust KDFs (or use the ROM) then our secret sharing based optimisation greatly reduces the bandwidth without affecting the run times. For the malicious case the improvement in the secret sharing based version is less pronounced due to the large number of commitments which need to be transmitted and opened. This clearly points to the place where future optimisation research needs to be performed, namely in reducing the number of commitments needed in the situation of malicious adversaries. However even without such future

optimisation we note that performance can be significantly reduced by taking advantage of the inherent parallelism in the algorithm in the Malicious case (in which P_1 generates many commitments and P_2 verifies a subset of them). For web service or cloud computing applications, where server farms are common place, an improvement in computational time by a factor around s_1 could be expected.

We end by noting that many application domains of a secure evaluation of AES, for example the one-time program example mentioned earlier from [13], require only security against semi-honest adversaries. Hence, such applications are already within the reach of practical realisation. Furthermore, this application requires no computation of the OT or data to be sent. Thus the party generating the one-time-program will take the time needed in our *Precomp Time* column, and the evaluator (after querying the one-time-memory) will take the time needed in the *Calc Time* column.

7 Acknowledgments

All authors would like to thank the EU projects CACE and eCrypt-2 for partially funding the work in this paper. The first author was also partially funded by the ERC project SFEROT. The third author was partially funded by a Royal Society Wolfson Merit Award, and the fourth author was partially funded by EPSRC. We would also like to thank Benny Applebaum, Yehuda Lindell and Ahmad-Reza Sadeghi for various comments and discussions.

References

1. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference – TCC 2007*, Springer-Verlag LNCS 4392, 137–156, 2007.
2. D. Beaver, Correlated pseudorandomness and the complexity of private computations. In *28th STOC*, 479–488, 1996.
3. D. Beaver, S. Micali and P. Rogaway. The round complexity of secure protocols. In *22nd STOC*, 503–513, 1990.
4. A. Ben-David, N. Nisan and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *Computer and Communications Security – CCS 2008*, ACM, 257–266, 2008.
5. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th STOC*, 1–10, 1988.
6. P. Bogetoft, D.L. Christensen, I. Dåmgard, M. Geisler, T. Jakobsen, M. Krøigaard, J.D. Nielsen, J.B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach and T. Toft, Secure multiparty computation goes live, To appear in *Financial Cryptography and Data Security – FC 2009*.
7. P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter. and T. Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Cryptography and Data Security – FC 2006*, Springer-Verlag LNCS 4107, 142–147, 2006.
8. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, **13(1)**, 143–202, 2000.
9. D. Chaum, C. Crepeau and I Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, 11–19, 1988.
10. M.J. Freedman, Y. Ishai, B. Pinkas and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference – TCC 2005*, Springer-Verlag LNCS 3378, 303–324, 2005.
11. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge Univ. Press, 2004.
12. O. Goldreich, S. Micali and A. Wigderson. How to play any mental game – A completeness theorem for protocols with honest majority. In *19th STOC,* 218–229, 1987.
13. S. Goldwasser, Y.T. Kalai and G.N. Rothblum. One-time programs. In *Advances in Cryptology – Crypto 2008*, Springer-Verlag LNCS 5157, 39–56, 2008.
14. V. Goyal, P. Mohassel and A. Smith. Efficient two party and multi-party computation against covert adversaries. In *Advances in Cryptology – EuroCrypt 2008*, Springer-Verlag LNCS 4965, 289–306, 2008.
15. D. Harnik, Y. Ishai, E. Kushilevitz and J.B. Nielsen. OT-Combiners via secure computation. In *Theory of Cryptography Conference – TCC 2008*, Springer-Verlag LNCS 4948, 393–411, 2008.
16. C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference – TCC 2008*, Springer-Verlag LNCS 4948, 155–175, 2008.
17. C. Hazay and Y. Lindell. Oblivious transfer, polynomial evaluation and set intersection. Manuscript, 2008
18. Y. Ishai, J. Kilian, K. Nissim and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – Crypto 2003*, Springer-Verlag LNCS 2729, 145–161, 2003.
19. Y. Ishai , M. Prabhakaran and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – Crypto 2008*, Springer-Verlag LNCS 5157, 572–591, 2008.
20. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology – EuroCrypt 2007*, Springer-Verlag LNCS 4515, 97–114, 2007.
21. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming – ICALP 2008*, Springer-Verlag LNCS 5126, 486–498, 2008.

22. S. Laur, H. Lipmaa and T. Mielikäinen. Private itemset support counting. In *Information and Communications Security – ICICS 2005*, Springer-Verlag LNCS 3783, 97–111, 2005.
23. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EuroCrypt 2007*, Springer-Verlag LNCS 4515, 52–78, 2007.
24. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, **22**, 161–188, 2009
25. Y. Lindell, B. Pinkas, N.P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks – SCN 2008*, Springer-Verlag LNCS 5229, 2–20, 2008.
26. P. Mohassel and M.K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography – PKC 2006*, Springer-Verlag LNCS 3958, 458–473, 2006.
27. D. Malkhi, N. Nisan, B. Pinkas and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. of 13th USENIX Security Symposium*, 2004.
28. M. Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology – CRYPTO 2003*, Springer-Verlag LNCS 2729, 96–109, 2003.
29. J.B. Nielsen and C. Orlandi. LEGO for two party secure computation. In *Theory of Cryptography Conference – TCC 2009*, LNCS 5444, 368-386, 2009.
30. M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology – CRYPTO '99*, Springer-Verlag LNCS 1666, 573–590, 1999.
31. M. Naor, B. Pinkas and R. Sumner. Privacy Preserving Auctions and Mechanism Design. In Proc. of the 1st ACM conf. on Electronic Commerce, November 1999.
32. M. Naor and O. Reingold, Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, **51**, 231–262, 2004.
33. R. Nieminen and T. Schneider. Breaking and Fixing Garbled Circuits when a Gate has Duplicate Input Wires. *Journal of Cryptology*, 2023.
34. B. Pinkas, T. Schneider, N. P. Smart and S. C. Williams. Secure Two-Party Computation is Practical. In *Advances in Cryptology – Asiacrypt 2009*, Springer-Verlag LNCS 5912, 250–267, 2009.
35. C. Peikert, V. Vaikuntanathan and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – Crypto 2008*, Springer-Verlag LNCS 5157, 554–571, 2008.
36. H. Ringberg, B. Applebaum, M.J. Freedman, M. Caesar and J. Rexford. Collaborative, privacy-preserving data aggregation at scale. manuscript, 2009.
37. A. Shamir. How to share a secret. *Communications of the ACM*, **11**, 612–613, 1979.
38. D. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Adv. in Cryptology – EuroCrypt 2007*, Springer-Verlag LNCS 4515, 79–96, 2007.
39. A. Yao. How to generate and exchange secrets. In *27th FOCS*, 162–167, 1986.

A Even 3-to-1 Gates

We replace even 3-to-1 gates with inputs a, b, c by at most one 2-to-1 gate and at most three XOR gates. It is easy to see that the optimal topology for replacing a non-trivial even 3-to-1 gate (written in postfix notation) is

$$(((\lambda)XOR, (\mu)XOR)[\tau]), \nu)XOR,$$

where $\lambda, \mu, \nu \subseteq \{a, b, c\}$, and τ is the gate table of an odd 2-to-1 gate. Based on this structure we computed by brute-force enumeration of all $(2^3)^3 \cdot 4^2 = 8192$ possibilities the optimal replacement for non-trivial even 3-to-1 gates.

In the following, the optimal replacements are listed. The first column is the even gate table, i.e., the ith element of the string is equal to the value of the function evaluated by the gate $G(a, b, c)$ for $i = 4a + 2b + c$. The second column is the corresponding functionally equivalent replacement in postfix notation. For example the bitstring 00001010 represents the function $a \wedge (\neg c)$ which can be represented by the 2-to-1 gate with the binary string representation 0100 with inputs c and a. Thus 00001010 is equivalent to $(c, a)[0100]$.

```
00000000 0
00000011 (b,a)[0001]
00000101 (c,a)[0001]
00000110 ((b,c)[0110],a)[0001]
00001001 ((b,c)[0110],a)[0100]
00001010 (c,a)[0100]
00001100 (b,a)[0100]
00001111 a
00010001 (c,b)[0001]
00010010 (b,(a,c)[0110])[0001]
00010100 (c,(a,b)[0110])[0001]
00010111 (((b,c)[0110],(a,c)[0110])[0001],c)[0110]
00011000 ((b,c)[0110],(a,c)[0110])[0100]
00011011 ((c,(a,b)[0110])[0001],a)[0110]
00011101 ((b,(a,c)[0110])[0001],a)[0110]
00011110 ((c,b)[0001],a)[0110]
```

```
00100001 (b,(a,c)[0110])[0010]
00100010 (c,b)[0100]
00100100 ((b,c)[0110],(a,c)[0110])[0010]
00100111 ((c,(a,b)[0110])[0001],b)[0110]
00101000 (c,(a,b)[0110])[0100]
00101011 (((b,c)[0110],(a,c)[0110])[0100],b)[0110]
00101101 ((c,b)[0100],a)[0110]
00101110 ((b,(a,c)[0110])[0010],a)[0110]
00110000 (b,a)[0010]
00110011 b
00110101 (((b,c)[0110],a)[0001],b)[0110]
00110110 ((c,a)[0001],b)[0110]
00111001 ((c,a)[0100],b)[0110]
00111010 (((b,c)[0110],a)[0100],b)[0110]
00111100 (a,b)[0110]
00111111 (b,a)[0111]
01000001 (c,(a,b)[0110])[0010]
01000010 ((b,c)[0110],(a,c)[0110])[0001]
01000100 (c,b)[0010]
01000111 ((b,(a,c)[0110])[0001],c)[0110]
01001000 (b,(a,c)[0110])[0100]
01001011 ((c,b)[0010],a)[0110]
01001101 (((b,c)[0110],(a,c)[0110])[0001],a)[0110]
01001110 ((c,(a,b)[0110])[0010],a)[0110]
01010000 (c,a)[0010]
01010011 (((b,c)[0110],a)[0001],c)[0110]
01010101 c
01010110 ((b,a)[0001],c)[0110]
01011001 ((b,a)[0100],c)[0110]
01011010 (a,c)[0110]
01011100 (((b,c)[0110],a)[0100],c)[0110]
01011111 (c,a)[0111]
01100000 ((b,c)[0110],a)[0010]
01100011 ((c,a)[0010],b)[0110]
01100101 ((b,a)[0010],c)[0110]
01100110 (b,c)[0110]
01101001 ((a,b)[0110],c)[0110]
01101010 ((b,a)[0111],c)[0110]
01101100 ((c,a)[0111],b)[0110]
01101111 ((b,c)[0110],a)[0111]
01110001 (((b,c)[0110],(a,c)[0110])[0001],b)[0110]
01110010 ((c,(a,b)[0110])[0010],b)[0110]
01110100 ((b,(a,c)[0110])[0010],c)[0110]
01110111 (c,b)[0111]
01111000 ((c,b)[0111],a)[0110]
01111011 (b,(a,c)[0110])[0111]
01111101 (c,(a,b)[0110])[0111]
01111110 ((b,c)[0110],(a,c)[0110])[0111]
10000001 ((b,c)[0110],(a,c)[0110])[1000]
10000010 (c,(a,b)[0110])[1000]
10000100 (b,(a,c)[0110])[1000]
10000111 ((c,b)[1000],a)[0110]
10001000 (c,b)[1000]
10001011 ((b,(a,c)[0110])[1000],a)[0110]
10001101 ((c,(a,b)[0110])[1000],a)[0110]
10001110 (((b,c)[0110],(a,c)[0110])[1000],a)[0110]
10010000 ((b,c)[0110],a)[1000]
10010011 ((c,a)[1000],b)[0110]
10010101 ((b,a)[1000],c)[0110]
10010110 ((a,b)[0110],c)[1001]
10011001 (b,c)[1001]
10011010 ((b,a)[1101],c)[0110]
10011100 ((c,a)[1101],b)[0110]
10011111 ((b,c)[0110],a)[1101]
10100000 (c,a)[1000]
10100011 (((b,c)[0110],a)[1000],b)[0110]
10100101 (a,c)[1001]
10100110 ((b,a)[1011],c)[0110]
10101001 ((b,a)[1110],c)[0110]
10101010 c[10]
10101100 (((b,c)[0110],a)[1101],b)[0110]
10101111 (c,a)[1101]
10110001 ((c,(a,b)[0110])[1000],b)[0110]
10110010 (((b,c)[0110],(a,c)[0110])[1000],b)[0110]
10110100 ((c,b)[1101],a)[0110]
10110111 (b,(a,c)[0110])[1011]
10111000 ((b,(a,c)[0110])[1011],a)[0110]
10111011 (c,b)[1101]
10111101 ((b,c)[0110],(a,c)[0110])[1110]
10111110 (c,(a,b)[0110])[1101]
11000000 (b,a)[1000]
11000011 (a,b)[1001]
11000101 (((b,c)[0110],a)[1000],c)[0110]
11000110 ((c,a)[1011],b)[0110]
11001001 ((c,a)[1110],b)[0110]
11001010 (((b,c)[0110],a)[1101],c)[0110]
11001100 b[10]
11001111 (b,a)[1101]
11010001 ((b,(a,c)[0110])[1000],c)[0110]
11010010 ((c,b)[1011],a)[0110]
11010100 (((b,c)[0110],(a,c)[0110])[1000],c)[0110]
11010111 (c,(a,b)[0110])[1011]
11011000 ((c,(a,b)[0110])[1011],a)[0110]
11011011 ((b,c)[0110],(a,c)[0110])[1101]
11011101 (c,b)[1011]
11011110 (b,(a,c)[0110])[1101]
11100001 ((c,b)[1110],a)[0110]
11100010 ((b,(a,c)[0110])[1011],c)[0110]
11100100 ((c,(a,b)[0110])[1011],b)[0110]
11100111 ((b,c)[0110],(a,c)[0110])[1011]
11101000 (((b,c)[0110],(a,c)[0110])[1101],b)[0110]
11101011 (c,(a,b)[0110])[1110]
11101101 (b,(a,c)[0110])[1110]
11101110 (c,b)[1110]
11110000 a[10]
11110011 (b,a)[1011]
11110101 (c,a)[1011]
11110110 ((b,c)[0110],a)[1011]
11111001 ((b,c)[0110],a)[1110]
11111010 (c,a)[1110]
11111100 (b,a)[1110]
11111111 1
```

B Garbled Table Reduction

We now turn to another way of reducing the size of the garbled tables in the case where the KDF is modeled as a PRF, which is similar to the GRR method used in the case of a KDF which is correlation robust. In order to reduce the size of the tables by more than 25% we must use the values of the input wires to define *both* garbled values of the output wire, rather than defining a single value, as in the optimisation in Section 4 above. Applying this method to "odd" gates with two inputs reduces the size of the tables by a factor of 1/3, while applying it to "even" gates reduces their size by 1/2.

B.1 Odd 2-to-1 Gates

We first restrict to 2-to-1 gates which are "odd". There are 8 such gates, including the OR, NOR, AND or NAND gates. Each gate of this type has three input combinations resulting in one output value, and a single input combination resulting in the other output value. Let us denote the former set of input combinations as the "set-of-three" (e.g., the set-of-three of an OR gate includes the input pairs (0,1), (1,0) and (1,1)). It is important not to disclose to the evaluator any information about the real values that pass through the gates. In particular, the evaluator must not learn whether the input values belong to the set-of-three.

Consider the output value which is output for each of the input combinations in the set-of-three. We divide the corresponding garbled value to three parts and define each of them to be a function of a different input combination from the set-of-three. As a result, a table entry corresponding to any of these input combinations should store only the encryptions of the other two thirds of the garbled value. In the evaluation phase, the evaluator will use the pair of garbled input values in order to directly compute one third of the garbled output value, and to decrypt the table entry which contains the other two thirds of this value. As for the remaining input combination, i.e., the one not in the set-of-three, we define an arbitrary third of the corresponding output value as a function of this combination, and store the remaining two thirds in the table. As a result, each entry of the table is 2/3 of its original length.

As before we let t denote the bit-length of the garbled values. Let us assume wlog that t is divisible by 3. We divide the garbled values to three equal parts. Denote the parts of the garbled values as $k_3^b = X_1^b \| X_2^b \| X_3^b$, for $b = 0, 1$.

We first map each input pair to an index of one third of the garbled value of the output wire. Namely, we choose for every gate a random function $\gamma(b_1, b_2) \to \{1, 2, 3\}$ subject to the constraint that γ does not map any two input combinations from the set-of-three to the same value. We should have actually referred to this function as γ_{Gid}, since a separate function is chosen for every gate, but we chose to keep the notation simpler.

Consider first the garbled value corresponding to the output value of the set-of-three. The j'th third of this value (for $j = 1, 2, 3$) is defined as $X_j^{G(b,b')} = \text{KDF}^{t/3}(k_1^b, k_2^{b'}, \pi_1(b) \| \pi_2(b') \| Gid \| 0)$, where $(b, b') = \gamma^{-1}(j)$. In other words, this third of the garbled output value is defined as a function of the garbled values of (b, b') in the set-of-three. Now, consider the input combination (b_1, b_2) which is not in the set-of-three. Let $j = \gamma(b_1, b_2)$. We define the value of the jth third of the corresponding garbled value to be

$$X_j^{G(b_1,b_2)} = \text{KDF}^{t/3}\left(k_1^{b_1}, k_2^{b_2}, \pi_1(b_1) \| \pi_2(b_2) \| Gid \| 0\right).$$

We choose random values for the remain two thirds of this garbled value.

The garbled table stores, for each input combination, the index of the third of the garbled output value which is directly defined by the corresponding input values, and an encryption of the other two thirds of the output value. We list below the contents of an entry (c_1, c_2) in the table. This entry corresponds to actual input values $b_1 = \pi_1^{-1}(c_1), b_2 = \pi_2^{-1}(c_2)$. Let us also define $j_1(b_1, b_2), j_2(b_1, b_2) \in \{1, 2, 3\}$ as the indexes of the two thirds of the output value whose encryptions are stored in the table entry (namely, these values satisfy that $\{j_1(b_1, b_2), j_2(b_1, b_2), \gamma(b_1, b_2)\} = \{1, 2, 3\}$, and $j_1(b_1, b_2) < j_2(b_1, b_2)$). The table stores the entry

$$c_1, c_2 : \left(\gamma(b_1, b_2) \| X_{j_1(b_1,b_2)}^{G(b_1,b_2)} \| X_{j_2(b_1,b_2)}^{G(b_1,b_2)} \| c_3\right) \oplus \text{KDF}^{2t/3+3}\left(k_1^{b_1}, k_2^{b_2}, c_1 \| c_2 \| \text{Gid} \| 1\right),$$

Thus instead of storing $t + 1$ bits, each table entry only stores $2t/3 + 3$ bits. Hence, we save around 1/3 of the required space for the tables.

On evaluating the garbled gate the evaluator uses his input to unmask the corresponding entry of the table. The entry reveals the value of the index j, and the values of the other two thirds of the garbled output value. The evaluator then computes the remaining third of the garbled value, X_j, by applying the KDF to his known values, i.e., computing $\text{KDF}(k_1^{b_1}, k_2^{b_2}, \pi_1(b_1) \| \pi_2(b_2) \| Gid \| 0)$. Note that the same operation is performed for every input combination, regardless of whether it is in the set-of-three or not.

B.2 Even 2-to-1 Gates

We first define $b_1^{(i)}$ and $b_2^{(i)}$ for $i = 1, 2, 3, 4$ via $G(b_1^{(1)}, b_2^{(1)}) = G(b_1^{(3)}, b_2^{(3)})$ and $G(b_1^{(2)}, b_2^{(2)}) = G(b_1^{(4)}, b_2^{(4)})$. We then define for $j = 1$ and $j = 2$.

$$k_3^{G(b_1^{(j)}, b_2^{(j)})} \| s_j = \mathrm{KDF}^{l+1}\left(k_1^{b_1^{(j)}}, k_2^{b_2^{(j)}}, \pi_1(b_1^{(j)}) \| \pi_2(b_2^{(j)}) \| \mathrm{Gid} \| 0\right).$$

The first l bits define the two garbled values of the output wire, and the final $l + 1$ bits define the value s_j. Now for the rows corresponding to the two external value pairs

$$(c_1, c_2) \in \left\{\left(\pi_1(b_1^{(1)}), \pi_2(b_2^{(1)})\right), \left(\pi_1(b_1^{(2)}), \pi_2(b_2^{(2)})\right)\right\}$$

we store in the garbled table the encryption of the external value of the wire, that is,

$$c_1, c_2 : \left(\pi_3(G(b_1^{(1)}, b_2^{(1)})) \oplus s_1, \pi_3(G(b_2^{(2)}, b_2^{(2)})) \oplus s_2\right).$$

The remaining rows are calculated in the standard way.

The evaluator can recover the garbled wire values. If the evaluator is given keys corresponding to one of the standard table rows, then the usual method can be used. Otherwise the evaluator can run the KDF to obtain the garbled wire value, and uses the $l + 1$ output bit of the KDF to decrypt the external wire value, stored in the garbled table.

C Multiple Outputs

In general the two parties have inputs x_1 and x_2. Player P_1 wishes to determine the output of $f_1(x_1, x_2)$, whilst player P_2 wishes to determine the output of $f_2(x_1, x_2)$. As we have just described we only have a method for player P_2, which is the player computing the circuit, to obtain the output of the function $f_2(x_1, x_2)$. This method must be extended to enable P_1 to obtain its own output $f_1(x_1, x_2)$. This can be achieved by requiring P_2 to compute $f_1(x_1, x_2)$ and send it to P_1. More accurately, in the semi-honest case P_2 computes and sends an encryption of $f_1(x_1, x_2)$ which only P_1 can decrypt, while in the malicious and covert cases P_2 computes an encrypted and authenticated copy of $f_1(x_1, x_2)$, which only P_1 can decrypt and verify. The encryption and authentication keys are used only a single time an therefore it is possible to use one-time schemes.

We describe here the method for the malicious case. This method has been described before, the novelty in our work is a modification that uses a MAC which has an efficient circuit based implementation. We assume the inputs of the two players are, respectively, n_1 and n_2 bits long, and that the output bit lengths of the two functions are m_1 and m_2, respectively. The keys used for encrypting and authenticating P_1's output are supplied by P_1, and therefore the input of P_1 is extended to be $n_1 + 3 \cdot m_1$ bits long, and shall be denoted by x_1, a, b and c where $|x_1| = n_1$ and $|a| = |b| = |c| = m_1$. The new circuit should produce the output of the function

$$G(x_1, a, b, c, x_2) = (e \| m) \| f_2(x_1, x_2),$$

where the value e acts as an encryption of $f_1(x_1, x_2)$ whilst m acts as an authentication tag. For this purpose, we use

$$e = f_1(x_1, x_2) \oplus c, \quad \text{and} \quad m = (a \otimes e) \oplus b.$$

By $\otimes$ we denote the convolution operator modulo 2, i.e., for bits strings a and e of length m_1 we have

$$(a \otimes e)_j = \bigoplus_{i=0}^{m_1 - 1} a_i \cdot e_{i+j}.$$

Consequently, m is an authentication tag based on a one-time MAC which is computed very efficiently by a binary circuit (using ${m_1}^2$ AND gates and ${m_1}^2$ XOR gates). The computation of this MAC is much

more efficient than, say, a binary circuit computing the one-time MAC function $a \cdot c + b$ over a finite field. The advantage of using the convolution, as opposed to multiplication in a finite field, such as $\mathbb{F}_{2^{m_1}}$, is that we do not need to worry about a different circuit for each value of m_1, the circuit construction is highly regular. The additional XOR gates needed in computing the convolution will produce a negligible cost, and will indeed produce no-extra cost given some of our later optimisations in the case of correlation robust KDFs. The circuit G has an output size of $2m_1 + m_2$ bits. This circuit is then evaluated, player P_2 obtains his output and then sends back the two values e and m to player P_1. Player P_1 checks the authentication tag m and obtains $f_1(x_1, x_2)$ by decrypting e.

Note in the case of semi-honest adversaries we only need to compute the value e (using m_1 XOR gates), and the authentication tag m (the computation of which requires $2{m_1}^2$ gates) can be ignored. Thus for semi-honest adversaries the circuit becomes considerably simpler.

D Malicious and Covert Adversaries

The protocol in Section 2 is only secure against so-called semi-honest adversaries, namely ones which are guaranteed to follow the protocol but who wish to break the secrecy. In [23] a protocol which is secure against malicious adversaries is given, whilst in [1] another protocol is given for covert adversaries. As mentioned earlier in Section 1, we only investigate these protocols in our work. In both these cases we try to protect against adversaries who may deviate from the protocol. In the case of security against malicious adversaries we require the honest party is guaranteed the other party has not deviated, except for a negligible probability. Whereas in the case of covert adversaries we require that the honest party detects the other party is deviating with a "reasonable" probability.

The methods of [1, 23] for converting the semi-honest protocol to higher levels of security follow two basic conversions. These are controlled by two security parameters s_1 and s_2, although the precise details of the protocols differ. First the original circuit is replaced by a new circuit which has an increased number of input wires for player P_2. The expansion in the number of input wires for player P_2 is controlled by the security parameter s_2. We examine it in more detail in Section D.1 and in our experimental results.

Player P_1 then generates s_1 garbled circuits which are passed to P_2, along with various commitments. Then by a process of cut-and-choose various circuits are selected for evaluation. The precise details of each protocol we leave to [23] and [1].

In the case of malicious adversaries it is conjectured in [25] that the probability of a malicious adversary being able to defeat the malicious protocol is bounded by $\max(2^{-s_1/4}, 2^{-s_2})$. Thus in the malicious case it makes sense to take $s_2 = s_1/4$. In [1] it is argued that the probability of a covert adversary being able to defeat the covert protocol without being detected is bounded by

$$1 - (1 - 1/s_1)(1 - 2^{-s_2+1}).$$

The probability $1/s_1$ is related to a covert P_1 being able to generate an invalid circuit without P_2 detecting, and the probability 2^{-s_2} is related to the covert P_2 being able to subvert the oblivious transfer without being detected. Hence, the probability of defeating the protocol is more accurately bounded by $\max(1/s_1, 2^{-s_2})$. Thus in the covert case it makes sense to take $s_2 \approx \log_2 s_1$. Typical values for s_1 and s_2 in the malicious case could be $s_1 = 160$ and $s_2 = 40$, ensuring that cheating probability is limited below 2^{-40}, whereas for the covert case we could take $s_1 = 16$ and $s_2 = 4$, limiting the cheating probability at 2^{-4}. Recall that s_1 is the number of copies of the circuit that must be communicated between the parties.

D.1 P_2's input circuit

As mentioned earlier, for the malicious and covert cases we need to replace the n_2 input wires of P_2 with a new set of n_2' input wires. We will let $x_2^{(i)}$ for $i = 1, \ldots, n_2$ denote the original input wires for P_2, and $x_2'^{(i)}$ for $i = 1, \ldots, n_2'$ denote the new input wires. The reasons for this are explained in [23]. The main idea is that each of the original input wires is defined to be an XOR of a subset of the new input wires. Therefore, any assignment to the first original input wire, can be realized using many assignments to the

new input wires. Therefore potential attacks of P_2 which might learn a value of a new input wire do not disclose information about any of the original input wires. There are two techniques proposed in [23] to perform this "expansion", which we shall now elaborate on. The first technique uses more input wires than the second one and as a result, requires the players to compute more OTs. However, the second technique is preferable in terms of the number of XOR gates that it uses. Both techniques depend on a statistical security parameter s_2, and, as proved in [23], reduce the cheating probability to roughly 2^{-s_2}.

Independent Inputs Technique: Here s_2 new input wires are assigned to each original input wire, which is defined to be the XOR of these new input wires. Namely, we set $n_2' = s_2 \cdot n_2$ and set

$$x_2^{(i)} = \bigoplus_{j=1}^{s_2} {x_2'}^{(s_2 \cdot (i-1)+j)}.$$

To sum up, the circuit has $s_2 \cdot n_2$ new input wires, and an additional $s_2 \cdot n_2$ XOR gates added to it. Each of the new input wires is used in computing only one of the original input wires.

Random Combinations Technique: Here new input wires are reused in the sense that each of the original input wires is defined to be the XOR of a random subset of the set of new wires. The number of new input wires is $n_2' = \max(4n_2, 8s_2)$, which is smaller than $s_2 \cdot n_2$, the number of new wires used by the previous technique. Each original input wire is defined to be a random linear combination of the new input wires. In other words, the system uses a random $n_2 \times n_2'$ binary matrix $A = (a_{i,j})$ that defines which linear combination of new input wires is assigned to each original input wire. The mapping between the original and the new input wires is given by

$$x_2(i) = \bigoplus_{j=1}^{n_2'} a_{i,j} \cdot {x_2'}^{(j)}.$$

Thus the circuit has roughly an extra $n_2 n_2'/2 = \max(2n_2^2, 4n_2 s_2)$ XOR gates added to it. This number is greater than the $s_2 \cdot n_2$ XOR gates used by the previous technique. Note that this number of gates might be larger than the entire original circuit. This happens in particular, if the size of the original circuit is linear in the number of its inputs n_2, as in the comparison circuit for the classical millionaires' problem.

In [25] a method is presented which reduces the number of additional XOR gates needed for the Random Combinations Technique by about 60%, by identifying common sub-expressions of multiple XOR expressions and reusing them. Still, their number will be larger than in the Independent Inputs Technique. Applying that technique, the total number of new inputs is $n_2' = \max(4n_2, 8s_2)$, and the number of new XOR gates is $0.3 n_2 n_2'$.

It is unclear in any given implementation exactly which technique is more efficient, since the run-times depend on the precise values of n_2 and s_2, and the relative costs of trading more OT's for less XOR gates. The value of s_2 will not only be affected by the desired security level, but also by whether we are protecting against malicious or covert adversaries, as detailed in our experiments, in the malicious case we set s_2 to be be equal to 40, and in the covert case we use $s_2 = 4$. Our experiments detail the performance of both techniques in different settings.