Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima

Vladimir Kolesnikov[1], Ahmad-Reza Sadeghi[2], and Thomas Schneider[2]

[1] Bell Laboratories, 600 Mountain Ave. Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[2] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{ahmad.sadeghi,thomas.schneider}@trust.rub.de*

Abstract. We consider generic Garbled Circuit (GC)-based techniques for Secure Function Evaluation (SFE) in the semi-honest model.
We describe efficient GC constructions for addition, subtraction, multiplication, and comparison functions. Our circuits for subtraction and comparison are approximately two times smaller (in terms of garbled tables) than previous constructions. This implies corresponding computation and communication improvements in SFE of functions using our efficient building blocks. The techniques rely on recently proposed "free XOR" GC technique.
Further, we present concrete and detailed improved GC protocols for the problem of secure integer comparison, and related problems of auctions, minimum selection, and minimal distance. Performance improvement comes both from building on our efficient basic blocks and several problem-specific GC optimizations. We provide precise cost evaluation of our constructions, which serves as a baseline for future protocols.

Keywords: Secure Computation, Garbled Circuit, Millionaires Problem, Auctions, Minimum Distance

1 Introduction

We are motivated by secure function evaluation (SFE) of integer comparison, and related problems such as auctions and biometric authentication. For this, we propose new, more efficient SFE protocols for these functions. More specifically, we propose improved constructions for subtraction, and comparison functions, and demonstrate their advantages on the example of our motivating applications.

Comparison is a widely used basic primitive. In particular, it plays an especially important role in financial transactions, biometric authentication, database mining applications, etc.

* Supported by EU FP6 project SPEED, EU FP7 project CACE and ECRYPT II.

Auctions. With the growth of the Internet and its widespread acceptance as the medium for electronic commerce, online auctions continue to grow in popularity. Additionally, many sellers consider the "name your price" model. For example, sites such as priceline.com ask a buyer for a price he is willing to pay for a product, and the deal is committed to if that price is greater than a certain (secret) threshold. In many such situations, it is vital to maintain the privacy of bids of the players. Indeed, revealing an item's worth can result in artificially high prices or low bids, specifically targeted for a particular buyer or seller. While a winning bid or a committed deal may necessarily reveal the cost of the transaction, it is highly desirable to keep all other information (e.g., unsuccessful bids) secret. There has been a large stream of work dedicated to ensuring privacy and security of online auctions and haggling (e.g., [13,7,39]). Our work complements, extends, and builds on it.

Biometric authentication. Widespread adoption of biometric authentication (e.g., fingerprint or face recognition) is causing strong concerns of privacy violations. Indeed, improper use of biometric information has far more implications than "simple" collection of personal information. Adoption of privacy-preserving biometric authentication is highly desired and will benefit the users and the administrators of the systems alike. Because biometric images are never scanned perfectly, the identity of the user is determined by proximity of the scanned and stored biometrics. It is natural, therefore, that threshold comparisons are frequently employed in such identification systems. Further, in some multi-user systems, it may be desired to simply find the closest match in the database. In such systems, secure comparison would be also extensively used.

State of the art for secure comparison and related algorithms. Starting with the original paper of Yao [45], secure comparison, also referred to as the "two Millionaires problem", has attracted much attention [46,20,37,31]. A variety of techniques are employed in these solutions – homomorphic encryption, evaluation of branching programs, Garbled Circuit (GC).

Today, in the standard computational setting, the most efficient protocol is the simple evaluation of the generic GC. Indeed, the size of the comparison circuit is quite small (linear in the size of the inputs), and its secure evaluation is rather efficient (linear number of Oblivious Transfers (OT) and evaluations of a cryptographic hash function, such as SHA-256).

Most popular alternative solutions are based on homomorphic encryptions. For comparison, they offer a similar complexity compared to GC, as they still must perform a linear (in the input) number of public key operations by both players. However, GC offers more flexible and cheap programming possibilities, due to its low cost of manipulation of boolean values. In contrast, homomorphic encryptions are not suitable, e.g., for branching based on the encrypted value which can be achieved only with much more expensive techniques than GC).

In sum, GC approach is a clear choice for integer comparison, its extensions, such as auctions, simple integer manipulations (addition and even multiplica-

tions) and a variety of other problems that have small circuit representation. We build our solutions in this framework.

Our contributions. As justified above, our work is based on GC. We advance the state of the art of SFE for subtraction and comparison functions, by constructing their more efficient GC representations. We work in the semi-honest model which is appropriate for many application scenarios.

More specifically, our optimizations take advantage of the recently proposed method of GC construction [33], where XOR gates are evaluated essentially for free (one XOR operation on keys, and no garbled table entries to generate or transfer). We show how to compute comparison and other basic functions with circuits consisting mostly of XOR gates. This results in reduction of the size of GC (i.e., the size of garbled tables) by approximately half (see Table 2 for detailed comparison). We note that the method of [33] (and thus our work) requires the use of a weak form of Random Oracle, namely of correlation-robust functions [26].

As further contribution, we then follow through, and discuss in detail GC-based constructions for the Millionaires problem, computing first-price auctions and minimum Hamming- or Euclidian distance. In addition to improvements due to our new building blocks, our protocols benefit from a number of GC-based optimizations. In addition to establishing a new performance baseline for these problems, we aim to promote GC as a very efficient solution, and prevent its frequent unfair dismissal as an "impractical generic approach".

Related work. SFE (and in particular GC), and secure comparison has received much attention in the literature, all of which we cannot possibly include here. In this section we summarize relevant work to give the reader a perspective on our results. We discuss additional related work (on which we improve) in the individual sections of the paper.

Circuit-Based Secure Function Evaluation. GC technique of SFE was introduced by Yao [46], with a formal proof of security (in the semi-honest model) given in [34]. Extensions of Yao's garbled circuit protocol to security against covert players were given in [1,25], and against malicious players in [27,35,40]. Our constructions rely on the recent work of [33], where a GC technique is proposed that allows evaluation of XOR gates "for free", i.e., with no communication and negligible computation costs. In [33] improved circuit constructions for multiplexer, addition and (in-)equality testing are presented. Our main contribution – the building block constructions – further improve their proposals (e.g., subtraction and comparison are improved by a factor of two.

Secure Two-Party Comparison. The first secure two-party comparison protocol was proposed in [45], and today GC [46] is the most efficient solution to this problem as shown in this paper: our solution for comparing two ℓ-bit numbers

requires $16\ell t$ bit offline communication and $3\ell t$ bit online communication, where t is a symmetric security parameter (i.e., length of a symmetric key).

Homomorphic encryption is another popular tool for comparison. The protocol of Fischlin [20] uses the Goldwasser-Micali XOR-homomorphic encryption scheme [24] and has communication complexity $\ell T(\kappa + 1)$, where κ is a statistical correctness parameter (e.g., $\kappa = 40$) and T is an asymmetric security parameter (i.e., size of an RSA modulus). The comparison protocol of [6] uses bitwise Paillier encryption and has communication complexity $4\ell T$. This protocol was improved in [14,16,15] to communication complexity $2\ell T$ by using a new homomorphic encryption scheme with smaller ciphertext size. These two-party protocols were extended to comparisons in the multi-party setting with logarithmic and linear round complexity in [21].

Minimum Selection. A two-party protocol for finding k-Nearest Neighbors was given in [44], and improved from quadratic to linear communication complexity in [43]. Our protocol for finding the nearest neighbor is a more efficient protocol for the special case $k = 1$. A simple protocol to select the minimum of homomorphically encrypted values based on the multiplicative hiding assumption was given in [30] in the context of privacy-preserving benchmarking. However, multiplicative blinding reveals some information about the magnitude of the blinded value. Our minimum selection protocol can be used as a provably secure replacement of this protocol. Finally, we note that in our minimum Hamming distance protocol we use several steps of the Hamming distance protocol of [28].

Efficient circuits for addition and multiplication. Boyar et al. [10,11,9] considered multiplicative complexity[3] of symmetric functions (i.e., functions only dependent on the hamming weight of their inputs). As a corollary, Boyar et al. describe efficient circuits for addition (and thus multiplication, via Karatsuba-Ofman method [29]). Our subtraction and comparison building blocks are extensions of their construction.

2 Preliminaries

In this section, we summarize our conventions and setting in §2.1 and cryptographic tools used in our constructions: oblivious transfer (OT) in §2.3, garbled circuits (GC) with free XOR in §2.4, and additively homomorphic encryption (HE) in §2.2. Reader familiar with the prerequisites may safely skip to §3.

2.1 Parameters, Notation and Model

We denote symmetric security parameter by t and the asymmetric security parameter, i.e., bitlength of RSA moduli, by T. Recommended parameters for short-term security (until 2010) are for example $t = 80$ and $T = 1024$ [23]. The

[3] Multiplicative complexity of a function measures the number of AND gates in its circuit (and gives NOT and XOR gates for free).

bitlength of a garbled value is $t' := t + 1$ (cf. §2.4 for details). The statistical correctness parameter is denoted with κ (the probability of a protocol failure is bounded by $2^{-\kappa}$) and the statistical security parameter with σ. In practice, one can choose $\kappa = \sigma = 80$. The bitlength of protocol inputs is denoted with ℓ and the number of inputs with n. We write $\mathbf{x}^\ell$ to denote ℓ-bit value x.

We work in the semi-honest model. We note that the method of [33] (and thus our work) requires the use of a weak form of Random Oracle, namely of correlation-robust functions [26].

2.2 Homomorphic Encryption (HE)

Some of our constructions make black-box usage of a semantically secure homomorphic encryption scheme with plaintext space $(P, +, 0)$, ciphertext space $(C, *, 1)$, and probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

An *additively homomorphic encryption* scheme allows addition under encryption as it satisfies $\forall x, y \in P : \mathsf{Dec}(\mathsf{Enc}(x) * \mathsf{Enc}(y)) = x + y$. It can be instantiated with a variety of cryptosystems including [41,17], or the cryptosystem of [14,16,15] which is restricted to small plaintext space P – just to name a few.

For the sake of completeness we mention, that the cryptosystem of [8] allows for an arbitrary number of additions and one multiplication and *fully homomorphic encryption* schemes allow to evaluate an arbitrary number of additions and multiplications on ciphertexts. Possible candidates are the cryptosystem of [3] (size of ciphertexts grows exponentially in the number of multiplications) or the recently proposed scheme without such a restriction [22]. However, the size of ciphertexts in these schemes is substantially larger than that of the purely additively homomorphic schemes.

2.3 Oblivious Transfer (OT)

Parallel 1-out-of-2 Oblivious Transfer of m ℓ-bit strings, denoted as OT_ℓ^m, is a two-party protocol run between a chooser $\mathcal{C}$ and a sender $\mathcal{S}$. For $i = 1, \ldots, m$, $\mathcal{S}$ inputs a pair of ℓ-bit strings $s_i^0, s_i^1 \in \{0,1\}^\ell$ and $\mathcal{C}$ inputs m choice bits $b_i \in \{0,1\}$. At the end of the protocol, $\mathcal{C}$ learns the chosen strings $s_i^{b_i}$, but nothing about the unchosen strings $s_i^{1-b_i}$ whereas $\mathcal{S}$ learns nothing about the choices b_i.

Efficient OT protocols. We use OT_ℓ^m as a black-box primitive which can be instantiated efficiently with different protocols [38,2,36,26]. For example the protocol of [2] implemented over a suitably chosen elliptic curve has communication complexity $m(6(2t+1)) + (2t+1) \sim 12mt$ bits and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the standard model as described in the full version of this paper [32]. Similarly, the protocol of [38] implemented over a suitably chosen elliptic curve has communication complexity $m(2(2t+1) + 2\ell)$ bits and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the random oracle model. Both protocols require $\mathcal{O}(m)$ scalar point multiplications.

Extending OT efficiently. The extensions of [26] can be used to efficiently reduce the number of computationally expensive public-key operations of OT_ℓ^m to

be independent of m. Their transformation for semi-honest receiver reduces OT_ℓ^m to OT_t^t and a small additional overhead: one additional message, $2m(\ell+t)$ bits additional communication, and $\mathcal{O}(m)$ invocations of a correlation robust hash function ($2m$ for $\mathcal{S}$ and m for $\mathcal{C}$) which is substantially cheaper than $\mathcal{O}(m)$ asymmetric operations. Also a slightly less efficient extension for malicious receiver is given in their paper.

2.4 Garbled Circuits (GC)

The most efficient method for secure evaluation of a boolean circuit C for computationally bounded players is Yao's garbled circuit approach [46,34]. We briefly summarize the main ideas of this protocol in the following. The circuit *constructor* (server $\mathcal{S}$) creates a *garbled circuit* $\widetilde{C}$ with algorithm $\mathsf{CreateGC}$: for each wire W_i of the circuit, he randomly chooses two garbled values $\widetilde{w}_i^0, \widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the *garbled value* of W_i's value j. (Note: $\widetilde{w}_i^j$ does not reveal j.) Further, for each gate G_i, $\mathcal{S}$ creates a *garbled table* $\widetilde{T}_i$ with the following property: given a set of garbled values of G_i's inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding G_i's output, but nothing else. $\mathcal{S}$ sends these garbled tables, called *garbled circuit* $\widetilde{C}$ to the *evaluator* (client $\mathcal{C}$). Additionally, $\mathcal{C}$ obliviously obtains the *garbled inputs* $\widetilde{w}_i$ corresponding to inputs of both parties (details on how this can be done later in §2.4). Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ on the garbled inputs with algorithm EvalGC to obtain the *garbled outputs* simply by evaluating the garbled circuit gate by gate, using the garbled tables $\widetilde{T}_i$. Finally, $\mathcal{C}$ translates the garbled outputs into output values given for the respective players (details below in §2.4). Correctness of GC follows from the method of how garbled tables $\widetilde{T}_i$ are constructed.

Improved Garbled Circuit with free XOR [33]. An efficient method for creating garbled circuits which allows "free" evaluation of XOR gates was presented in [33]. More specifically, a garbled XOR gate has no garbled table (*no communication*) and its evaluation consists of XOR-ing its garbled input values (*negligible computation*). The other gates, called *non-XOR gates*, are evaluated as in Yao's GC construction [46] with a *point-and-permute technique* (as used in [37]) to speed up the implementation of the GC protocol: the garbled values $\widetilde{w}_i = \langle k_i, \pi_i \rangle \in \{0,1\}^{t'}$ consist of a symmetric key $k_i \in \{0,1\}^t$ and a random permutation bit $\pi_i \in \{0,1\}$ and hence have length $t' = t+1$ bits. The permutation bit π_i is used to select the right table entry for decryption with the t-bit key k_i (recall, t is the symmetric security parameter). The encryption of the garbled table entries is done with the symmetric encryption function $\mathsf{Enc}^s_{k_1,\ldots,k_d}(m) = m \oplus H(k_1||\ldots||k_d||s)$, where d is the number of inputs of the gate, s is a unique identifier for the specific row in the gate's garbled table used once, and H is a suitably chosen cryptographic hash function. Hence, creation of the garbled table of a non-XOR d-input gate requires 2^d invocations of H and its evaluation needs one invocation, while XOR gates are "for free".

The main observation of [33] is, that the constructor $\mathcal{S}$ chooses a global key difference $\Delta \in_R \{0,1\}^t$ which remains unknown to evaluator $\mathcal{C}$ and relates the garbled values as $k_i^0 = k_i^1 \oplus \Delta$. (This technique was subsequently extended in the LEGO paper [40] which allows to compose garbled circuits dynamically with security against malicious circuit constructor). Clearly, the usage of such garbled values allows for *free evaluation of XOR gates* with input wires W_1, W_2 and output wire W_3 by computing $\widetilde{w}_3 = \widetilde{w}_1 \oplus \widetilde{w}_2$ (no communication and negligible computation). However, using related garbled values requires that the hash function H used to create the garbled tables of non-XOR gates has to be modeled to be correlation robust (as defined in [26]) which is stronger than modeling H as a key-derivation function (standard model) but weaker than modeling H as a random-oracle (ROM). In practice, H can be chosen from the SHA-2 family.

Input/Output Conversion. In secure two-party computation protocols executed between circuit constructor $\mathcal{S}$ and circuit evaluator $\mathcal{C}$, each of the inputs and outputs of the securely computed functionality can be given in different forms depending on the application scenario: privately known to one party (§A.1), secret-shared between both parties (§A.2), or homomorphically encrypted under the public key of the other party (§A.3). These inputs can be converted from different forms to garbled inputs given to $\mathcal{C}$. Afterwards, $\mathcal{C}$ evaluates the garbled circuit, obtains the garbled outputs, and converts them into outputs in the needed form.

The resulting communication complexities of these input and output conversion protocols for semi-honest parties are summarized in Table 1 and a detailed description of these known techniques is given in §A.

Table 1. Communication complexity for converting ℓ-bit inputs/outputs in different forms to inputs/outputs of a garbled circuit (parameters defined in §2.1). SS: Secret-Shared, HE: Homomorphically Encrypted.

	Input	Output
Private $\mathcal{S}$ (§A.1)	$\ell t'$ bits	ℓ bits
Private $\mathcal{C}$ (§A.1)	$\mathrm{OT}_{t'}^{\ell}$	ℓ bits
SS (§A.2)	$\mathrm{OT}_{t'}^{\ell}$	ℓ bits
HE (§A.3)	1 ciphertext + $5\ell t'$ bits + $\mathrm{OT}_{t'}^{\ell}$	1 ciphertext + $(\ell+\sigma)(5t'+1)$ bits

3 Building Blocks for GC

In this section we present our basic contribution – improved circuit constructions for several frequently used primitives, such as integer subtraction (§3.1), comparison (§3.2), and selection of the minimum value and index (§3.3)[4]. As

[4] As noted in §1, Boyar et al. [11,9] had previously proposed improved circuits for addition and multiplication. Further, the circuits for subtraction and comparison

summarized in Table 2, our improved circuit constructions are smaller than previous solutions by 33% to 50% when used with the GC of [33]. This reduction in size immediately translates into a corresponding improvement in communication and computation complexity of any GC protocol built from these blocks. The efficiency improvements are achieved by modifying the underlying circuits, i.e., by carefully replacing larger (more costly) non-XOR gates (e.g., a 3-input gate) with smaller non-XOR gates (e.g., a 2-input gate) and (free) XOR gates.

Table 2. Size of efficient circuit constructions for ℓ-bit values and computing the minimum value and index of n ℓ-bit values (in table entries).

Circuit	Standard GC	[33]	This Work (Improvement)
Multiplexer	8ℓ	4ℓ	
Addition/Subtraction (§3.1)	16ℓ	8ℓ	4ℓ (50%)
Multiplication (§3.1)	$20\ell^2 - 16\ell$	$12\ell^2 - 8\ell$	$8\ell^2 - 4\ell$ (33%)
Equality Test (§3.2)	8ℓ	4ℓ	
Comparison (§3.2)	8ℓ		4ℓ (50%)
Minimum Value + Index (§3.3)	$\approx 15\ell n$ [39]		$8\ell(n-1) + 4(n+1)$ (47%)

Multiplexer Circuit (MUX). Our constructions use ℓ-bit multiplexer circuits MUX to select one of the ℓ-bit inputs $\mathbf{x}^\ell$ or $\mathbf{y}^\ell$ as output $\mathbf{z}^\ell$, depending on the selection bit c. We use the construction of [33] with ℓ non-XOR gates.

3.1 Integer Addition, Subtraction and Multiplication

Addition circuits (ADD) to add two unsigned integer values $\mathbf{x}^\ell, \mathbf{y}^\ell$ can be efficiently composed from a chain of 1-bit adders ($+$), often called full-adders, as shown in Fig. 1. (The first 1-bit adder has constant input $c_1 = 0$ and can be replaced by a smaller half-adder). Each 1-bit adder has as inputs the carry-in bit c_i from the previous 1-bit adder and the two input bits x_i, y_i. The outputs are the carry-out bit $c_{i+1} = (x_i \wedge y_i) \vee (x_i \wedge c_i) \vee (y_i \wedge c_i)$ and the sum bit $s_i = x_i \oplus y_i \oplus c_i$ (the latter can be computed "for free" using "free XOR" [33]). The efficient construction of a 1-bit adder shown in Fig. 2 computes the carry-out bit as $c_{i+1} = c_i \oplus ((x_i \oplus c_i) \wedge (y_i \oplus c_i))$. Overall, the efficient construction for a 1-bit adder consists of four free XOR gates and a single 2-input AND gate which has size $2^2 = 4$ table entries. The overall size of the efficient addition circuit is $\left|\mathsf{ADD}^\ell\right| = \ell \cdot |+| = 4\ell$ table entries.

Subtraction in two's complement representation is defined as $\mathbf{x}^\ell - \mathbf{y}^\ell = \mathbf{x}^\ell + \bar{\mathbf{y}}^\ell + 1$. Hence, a subtraction circuit (SUB) can be constructed analogously to the addition circuit from 1-bit subtractors ($-$) as shown in Fig. 3. Each 1-bit

can be relatively naturally derived from the same ideas. We leave these building blocks in our presentation for completeness and readability.

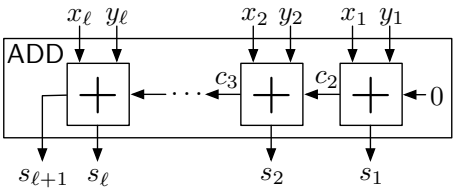

Fig. 1. Addition Circuit (ADD)

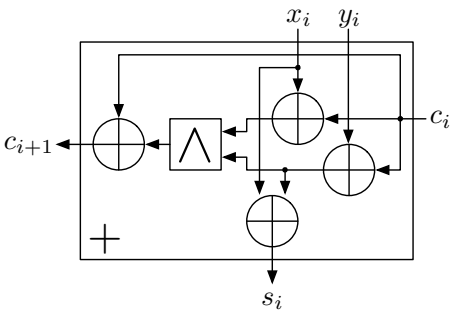

Fig. 2. Improved 1-bit Adder (+)

subtractor computes the carry-out bit $c_{i+1} = (x_i \wedge \bar{y}_i) \vee (x_i \wedge c_i) \vee (\bar{y}_i \wedge c_i)$ and the difference bit $d_i = x_i \oplus \bar{y}_i \oplus c_i$. We instantiate the 1-bit subtractor efficiently as shown in Fig. 4 to compute $c_{i+1} = x_i \oplus ((x_i \oplus c_i) \wedge (y_i \oplus c_i))$ with the same size as the 1-bit adder.

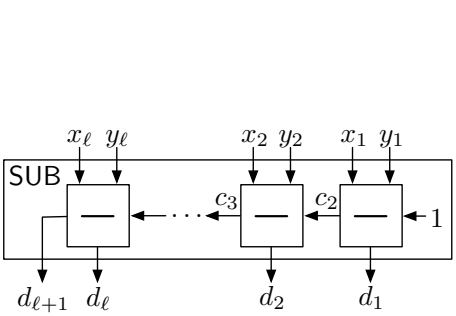

Fig. 3. Subtraction Circuit (SUB)

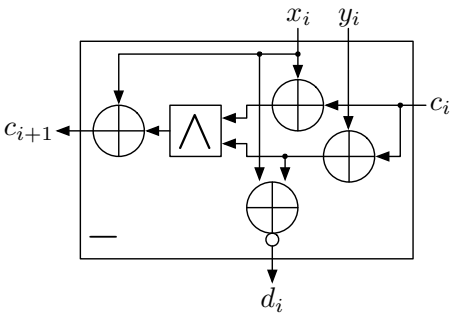

Fig. 4. Improved 1-bit Subtractor (−)

Multiplication circuits (MUL) to multiply two ℓ-bit integers $\mathbf{x}^\ell, \mathbf{y}^\ell$ can be constructed according to the "school method" for multiplication, i.e., adding up the bitwise multiplications of y_i and $\mathbf{x}^\ell$ shifted corresponding to the position: $\boldsymbol{x}^\ell \cdot \boldsymbol{y}^\ell = \sum_{i=1}^{\ell} 2^{i-1}(y_i \cdot \boldsymbol{x}^\ell)$. This circuit is composed from ℓ^2 of 1-bit multipliers (2-input AND gates) and $(\ell - 1)$ of ℓ-bit adders. Using the efficient implementation for adders, the size of the multiplication circuit is improved to $4\ell^2 + 4\ell(\ell - 1) = 8\ell^2 - 4\ell$ table entries. Alternatively, for multiplication of large ℓ-bit numbers, a circuit based on Karatsuba-Ofman multiplication [29] of size approximately $\mathcal{O}(\ell^{1.6})$ is more efficient.

3.2 Integer Comparison

We present improved circuit constructions for comparison of two ℓ-bit integers $\mathbf{x}^\ell$ and $\mathbf{y}^\ell$, i.e.,

$$z = \left[\mathbf{x}^\ell > \mathbf{y}^\ell\right] := \begin{cases} 1 & \text{if } \mathbf{x}^\ell > \mathbf{y}^\ell, \\ 0 & \text{else.} \end{cases}$$

Note that this functionality is more general than checking equality of ℓ-bit integers $\mathbf{x}^\ell$ and $\mathbf{y}^\ell$, i.e., $z = \left[\mathbf{x}^\ell = \mathbf{y}^\ell\right]$, for which an improved construction was given in [33].

As shown in Fig. 5, a comparison circuit (CMP) can be composed from ℓ sequential 1-bit comparators ($>$). (The first 1-bit comparator has constant input $c_1 = 0$ and can be replaced by a smaller gate). Our improved instantiation for a 1-bit comparator shown in Fig. 6 uses one 2-input AND gate with 4 table entries and three free XOR gates. Note, this improved bit comparator is exactly the improved bit subtractor shown in Fig. 4 restricted to the carry output: $\left[\mathbf{x}^\ell > \mathbf{y}^\ell\right] \Leftrightarrow \left[\mathbf{x}^\ell - \mathbf{y}^\ell - 1 \geq 0\right]$ which coincides with an underflow in the corresponding subtraction denoted by subtractor's most significant output bit $d_{\ell+1}$. The size of this comparison circuit is $\left|\mathsf{CMP}^\ell\right| = \ell \cdot |>| = 4\ell$ table entries.

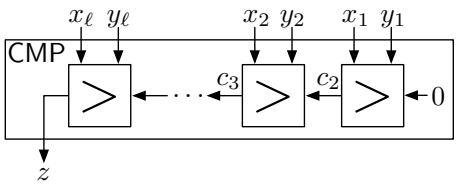

Fig. 5. Comparison Circuit (CMP)

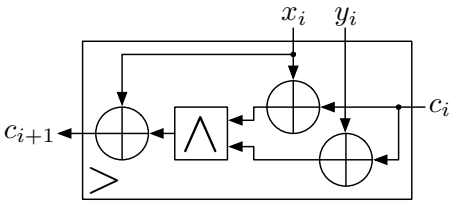

Fig. 6. Improved 1-bit Comparator ($>$)

Improved comparison circuits for $\left[\mathbf{x}^\ell < \mathbf{y}^\ell\right]$, $\left[\mathbf{x}^\ell \geq \mathbf{y}^\ell\right]$, or $\left[\mathbf{x}^\ell \leq \mathbf{y}^\ell\right]$ can be obtained from the improved circuit for $\left[\mathbf{x}^\ell > \mathbf{y}^\ell\right]$ by interchanging $\mathbf{x}^\ell$ with $\mathbf{y}^\ell$ and/or setting the initial carry to $c_1 = 1$.

3.3 Minimum Value and Minimum Index

Finally, we show how the improved blocks presented above can be combined to obtain an improved minimum circuit (MIN) which selects the minimum value $\mathbf{m}^\ell$ and minimum index i of a list of n ℓ-bit values $\mathbf{x}_0^\ell, \ldots, \mathbf{x}_{n-1}^\ell$, i.e., $\forall j \in \{0, \ldots, n-1\} : (\mathbf{m}^\ell < \mathbf{x}_\mathbf{j}^\ell) \vee (\mathbf{m}^\ell = \mathbf{x}_\mathbf{j}^\ell \wedge i \leq j)$. E.g., for the list $3, 2, 5, 2$ the outputs would be $\mathbf{m}^\ell = 2$ and $i = 1$ as the leftmost minimum value of 2 is at position 1. W.l.o.g. we assume that n is a power of two, so the minimum index can be represented with $\log n$ bits.

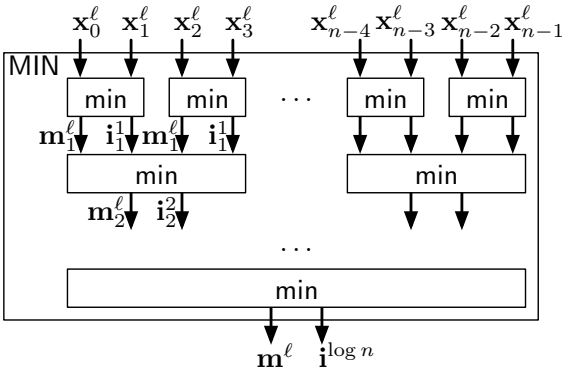

Fig. 7. Minimum Circuit (MIN)

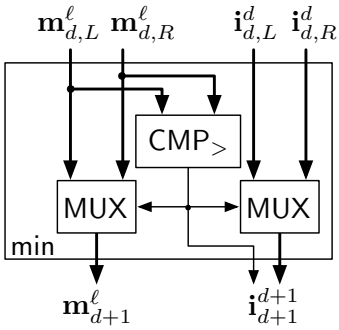

Fig. 8. Minimum Block (min)

Performance improvement of MIN mainly comes from the improved building blocks for integer comparison. We shave off an additive factor by carefully arranging tournament-style circuit so that some of the index wires can be reused and eliminated. That is, at depth d of the resulting tree we keep track of the ℓ-bit minimum value $\mathbf{m}^\ell$ of the sub-tree containing 2^d values but store and propagate only the d least significant bits $\mathbf{i}_d^d$ of the minimum index.

More specifically, the minimum value and minimum index are selected pairwise in a tournament-like way using a tree of minimum blocks (min) as shown in Fig. 7. As shown in Fig. 8, each minimum block at depth d gets as inputs the minimum ℓ-bit values $\mathbf{m}_{d,L}^\ell$ and $\mathbf{m}_{d,R}^\ell$ of its left and right subtrees T_L, T_R and the d least significant bits of their minimum indices $\mathbf{i}_{d,L}^d$ and $\mathbf{i}_{d,R}^d$, and outputs the minimum ℓ-bit value $\mathbf{m}_{d+1}^\ell$ and $(d+1)$-bit minimum index $\mathbf{i}_{d+1}^{d+1}$ of the tree. First, the two minimum values are compared with a comparison circuit (cf. §3.2). If the minimum value of T_L is bigger than that of T_R (in this case, the comparison circuit outputs value 1), $\mathbf{m}_{d+1}^\ell$ is chosen to be the value of T_R with an ℓ-bit multiplexer block [33]. In this case, the minimum index $\mathbf{i}_{d+1}^{d+1}$ is set to 1 concatenated with the minimum index of T_R using another d-bit multiplexer. Alternatively, if the comparison yields 0, the minimum value of T_L and the value 0 concatenated with the minimum index of T_L are output. Overall, the size of the efficient minimum circuit is $\left|\mathsf{MIN}^{\ell,n}\right| = (n-1)\cdot(\left|\mathsf{CMP}^\ell\right| + \left|\mathsf{MUX}^\ell\right|) + \sum_{j=1}^{\log n} \frac{n}{2^j}\left|\mathsf{MUX}^{j-1}\right| = 8\ell(n-1) + 4n\sum_{j=1}^{\log n}\frac{j-1}{2^j} < 8\ell(n-1) + 4n(1+\frac{1}{n}) = 8\ell(n-1) + 4(n+1)$.

Our method of putting the minimum blocks together in a tree (cf. Fig. 7) is non-trivial: If the minimum blocks would have been arranged sequentially (according to the standard selection algorithm to find the minimum), the size of the circuit would have been $(n-1)\cdot(\left|\mathsf{CMP}^\ell\right| + \left|\mathsf{MUX}^\ell\right| + \left|\mathsf{MUX}^{\log n}\right|) = 8\ell(n-1) + 4(n-1)\log n$ table entries which is less efficient than the tree.

In previous work [39], a circuit for computing first-price auctions (which is functionally equivalent to computing the maximum value and index) with a

size of approximately $15\ell n$ table entries is mentioned over which our explicit construction improves by a factor of approximately $\frac{15}{8}$.

4 Applications

We now describe how our efficient circuit constructions (§3) can be applied to improve previous solutions for several applications. We note that constructions of this section are not novel and may be folklore knowledge. We explicate them for concreteness, and use them to demonstrate the usefulness of our building blocks and to arrive at performance estimates to form a baseline for future protocols.

4.1 Integer Comparison (Millionaires Problem)

The "Millionaires problem" was introduced by Yao in [45] as motivation for secure compuation: two millionaires want to securely compare their respective private input values (e.g., their amount of money) without revealing more information than the outcome of the comparison to the other party. More concretely, client $\mathcal{C}$ holds a private ℓ-bit value x^ℓ and server $\mathcal{S}$ holds a private ℓ-bit value y^ℓ. The output bit $z = [x^\ell > y^\ell]$ should be revealed to both.

We obtain an efficient comparison protocol by evaluating the comparison circuit of §3.2 with the GC protocol of [33] and an efficient OT protocol. Our protocol, when executed without precomputation has asymptotic communication complexity $5\ell t + \mathrm{OT}_t^\ell$ bit with symmetric security parameter t (cf. §2.1).

In many practical application scenarios it is beneficial to shift as much of the computation and communication cost of a protocol into a setup (precomputation) phase, which is executed before the parties' inputs are known, while the parties' workload is low. In the following we apply a folklore technique, which demonstrates that GC protocols are ideally suited for precomputation as (in contrast to many protocols based on homomorphic encryption) almost their entire cost can be shifted into the setup phase.

Millionaires with setup. GC protocols allow to move all expensive operations (i.e., computationally expensive OT and creation of GC, as well as the transfer of GC which dominates the communication complexity) into the setup phase. The idea is to create and transfer the garbled circuit in the setup phase, and pre-compute the OTs [5]: for this, the parallel OT protocol is run on randomly chosen (by $\mathcal{C}$ and $\mathcal{S}$) values of corresponding sizes (instead of private inputs of $\mathcal{C}$ and pairs of garbled input values of $\mathcal{S}$). Then, in the online phase, $\mathcal{C}$ uses its randomly chosen value to mask his private inputs, and sends them to $\mathcal{S}$. $\mathcal{S}$ replies with encryptions of wire's garbled inputs using his random values from the setup phase. Which garbled input is masked with which random value is determined by $\mathcal{C}$'s message. Finally, $\mathcal{C}$ can use the masks he received from the OT protocol in the setup phase to exactly decrypt the correct garbled input value.

More precisely, the **setup phase** works as follows: for $i = 1, \ldots, \ell$, $\mathcal{C}$ chooses random bits $r_i \in_R \{0,1\}$ and $\mathcal{S}$ chooses random masks $m_i^0, m_i^1 \in_R \{0,1\}^{t'}$ (recall,

$t' = t+1$ is the bitlength of garbled values). Both parties run a $\mathrm{OT}^{\ell}_{t'}$ protocol on these randomly chosen values, where $\mathcal{S}$ inputs the pairs m_i^0, m_i^1 and $\mathcal{C}$ inputs r_i and $\mathcal{C}$ obliviously obtains the mask $m_i = m_i^{r_i}$. Additionally, $\mathcal{S}$ creates a garbled circuit $\widetilde{C}$ with garbled inputs $\widetilde{x}_i^0, \widetilde{x}_i^1$ and $\widetilde{y}_i^0, \widetilde{y}_i^1$ and sends $\widetilde{C}$ together with the output decryption table to $\mathcal{C}$. This message has the size $4\ell t' + 1 \sim 4\ell t$ bits. Overall, the setup phase has a slightly smaller communication complexity than the Millionaires protocol without setup described above.

In the **online phase**, $\mathcal{S}$ sends the garbled values $\widetilde{\mathbf{y}}^{\ell}$ corresponding to his input $\mathbf{y}^{\ell}$ to $\mathcal{C}$ and the online part of the OT protocol is executed: for each $i = 1, \ldots, \ell$, $\mathcal{C}$ masks its input bits x_i with r_i as $X_i = x_i \oplus r_i$ and sends these masked bits to $\mathcal{S}$. $\mathcal{S}$ responds with the masked pair of t'-bit strings $\langle M_i^0, M_i^1 \rangle = \langle m_i^0 \oplus \widetilde{x}_i^0, m_i^1 \oplus \widetilde{x}_i^1 \rangle$ if $X_i = 0$ or $\langle M_i^0, M_i^1 \rangle = \langle m_i^0 \oplus \widetilde{x}_i^1, m_i^1 \oplus \widetilde{x}_i^0 \rangle$ otherwise. $\mathcal{C}$ obtains $\langle M_i^0, M_i^1 \rangle$ and decrypts $\widetilde{x}_i = M_i^{r_i} \oplus m_i$. Using the garbled inputs $\widetilde{\mathbf{x}}^{\ell}, \widetilde{\mathbf{y}}^{\ell}$, $\mathcal{C}$ evaluates the garbled circuit $\widetilde{C}$, obtains the result from the output decryption table and sends it back to $\mathcal{S}$. Overall, in the online phase $\ell t' + 2\ell t' + 1 \sim 3\ell t$ bits are sent.

Cost Evaluation. In the following we show how the GC-based comparison protocol outperforms those based on homomorphic encryption:

Computation Complexity. As our improved GC for integer comparison consists of no more than ℓ non-XOR 2-to-1 gates (cf. comparison circuit in §3.2), $\mathcal{C}$ needs to invoke the underlying cryptographic hash-function (e.g., SHA-256 for $t = 128$ bit symmetric security) exactly ℓ times to evaluate the GC (cf. §2.4). All other operations are negligible (XOR of t-bit strings). Hence, the computational complexity of the online phase of our protocol is negligible as compared to that of protocols based on homomorphic encryption. Even with an additional setup phase, those protocols need to invoke a few modular operations for each input bit which are usually by several orders of magnitude more expensive than the evaluation of a cryptographic hash function used in our protocols. Further the computational complexity of the setup phase in our protocol is more efficient than in protocols based on homomorphic encryption when using efficient OT protocols implemented over elliptic curves and efficient extensions of OT for a large number of inputs (cf. §2.3).

Communication Complexity. Table 3 shows that also the communication complexity of our protocol is much lower than that of previous protocols which are based on homomorphic encryption. As underlying $\mathrm{OT}^{\ell}_{t'}$ protocol we use the protocol of [2] implemented over a suitably chosen elliptic curve and using point compression (see the full version of this paper [32] for details). This protocol has asymptotic communication complexity $12\ell t$ bits and is secure in the standard model. (Using the protocol of [38] which is secure in the random oracle model would result in communication complexity $6\ell t$ bits and much lower computation complexity.) The chosen values for the security parameters correspond to standard recommendations for short-term (upto 2010), medium-term (upto 2030) and long-term security (after 2030) [23].

Table 3. Asymptotic communication complexity of comparison protocols on ℓ-bit values. Parameters defined in §2.1: $\ell = 16, \kappa = 40$, short-term security: $t = 80, T = 1024$, medium-term security: $t = 112, T = 2048$, long-term security: $t = 128, T = 3082$.

Communication	Previous Work			This Work		
Complexity	[20]	[6]	[14]	Setup Phase	Online Phase	Total
Asymptotic	$(\kappa+1)\ell T$	$4\ell T$	$2\ell T$	$16\ell t$	$3\ell t$	$19\ell t$
short-term	82 kByte	8 kByte	4 kByte	2.5 kByte	0.5 kByte	3.0 kByte
medium-term	164 kByte	16 kByte	8 kByte	3.5 kByte	0.7 kByte	4.2 kByte
long-term	246 kByte	24 kByte	12 kByte	4.0 kByte	0.8 kByte	4.8 kByte

4.2 First-Price Auctions

In standard auction systems such as ebay, the auctioneer learns the inputs of all bidders and hence can deduce valuable information about the bidding behavior of unsuccessful bidders or cheat while computing the auction function depending on bidders' input values. To overcome this, a secure protocol can be used instead. Bidders provide their bids in a "smartly" encrypted form to the protocol which allows the auctioneer to compute the auction function without learning the bids. In the following we show how our constructions can be used to improve two previously proposed secure auction systems: one in which all bids are collected before the auction function is computed (Offline Auctions), and another one where bids are input dynamically and the current highest bid is published (Online Auctions).

Offline Auctions. In the offline auction system of [39], the auction function is computed by two parties, an *auction issuer* and the *auctioneer*, who are assumed not to collude. The auction issuer creates a garbled circuit which computes the auction function and sends it to the auctioneer. For each of the bidders' input bits b, a proxy-OT protocol is run, where the auction issuer inputs the two complementary garbled input values $\widetilde{b}^0, \widetilde{b}^1$ of the garbled circuit, the bidder inputs b and the auctioneer obtains the corresponding garbled value $\widetilde{b}$. Then, the auctioneer evaluates the garbled circuit on the garbled inputs and obtains the outcome of the auction as output.

In order to run a first-price auction which outputs the maximum bid and the index of the maximum bidder, our improved minimum circuit of §3.3 can be used. This circuit is substantially smaller and hence the resulting protocol is more efficient than the circuit used in [39] as shown in Table 2.

Online Auctions. In the following we show that our GC-based comparison protocol outperforms the comparison protocol of Damgård, Geisler and Kroigård presented in [14] for the online auction scenario.

The auction system proposed in [14,15,16] extends the idea of splitting the computation of the auction function between two parties, the auctioneer (called

server) and another party (called *assisting server*) who are assumed not to collude. Each bidder can submit a maximum bid b which he secret-shares between server and assisting server over respective secure channels. Afterwards, the bidder can go offline, while the server and assisting server run a secure comparison protocol to compare the secret-shared maximum bid with the publicly known value of the currently highest bid to keep track which bidder is still "in the game". A detailed description of the scenario can be found in [16].

Our protocol uses the efficient comparison protocol of §4.1 with inputs given in different forms: the bid is secret-shared between both players (cf. §A.2 for simple folklore technique to use such inputs in GC) and the other input is publicly known to both parties (e.g., can be treated as a private input of circuit constructor $\mathcal{S}$). The resulting circuit-based protocol for online auctions has exactly the same performance as our solution for the Millionaires problem described in §4.1 with the same efficiency improvements over previous solutions. In particular, the possibility to move all expensive operations into the setup phase, which can be executed during idle times (whenever no new bids are received), is very beneficial for this application as this enables the bidders to instantly see if their current bid was successful or if another bidder meanwhile gave a higher bid. This feature is important towards the end of the auction, where the frequency of bids is high. We further recall that the workload of the setup phases can be reduced by extending OTs efficiently (cf. §2.3).

4.3 Minimum Distance

Finally, we give an efficient protocol for secure computation of the *minimum distance* (or *nearest neighbor*) between a private query point Q, held by client $\mathcal{C}$, and an ordered list of private points $P_0, \ldots, P_{n-1}$ (called *database*), held by server $\mathcal{S}$. The protocol consists of two sub-protocols: the first sub-protocol computes for $i = 1, \ldots, n$ the encrypted distance $[\![\delta_i]\!]$ of the query point Q to each point P_i in the database, using a suitably chosen homomorphic encryption scheme, and outputs these encrypted distances to $\mathcal{S}$. The second sub-protocol securely selects the minimum value and index of these encrypted distances and outputs the minimum distance δ_{min} and minimum index i_{min} to $\mathcal{C}$.

Distance Computation. We sketch the sub-protocols to securely compute the distance $[\![\delta_i]\!]$ between query point Q and points P_i in the database next.

Hamming Distance. The Hamming distance between two points $P = (p_1, \ldots, p_m)$ and $Q = (q_1, \ldots, q_m)$ with $p_j, q_j \in \{0, 1\}$ is defined as $d_H(P, Q) := \sum_{j=1}^{m} p_j \oplus q_j = \sum_{j=1}^{m} (1-p_j)q_j + p_j(1-q_j)$. With an additively homomorphic cryptosystem, the Hamming distance can be computed as follows: $\mathcal{C}$ generates a public-key pk and corresponding secret-key and sends (the verifiably correct) pk and bitwise homomorphic encryptions of Q, $[\![q_1]\!], \ldots, [\![q_m]\!]$, to $\mathcal{S}$. As computing the Hamming distance is a linear operation, S can compute the encrypted Hamming distance to each point $P = P_i$ in its database as $[\![\delta_i]\!] = [\![d_H(P, Q)]\!]$ from $[\![q_j]\!]$ and p_j using standard techniques as proposed in [28].

Euclidean Distance. The Euclidean distance can be seen as an extension of the Hamming distance from 1-bit coordinates to ℓ-bit coordinates, i.e., for $j = 1, \ldots, m : p_j, q_j \in \{0,1\}^\ell$. The Euclidean distance is then defined as $d_E(P,Q) := \sqrt{\sum_{j=1}^{m}(p_j - q_j)^2}$. As the Euclidean distance is not negative, it is sufficient to compute the square of the Euclidean distance instead, in order to find the minimum (or maximum) Euclidean distance: $d_E(P,Q)^2 = \sum_{j=1}^{m}(p_j - q_j)^2$. The encryption of the square of the Euclidean distance $[\![\delta_i^2]\!] = [\![d_E(P_i,Q)^2]\!]$ can be computed analogously to the protocol for the Hamming distance by using additively homomorphic encryption which allows for at least one multiplication (cf. §2.2). Alternatively, when using an additively homomorphic encryption scheme, one can run an additional round for multiplication as used in [18].

Minimum Selection. After having securely computed the homomorphically encrypted distances $[\![\delta_i]\!]$ held by $\mathcal{S}$, the minimum and minimum index of these values can be selected by converting these homomorphically encrypted values to garbled values as described in §A.3 and securely evaluating the minimum circuit of §3.3. The asymptotic communication complexity of this minimum selection protocol is $13\ell nt$ bits for the garbled circuits (when GCs are pre-computed), n homomorphic ciphertexts, and $\mathrm{OT}_{t'}^{n\ell}$. The number of homomorphic ciphertexts can be further reduced using packing (§A.3), and the number of OTs can be reduced to a constant number of OTs (§2.3). As for the other application scenarios described before, all expensive operations can be moved into a setup phase and the entire protocol has a constant number of rounds.

Our minimum selection protocol can also be used as a provably secure[5] replacement for the minimum selection protocol of [30], which was used in the context of privacy-preserving benchmarking. In this scenario, mutually distrusting companies want to compare their key performance indicators (KPI) with the statistics of their peer group using an untrusted central server.

Acknowledgements. We thank anonymous reviewers of CANS 2009 for helpful comments, in particular for pointing out previous work by Boyar et al.

References

1. L. v. Ahn, N. J. Hopper, and J. Langford. Covert two-party computation. In *ACM Symposium on Theory of Computing (STOC'05)*, pages 513–522. ACM, 2005.
2. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.
3. F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. Cryptology ePrint Archive, Report 2008/422, 2008.

[5] The minimum selection protocol of [30] requires multiplicative-blinding of an additively homomorphically encrypted value which reveals some information about the magnitude of the blinded value.

4. M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, LNCS. Springer, 2009. Full version available at http://eprint.iacr.org/2009/195.
5. D. Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.
6. I. F. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology – ASIACRYPT'04*, volume 3329 of *LNCS*, pages 515–529. Springer, 2004.
7. I. F. Blake and V. Kolesnikov. Conditional encrypted mapping and comparing encrypted numbers. In *Financial Cryptography and Data Security (FC'06)*, volume 4107 of *LNCS*, pages 206–220. Springer, 2006.
8. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography (TCC'05)*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005.
9. J. Boyar, I. Damgård, and R. Peralta. Short non-interactive cryptographic proofs. *Journal of Cryptology*, 13(4):449–472, 12 2000.
10. J. Boyar and R. Peralta. Short discreet proofs. In *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *LNCS*, pages 131–142. Springer, 1996.
11. J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. *Theor. Comput. Sci.*, 235(1):43–57, 2000.
12. J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *ACM Conference on Computer and Communications Security (CCS'07)*, pages 498–507. ACM, 2007.
13. G. Di Crescenzo. Private selective payment protocols. In *Financial Cryptography and Data Security (FC'00)*, volume 1962 of *LNCS*, pages 72–89. Springer, 2000.
14. I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy (ACISP'07)*, volume 4586 of *LNCS*, pages 416–430. Springer, 2007.
15. I. Damgård, M. Geisler, and M. Krøigaard. A correction to "efficient and secure comparison for on-line auctions". Cryptology ePrint Archive, Report 2008/321, 2008.
16. I. Damgård, M. Geisler, and M. Krøigaard. Homomorphic encryption and secure comparison. *Journal of Applied Cryptology*, 1(1):22–31, 2008.
17. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public-Key Cryptography (PKC'01)*, LNCS, pages 119–136. Springer, 2001.
18. Z. Erkin, M. Franz, J. Guajardo, Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies (PET'09)*, volume 5672 of *LNCS*, pages 235–253. Springer, 2009.
19. J. Feigenbaum, B. Pinkas, R. S. Ryger, and F. Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols (SMP)*. ECRYPT, 2004.
20. M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Cryptographer's Track at RSA Conference (CT-RSA'01)*, volume 2020 of *LNCS*, pages 457–472. Springer, 2001.
21. J. A. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography (PKC'07)*, volume 4450 of *LNCS*, pages 330–342. Springer, 2007.
22. C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC'09)*, pages 169–178. ACM, 2009.

23. D. Giry and J.-J. Quisquater. Cryptographic key length recommendation, March 2009. http://keylength.com.
24. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
25. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology – EUROCRYPT'08*, volume 4965 of *LNCS*, pages 289–306. Springer, 2008.
26. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*. Springer, 2003.
27. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.
28. A. Jarrous and B. Pinkas. Secure hamming distance based computation and its applications. In *Applied Cryptography and Network Security (ACNS'09)*, volume 5536 of *LNCS*, pages 107–124. Springer, 2009.
29. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of the SSSR Academy of Sciences*, 145:293–294, 1962.
30. F. Kerschbaum. Practical privacy-preserving benchmarking. *International Information Security Conference (SEC'08)*, 278:17–31, 2008.
31. V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology – ASIACRYPT'05*, volume 3788 of *LNCS*, pages 136–155. Springer, 2005.
32. V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. Cryptology ePrint Archive, Report 2009/411, 2009.
33. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.
34. Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063, Electronic Colloquium on Computational Complexity (ECCC), 2004.
35. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.
36. H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology – ASIACRYPT'03*, volume 2894 of *LNCS*. Springer, 2003.
37. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX*, 2004.
38. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.
39. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.
40. J. B. Nielsen and C. Orlandi. Lego for two-party secure computation. In *Theory of Cryptography (TCC'09)*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.
41. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

42. A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *Applied Cryptography and Network Security (ACNS'09)*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.
43. Y. Qi and M. J. Atallah. Efficient privacy-preserving k-nearest neighbor search. In *International Conference on Distributed Computing Systems (ICDCS'08)*, pages 311–319. IEEE, 2008.
44. M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. In *International Conference on Data Mining - Workshops (ICDMW'06)*, pages 541–545. IEEE, 2006.
45. A. C. Yao. Protocols for secure computations. In *Symposium on Foundations of Computer Science (SFCS'82)*, pages 160–164. IEEE, 1982.
46. A. C. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

A Input/Output Conversion Protocols

A.1 Private Inputs and Outputs

Private $\mathcal{S}$ Input: Inputs privately known to the circuit constructor $\mathcal{S}$ are easiest to deal with. For each of these inputs i, $\mathcal{S}$ sends the garbled value $\widetilde{w}_i^{v_i}$ corresponding to the plain value v_i to evaluator $\mathcal{C}$. As described in [42], in case of semi-honest constructor (i.e., with no cut-and-choose), the inputs of $\mathcal{S}$ can also be securely incorporated into the garbled circuit. This optimization avoids to transfer any additional data for $\mathcal{S}$'s private inputs and the size of the GC can be reduced as well. However, in many applications it is beneficial even in the semi-honest scenario to separate conversion of the inputs from creation of the garbled circuit, as this allows $\mathcal{S}$ to create the garbled circuit in an offline pre-computation phase already before its private inputs are known.

Private $\mathcal{C}$ Input: For private inputs w_i of the evaluator $\mathcal{C}$, both parties execute an OT protocol for each input bit in which constructor $\mathcal{S}$ inputs the two garbled t'-bit values $\widetilde{w}_i^0, \widetilde{w}_i^1$ and $\mathcal{C}$ inputs its plain value v_i to obtain $\widetilde{w}_i^{v_i}$ as output. For ℓ input bits, the OTs can be executed in a parallel $\mathrm{OT}_{t'}^{\ell}$ protocol which can efficiently be extended to OT_t^t as described in 2.3.

Private $\mathcal{S}$ Output: If the output of the functionality is a private output w_i of the evaluator $\mathcal{C}$, constructor $\mathcal{S}$ provides $\mathcal{C}$ with the output decryption table for w_i, i.e., the permutation bit π_i chosen when creating the garbled value $\widetilde{w}_i^0 = \langle k_i^0, \pi_i \rangle$.

Private $\mathcal{C}$ Output: For private outputs w_i of the constructor $\mathcal{S}$, evaluator $\mathcal{C}$ does not get an output decryption table but sends the obtained permutation bit π_i of the obtained garbled value $\widetilde{w}_i = \langle k_i, \pi_i \rangle$ back to $\mathcal{S}$ who can deduce the corresponding plain value from this. Clearly, this works only if $\mathcal{C}$ is semi-honest as otherwise he could easily flip the output bit. This can be prevented by requiring $\mathcal{C}$ to send the output key k_i instead.

A.2 Secret-Shared Inputs and Outputs

Secret-Shared Input: As proposed in [19], a bit b can be secret-shared between $\mathcal{C}$ holding share $b_{\mathcal{C}}$ and $\mathcal{S}$ holding share $b_{\mathcal{S}}$, with $b = b_{\mathcal{C}} \oplus b_{\mathcal{S}}$. A secret-shared input bit b can be converted into a garbled input $\widetilde{b}$ using an $\mathrm{OT}^{\ell}_{\ell'}$ protocol: $\mathcal{C}$ inputs $b_{\mathcal{C}}$ and $\mathcal{S}$ inputs the two corresponding garbled values in the usual order $\widetilde{b}^0, \widetilde{b}^1$ if $b_{\mathcal{S}} = 0$ or swaps them to $\widetilde{b}^1, \widetilde{b}^0$ otherwise. It is easy to verify that $\mathcal{C}$ obliviously obtains the correct garbled value $\widetilde{b}$ for the shared bit b.

Secret-Shared Output: A similar method can be used for a secret-shared output bit b. $\mathcal{S}$ chooses a random share $b_{\mathcal{S}}$ and provides $\mathcal{C}$ with an output decryption table (cf. private output to $\mathcal{C}$) in the correct order in case $b_{\mathcal{S}} = 0$ or with swapped entries otherwise. $\mathcal{C}$ decrypts the garbled output to $b_{\mathcal{C}}$ which satisfies $b = b_{\mathcal{C}} \oplus b_{\mathcal{S}}$.

A.3 Homomorphically Encrypted Inputs and Outputs

In the scenario of secure two-party computation based on homomorphic encryption, one party, say client $\mathcal{C}$, generates a key-pair of the homomorphic encryption scheme and sends the (verifiably correct) public key and its inputs encrypted under the public key to $\mathcal{S}$. Afterwards, $\mathcal{S}$ can perform operations on the ciphertexts which result in corresponding operations on the encrypted plaintext data (using the homomorphic property of the cryptosystem). In order to compute operations that are not compatible with the homomorphic property (e.g., multiplication of two ciphertexts encrypted with an additively homomorphic encryption scheme), additional communication rounds must be performed. In the following we show how computing on homomorphically encrypted data can be combined with a garbled circuit to efficiently evaluate non-linear functions, such as comparison, minimum search, or other functionalities in a constant number of rounds.

Homomorphically Encrypted Input: If $\mathcal{S}$ holds an ℓ-bit value $[\![\mathbf{x}^{\ell}]\!]$, additively homomorphically encrypted under $\mathcal{C}$'s public key, this value can be converted into a garbled value $\widetilde{\mathbf{x}}^{\ell}$ output to $\mathcal{C}$ as follows: $\mathcal{S}$ chooses a random value r from the plaintext space P and adds this to the encrypted value: $[\![y]\!] = [\![\mathbf{x}^{\ell} + r]\!]$. In order to avoid an overflow, this requires that $\ell + \kappa \leq |P|$ for a statistical correctness parameter κ (e.g., $\kappa = 40$). $\mathcal{S}$ sends $[\![y]\!]$ to $\mathcal{C}$ who decrypts into y. Afterwards, both parties evaluate a garbled circuit which takes off the additive blinding: the private input of $\mathcal{S}$ into this garbled circuit are the ℓ least significant bits of r, $\mathbf{r}^{\ell} = r \mod 2^{\ell}$, and $\mathcal{C}$ inputs the ℓ least significant bits of y, $\mathbf{y}^{\ell} = y \mod 2^{\ell}$. The garbled circuit is an ℓ-bit subtraction circuit (cf. §3.1) which recovers the plaintext value from the blinded value as $\widetilde{x}^{\ell} = \widetilde{\mathbf{y}}^{\ell} - \widetilde{\mathbf{r}}^{\ell}$. This conversion protocol from additively homomorphically encrypted values into garbled values was used in [12,28]. A detailed proof and efficiency improvements (by packing together multiple values and converting the packed value at once) is given in [4].

Homomorphically Encrypted Output: This is similar to an homomorphically encrypted input (add random $(\ell + \sigma)$-bit mask in GC and remove it under homomorphic encryption) and included in the full version of this paper [32].