Modular Design of Efficient Secure Function Evaluation Protocols

Vladimir Kolesnikov
Systems and Security Research
Alcatel-Lucent Bell Laboratories
Murray Hill, NJ, USA
kolesnikov@research.bell-labs.com

Ahmad-Reza Sadeghi and Thomas Schneider
System Security Lab
Ruhr-University Bochum
Bochum, Germany
{*ahmad.sadeghi,thomas.schneider*}*@trust.rub.de*

***Abstract*—Two-party Secure Function Evaluation (SFE) allows mutually distrusting parties to (jointly) correctly compute a function on their private input data, without revealing the inputs. SFE, properly designed, guarantees to satisfy the most stringent security requirements, even for interactive computation. Two-party SFE can benefit almost any client-server interaction where privacy is required, such as banking, TV, targeted advertisements, etc. Today, SFE is a subject of immense amount of research in a variety of directions, and is not easy to navigate.**

In this work, we systematize some of the vast SFE research knowledge. It turns out that the most efficient SFE protocols are obtained by combining several basic techniques, such as Garbled Circuit (GC) and computation under Homomorphic Encryption (HE). We show how these techniques can be viewed as building blocks with clean interfaces, which can be easily combined for a complete efficient solution. Further, our approach naturally lends itself to automated protocol generation (compilation). We believe, today, this approach is the best candidate for implementation and deployment.

We consider passive and active cheating, and give exact computation and communication costs of the building blocks.

***Keywords*-protocol design; privacy-preserving protocols; homomorphic encryption; garbled circuits; garbled OBDDs;**

I. Introduction

The concept of *Secure Two-Party Computation* was introduced in 1982 by Andrew Yao [1]. The basic idea is to let two mutually mistrusting parties compute an arbitrary function on their private inputs without revealing any information about their input beyond the output of the function. Since then this concept has been an appealing research subject in cryptographic and security community with many interesting and exciting results.

Although a large number of security-critical applications (e.g., electronic auctions and voting, data classification, remote diagnostics, etc.) with sophisticated privacy and security requirements can benefit from secure computation, its deployment for real world applications was believed to be very limited and expensive for a relatively long time. However, this has dramatically changed in the recent years thanks to many algorithmic improvements and automatic tools, as well as faster computing platforms and communication networks.

The aim of this paper is to put together and in perspective the main current approaches to *efficient* secure two-party computation. We demonstrate that in most cases, the most efficient protocols are simply a combination of a few basic techniques. We build our presentation in the style of a tutorial, and aim for the paper to be understandable to non-specialists in secure computation.

One of the main reasons for this work is to clarify the unsubstantiated, but popular, belief that generic techniques, such as Garbled Circuit (GC) introduced by Yao in 1986 [2], are too slow, and cannot be used in practice due to their inefficiency. This state of affairs is evidenced by authors' personal communications, and by the abundance of submissions and publications in security and cryptography conferences, where the proposed solutions, often entirely based on homomorphic encryption (HE), fall far behind GC in efficiency and are subsequently replaced with their GC-based equivalents (e.g., [3]). Authors often assume that the generality of GC necessarily makes it inefficient. One of our aims is to explain and to promote GC as one of the most efficient and versatile techniques today. We note that with the recent GC improvements (e.g., [4]–[6]), it is increasingly difficult to outperform GC even by special-purpose solutions.

Our remaining goal is to establish and promote the efficiency baseline, against which future work could be measured and improved.

A. Background: Where SFE Fits in Secure Computing

Cryptography (from Greek "secret writing") with thousands of years of history [7] has emerged as a tool for secret communication. However, only recently, with the development of fast computing devices, has cryptography grown into a structured and mathematical science. The science of secret communications became more formal and rigorous, and, simultaneously, new directions of cryptography appeared and developed. Modern cryptography encompasses much more than the original intent. Examples of new directions include ability to prove knowledge of secrets without revealing any information about them, means of electronic identification, secure financial transactions, and much more.

The state of modern communications allows easy access to almost any imaginable resource or person. At the same time, the underlying connectivity layer provides weak, if any, guarantees. For example, if Alice sends a message to Bob, this message not only may be lost, it may also be read and, more importantly, modified by an adversary, while in transit. While most Internet traffic is of little or no interest to attackers, a portion of it serves transactions of value, and requires strong security. Protection against eavesdropping and interference with the legitimate communication is relatively well understood and remains perhaps the most commonly used fruit of cryptography.

However, even a perfectly secure communication system is only a part of the solution. Imagine a situation where Alice participates in a transaction with Bob, but does not completely trust him. This occurs in many settings where the participants may have conflicting interests, including contract signing, buy/sell transactions, outsourcing computation or storage to untrusted servers, etc. Securing the communication channel cannot provide any assurance that Bob does not cheat. Can we protect Alice's (and everyone else's) interests in this setting? A study of *Secure Function Evaluation* (SFE), which began in the 1980's, emerged from the need not only to communicate, but also to *compute* securely. It addresses the problem of providing security against cheating participants of the computation.

B. Applications

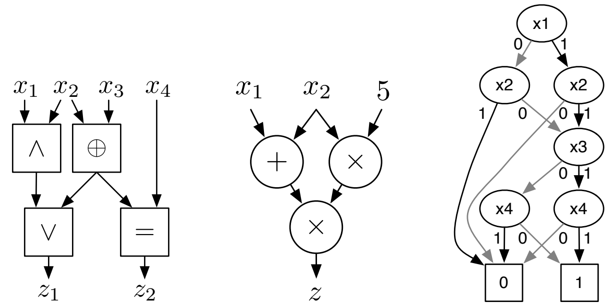
There is a large body of literature on SFE applications, in particular those with strong privacy requirements such as Privacy-Preserving Genomic Computation [3], [8], [9], Remote Diagnostics [10], Graph Algorithms [11], Data Mining [12], [13], Credit Checking [14], Medical Diagnostics [15], Face Recognition [16], [17], or Policy Checking [18]–[20], just to name a few. Here different cryptographic techniques such as Garbled Circuits (GC) and/or Homomorphic Encryptions (HE) are used. Recently, verifiable outsourcing of computations for cloud-computing applications has been proposed, based on a combination of algebraically homomorphic encryption and garbled circuits [21][1]. Existence of a variety of SFE compilers, coming from both academia, e.g. [23], and industry [24], further proves significant interest in the SFE technology.

Moreover, we note that secure two-party protocols can often be naturally extended to secure multi-party protocols. Examples include secure mobile agents which can be based on HE [25] and GC [26], as well as privacy-preserving auction systems based on GC [27] or HE [28].

C. Outline of the Presentation

We start our discussion in §II with a few of most popular function representations, and pointing out their relative advantages in terms of possibility of efficient secure evaluation. We note that it is possible to "mix-and-match" the representations in construction of the protocol. Then, in §III, we discuss various notions of security and their relationship. In §IV, we describe today's efficient SFE constructions for each of the function representations we consider. We aim to present our constructions modularly, with clean "interfaces", so that it remains intuitive that they can be composed. We handle actual details of the composition, namely the techniques to convert encrypted intermediate values between the the protocols in §V (semi-honest players) and §VI (malicious players).

[1]Because of its complexity and novelty, the practical performance of algebraically homomorphic encryption [22] is still under investigation.



(a) Boolean Circuit (b) Arithmetic Circuit (c) OBDD

Figure 1. Function Representations

II. Function Representations

Given the function to be securely computed, the first decision we face is the choice of the "programming language" for describing the function. It turns out that this decision has a major impact on the efficiency of the final solution. Further, it is not feasible to describe the optimal choice strategy as finding minimal function representations is hard [29], [30]. In this section we describe several function representations considered in SFE, and give general guidelines with respect to the efficiency choices.

We note that the cost of implementing SFE protocols varies greatly among the function representations. For example, boolean circuit is securely evaluated using GC technique, which is much more efficient than techniques for evaluating arithmetic circuits. However, some functions are represented much more compactly as an arithmetic circuit. In this work, we explain and advocate a hybrid approach, where function blocks can be evaluated using different techniques, and their encrypted intermediate results then glued together.

We now proceed to summarizing function representations which are particularly useful for secure computation: boolean circuits, arithmetic circuits and ordered binary decision diagrams.

A. Boolean Circuits

Boolean circuits is a classical representation of functions in engineering and computer science.

A *boolean circuit* with u inputs, v outputs and k gates is a *directed acyclic graph* (DAG) with $|V| = u+v+k$ vertices (nodes) and $|E|$ edges. Each node corresponds to either a *gate*, an *input* or an *output*. The edges are called *wires*. For simplicity, the input- and output nodes are often omitted in the graphical representation of a boolean circuit as shown in Fig. 1(a). For a more detailed definition see [31].

A d-input gate G computes a d-ary boolean function $g : \{0,1\}^d \to \{0,1\}$. Typical gates are XOR ($\oplus$), XNOR ($=$), AND ($\wedge$), OR ($\vee$); gates are often specified by their function table, which contains 2^d entries.

Gates of the boolean circuit can be evaluated in any order, as long as all of the current gate inputs are available. This property is ensured by sorting (and evaluating) the gates topologically, which can be done efficiently in $O(|V|+|E|)$ [32, Topological sort, pp. 549-552]. The topologic order of a boolean circuit indexes the gates with labels $G_1, \ldots, G_k$ and ensures that the i-th gate G_i has no inputs that are outputs of a successive gate $G_{j>i}$. In complexity theory, a circuit with such a topologic order is called a *straight-line program* [33]. Given the values of the inputs, the output of the boolean circuit can be evaluated by evaluating the gates one-by-one in topologic order. A valid topologic order for the example boolean circuit in Fig. 1(a) would be $\wedge, \oplus, \vee, =$. The topologic order is not necessarily unique.

Automatic Generation: Boolean circuits can be automatically generated from a high-level specification of the function. A prominent example is the well-established Fairplay compiler [23]. Fairplay's *Secure Function Description Language* (SFDL) resembles a simplified version of a hardware description language, such as VHDL[2], and supports types, variables, functions, boolean operators $(\wedge, \vee, \oplus, \ldots)$, arithmetic operators $(+, -, *, /)$, comparison $(<, \geq, =, \ldots)$ and control structures like if-then-else or for-loops with constant range (cf. [23, Appendix A] for a detailed description of the syntax and semantics of SFDL). Fairplay also includes a GUI that assists the programmer in creating SFDL programs with graphical code templates. The Fairplay compiler automatically transforms the functionality described as SFDL program into the corresponding boolean circuit.

B. Arithmetic Circuits

Arithmetic circuits is a more compact function representation than boolean circuits.

An *arithmetic circuit* over a ring R and the set of variables $x_1, ..., x_n$ is a directed acyclic graph (DAG). Fig. 1(b) illustrates an example. Each node with in-degree zero is called an input gate labeled by either a variable x_i or an element in R. Every other node is called a gate and labeled by either $+$ or $\times$ denoting addition or multiplication in R.

Any boolean circuit can be expressed as an arithmetic circuit over $R = \mathbb{Z}_2$. However, if we use $R = \mathbb{Z}_m$ for sufficiently large modulus m, the arithmetic circuit can be much smaller than its corresponding boolean circuit, as integer addition and multiplication can be expressed as single operations in $\mathbb{Z}_m$.

Number Representation: We note that arithmetic circuits can simulate computations on both positive and negative integers by mapping them into elements of $\mathbb{Z}_m$ as follows. Zero and positive values are mapped to the elements $0, 1, 2, \ldots$ whereas negative values are mapped to $m-1, m-2, \ldots$. As with all fixed precision arithmetics, overflows or underflows must be avoided.

C. Ordered Binary Decision Diagrams

Another possibility to represent boolean functions are Ordered Binary Decision Diagrams (OBDD) introduced by Bryant [34].

A *binary decision diagram* (BDD) is a rooted, directed acyclic graph (DAG) which consists of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each decision node is labeled by a boolean decision variable and has two child nodes, called low child and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1). An *ordered binary decision diagram* (OBDD) is a BDD in which the decision variables appear in the same order on all paths from the root.

Given an assignment $\langle x_1 \leftarrow b_1, \ldots, x_n \leftarrow b_n \rangle$ to the variables $x_1, \ldots, x_n$, the value of the Boolean function $f(b_1, \ldots, b_n)$ can be found by starting at the root and following the path where the edges on the path are labeled with $b_1, \ldots, b_n$.

Example: Fig. 1(c) shows the OBDD for the function $f(x_1, x_2, x_3, x_4) = (x_1 = x_2) \wedge (x_3 = x_4)$ of four variables x_1, x_2, x_3, x_4 with the total ordering $x_1 < x_2 < x_3 < x_4$.[3] Consider the assignment $\langle x_1 \leftarrow 1, x_2 \leftarrow 1, x_3 \leftarrow 0, x_4 \leftarrow 0 \rangle$. In the OBDD shown in Fig. 1(c), if we start at the root and follow the edges corresponding to the assignment, we end up at the 1-terminal which implies that $f(1,1,0,0) = 1$.

Generalizations: Multiple OBDDs can be used to represent a function g with multiple outputs. If g's outputs can be encoded by k boolean variables, then g can be represented by k OBDDs where the i-th OBDD computes the i-th output bit.

Further generalizations of OBDDs can be obtained by having multiple terminal nodes (called *classification nodes*) and more general branching conditions: In a *Branching Program* [10] the child node is determined depending on the comparison of the ℓ-bit input variable x_{α_i} with a decision node specific threshold t_i. In *Linear Branching Programs* [15] the branching condition is the comparison of the scalar product between the input vector $\mathbf{x}$ of n ℓ-bit values and a decision node specific coefficient vector $\mathbf{a_i}$ with a decision node specific threshold t_i.

[2] Very high speed integrated circuit Hardware Description Language

[3] OBDDs are sensitive to variable ordering, e.g., with the ordering $x_1 < x_3 < x_2 < x_4$ the OBDD for f has 11 nodes.

Efficiency: Although some functions require in the worst case an OBDD of size exponential in the number of inputs, many functions encountered in typical applications (e.g., addition or comparison) have a reasonably small OBDD representation [34].

Even though finding an optimal variable ordering for OBDDs is NP-complete [29], in many practical cases OBDDs can be minimized to a reasonable size. Algorithms to improve the variable ordering of OBDDs are Rudell's sifting algorithm [35], the window permutation algorithm [36], genetic algorithms [37], [38], or algorithms based on simulated annealing [39].

Nevertheless, some functions have a lower bound for the size of the smallest OBDD representation which is exponential. For example n-bit integer multiplication has an exponential size OBDD [40], [41] but requires only one multiplication gate in an arithmetic circuit over a sufficiently large ring. Multiplication within a boolean circuit has complexity $\mathcal{O}(n^2)$ using school method or $\mathcal{O}(n^{\log_2 3})$ with the method of [42]. Fast multiplication methods which apply the Fourier transformation have better asymptotic complexity but hide large constant factors in the $\mathcal{O}$ notation which makes them more efficient for large inputs (thousands of bits) only: $\mathcal{O}(n \log n \log \log n)$ [43] and $n \log n 2^{\mathcal{O}(\mathsf{lg}^* n)}$ [44][4].

III. Secure Function Evaluation

Before presenting SFE protocols in §IV, we discuss the notions of security, give some insight in two-party computation, and introduce notation.

A. Security Notions

In this section, we give intuition for the notions of security that we use. For the lack of space, and because the definitions are standard, we do not include the full formal definitions here. However, we refer the reader to standard sources. We consider *semi-honest*, *covert* and *malicious* players.

Perhaps, the most natural and the strongest notion is the *malicious* adversary. Such attacker is allowed to arbitrarily deviate from the prescribed protocol, aiming to learn private inputs of the parties and/or to influence the outcome of the computation. This is the strongest and most general type of adversary, and, not surprisingly, protection against such attacks is relatively expensive, as we discuss in §VI. To be malicious-secure, a protocol must guarantee that there does not exist a course of action that results in any gain to the attacker.

A somewhat weaker *covert* adversary is similar to malicious, but with the restriction that he must avoid being caught cheating. That is, a protocol in which an active attacker may gain advantage may still be considered secure if attacks are discovered with certain fixed probability (e.g., $1/2$). It is reasonable to assume that in many social, political and business scenarios the consequences of being caught overweight the gain from cheating; we believe covert adversaries is the right way to model the behavior of players in the interactions of interest. At the same time, protocols secure against covert adversaries are substantially more efficient than those secure against malicious players.

Finally, we consider the *semi-honest* adversary, one who does not deviate from the protocol, but aims to learn the output of the computation. At first, it may appear contrived and trivial. Consideration of semi-honest adversaries, however, is important in many typical practical settings. Firstly, even externally unobservable cheating, such as poor random number generation, manipulations under encryption, etc., can be uncovered by an audit or reported by a conscientious insider, and cause negative publicity. Therefore, especially if the gain from cheating is low, it is often reasonable to assume that a well-established organization will exactly follow the protocol (and thus can be modeled as semi-honest). Further, even if players are trusted to be *fully honest*, it is sometimes desired to ensure that the transcript of the interaction reveals no information. This is because in many cases, it is not clear how to reliably delete the transcript due to lack of control of underlying computing infrastructure (network caching, virtual memory, etc.) Running an SFE protocol ensures that player's input cannot be subsequently revealed even by forensic analysis.

At the same time, designing semi-honest-secure SFE protocols is far from trivial, and is in fact an important basic step in the design of the malicious protocols.

1) Intuition for Formalization: Formal definitions of security of SFE are very detailed (pages long) and subtle. Here we convey the basic idea behind the formalization and the employed ideal/real paradigm.

Intuitively, a protocol transcript does not leak player's input, if an indistinguishable (i.e., similar-looking) transcript can be constructed without any knowledge of the input. (We note that the two transcripts, *real* and *simulated*, must look the same to a powerful distinguisher who, in particular, knows the inputs.) It is now intuitive that if the protocol leaks some information on the inputs, there will exist a distinguisher who simply extracts this information from the transcript, and compares to the player's input. Since simulated transcript was constructed without the knowledge of the input, distinguisher will be able to distinguish it from the real one, and such protocol will be insecure by definition. Further, the proof of security for players A and B in the protocol Π consists of constructing such simulator Sim_A, Sim_B, and proving that their output is indistinguishable from the real transcript of the protocol.

The above intuition is sufficient for the formalization of the semi-honest model. However, in the presence of actively cheating players (who can substitute their input, among other things), this does not quite work, as it is not even clear

[4] $\mathsf{lg}^* n = \min_{i \geq 0} \mathsf{lg}^{(i)} n \leq 1, \mathsf{lg}^{(0)} n = n, \mathsf{lg}^{(i+1)} n = \log_2 \mathsf{lg}^{(i)} n.$

if the players indeed evaluate the intended function. Thus, the following extension of the simulation paradigm was introduced. We now define an *ideal* world, where players have very limited cheating powers (they are allowed to abort, substitute their local inputs, and output what they wish), and rely on a trusted party to provide them with the resulting output of the computation over a perfectly secure channel. We say that a real-world protocol Π is secure if for any real-world attacker there is a corresponding ideal-world attacker that can do "the same harm". Since ideal world clearly limits the attack powers, the same limit would apply to the real world. This is formalized by the ability to simulate the real-world transcript (i.e., to generate an indistinguishable transcript) by the ideal-world simulator.

The formal definitions for the semi-honest and malicious player security can be found in [45].

The formalization of the covert adversaries is similar to that of the malicious; the difference is in the definition of the ideal world, where ideal world adversaries are given the option to cheat, but are caught (i.e., their opponent is notified) with certain fixed probability. Other aspects of definition remain the same; because of simulatability properties and the general approach of ideal-real paradigm, a secure real-world covert adversary also may choose to cheat, but be caught by the opponent with the specified probability. The formal definitions for covert security (three variations) were proposed in [46].

We note that SFE protocols will guarantee security for the honestly behaving player who may be engaging with cheating adversary. If both players are deviating from the protocol, definitions provide no guarantees.

2) Hybrid Security: It is often the case that protocol participants are not equal in their capabilities, trustworthiness, and motivation. This is true especially often in the client-server scenarios. For example, it may be reasonable to assume that the bank will not deviate from the protocol (act semi-honestly), but similar assumption should not be made on bank clients, who are not established and may be much more willing to risk committing fraud.

This can be naturally reflected in protocol design and the guarantees given by the protocol. This is because security definitions already separately state security against player A and player B. When proposing a protocol, the security claim may be in the form "Protocol Π is secure against malicious A and semi-honest B." The proof of security then involves two different definitions, and simulator constructions would also be correspondingly different. The benefit of this hybrid approach is the possibility to design significantly more efficient protocols. For example, the garbled circuit protocol (in which players take the roles of garbled circuit constructor or evaluator) is almost free to secure against malicious evaluator, and much more expensive to secure against malicious constructor (details later in §IV-B3). Thus, GC-based protocols are good candidates for settings with corresponding trust relationships, e.g., banking.

B. Computation under encryption

Before presenting the protocols in the next section, we find it instructive to present the following simple insight on two-party SFE. We believe the reader will benefit from keeping it in mind while reading the SFE protocol descriptions. Each of the SFE techniques we consider can be viewed as *evaluation under encryption*. In fact, this holds for any two-party SFE technique.

Intuitively, this is the case since parties must use their entire inputs in the computation, to be able to correctly compute the output. This means that players must exchange messages that are dependent on their inputs. Moreover, the messages must cumulatively contain the entire inputs. (To be more precise, at least one party must send such messages, and the other party may only perform actions based on his inputs.) Because of the input privacy requirements, the messages must not reveal their plaintext content (inputs), and thus are encryptions of the inputs. Further, the message recipient acts based on the message and his input, which amounts to computing under encryption. (We note that this intuition does not hold in secure multiparty computation, where there are three or more players. There, secure computation is possible without sending encryptions of the input and evaluation under encryption.)

We note that evaluation under encryption is very complicated in its generality. In fact, only recently, the first promising candidate was proposed – an encryption scheme that allows to perform an arbitrary number of both multiplications and additions on the plaintext [22]. What we solve is a much simpler problem, where the computed function is fixed. Now, for example, one party can send his encrypted input and then collaborate with the other party to help him evaluate under encryption. We further simplify our work by considering only elementary operations, e.g., boolean gates. If we give a self-composing protocol for evaluation of a few basic gates, the entire SFE problem is solved, since any function can be built of these gates.

C. Parameters and Notation

We denote *symmetric security parameter* by t and the *asymmetric security parameter*, e.g., bitlength of RSA moduli, by T. NIST recommends choosing for short-term security (until year 2010) $t = 80$ and $T = 1024$, for medium-term security (until year 2030) $t = 112$ and $T = 2048$ and for long-term security (after year 2030) $t = 128$ and $T = 3072$. For a detailed summary on various recommendations for security parameters we refer to [47].

The *statistical security parameter* is denoted by σ and can be chosen as $\sigma = 80$ in practice. This parameter controls the statistical distance for blinded values.

We denote the *bitlength* of a variable x with $|x|$.

In the following, we will refer to the two SFE participants as client $\mathcal{C}$ and server $\mathcal{S}$. Our naming choice is mainly influenced by the asymmetry in the SFE protocols, which fits into client-server model. We stress that, while in most of the real-life two-party SFE scenarios the corresponding client-server relationship in fact exists in the evaluated function, we do not limit ourself to this setting.

IV. Building Blocks: SFE Techniques for OBDD, Arithmetic and Boolean Circuits in the Semi-honest Model

As mentioned above, to reduce complexity, functions can be decomposed into several sub-functions (blocks). Each of these blocks can be represented in a different way, e.g., a multiplication block can be represented as an arithmetic circuit, a comparison block as a boolean circuit and a specific decision tree as an ordered binary decision diagram.

In this section, we present the SFE protocols for the three representations we consider. At this time, we only consider semi-honest adversaries. We explain how to prevent/detect deviations from the protocol in §VI.

It is our goal to be able to arbitrarily compose the three protocols. This means that the encrypted output of one protocol will be fed as input into another. To preserve a common interface and simplify the presentation, we will extract and describe separately the core (computation under encryption) of each protocol. (For completeness, we also discuss the simple issue of how to appropriately encrypt the inputs and decrypt the outputs.) Thus, protocol structure will look as follows: encrypt the plaintext inputs, perform the computation under encryption (which may include a composition of encrypted computations), and, finally decrypt the output value. We will discuss the issues of composition of the protocols, such as conversions of encryptions, in §V.

A. Homomorphic Encryption and Evaluation of Arithmetic Circuits

In this section, we describe semantically secure homomorphic encryption schemes and how they can be used for secure evaluation of arithmetic circuits.

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme with plaintext space P and ciphertext space C. We write $[\![m]\!]$ for $\mathsf{Enc}(m, r)$.

1) Additively Homomorphic Cryptosystems: An *additively homomorphic* encryption scheme allows addition under encryption as follows. It defines an operation $+$ on plaintexts and a corresponding operation $\boxplus$ on ciphertexts, satisfying $\forall x, y \in P : [\![x]\!] \boxplus [\![y]\!] = [\![x + y]\!]$. This naturally allows for multiplication with a plaintext constant a using repeated doubling and adding: $\forall a \in \mathbb{Z}, x \in P : a[\![x]\!] = [\![ax]\!]$.

Popular instantiations for additively homomorphic encryption schemes are summarized in Table I: The Paillier cryptosystem [48] provides a T-bit plaintext space, where T is the size of the RSA modulus N, and is sufficient for most applications. The Damgård-Jurik cryptosystem [49] is a generalization of the Paillier cryptosystem which provides a large plaintext space of size sT-bit for arbitrary $s \geq 1$. The Damgård-Geisler-Krøigaard (DGK) cryptosystem [28], [50], [51] has smaller ciphertexts but can be used with a small plaintext space only as decryption requires to solve a discrete log.

Table I
Additively Homomorphic Encryption Schemes with N: RSA modulus, $s \geq 1$, u: small prime.

Scheme	P	C	$\mathsf{Enc}(m, r)$
Paillier [48]	$\mathbb{Z}_N$	$\mathbb{Z}^*_{N^2}$	$g^m r^N \bmod N^2$
Damgård-Jurik [49]	$\mathbb{Z}_{N^s}$	$\mathbb{Z}^*_{N^{s+1}}$	$g^m r^{N^s} \bmod N^{s+1}$
DGK [28], [50], [51]	$\mathbb{Z}_u$	$\mathbb{Z}^*_N$	$g^m h^r \bmod N$

Homomorphic Cryptosystems with Addition and Multiplication: For the sake of completeness, we mention that some cryptosystems allow addition and multiplication under encryption. For this, a separate operation $\times$ for multiplication of plaintexts and a corresponding operation $\boxtimes$ on ciphertexts is defined satisfying $\forall x, y \in P : [\![x]\!] \boxtimes [\![y]\!] = [\![x \times y]\!]$. Cryptosystems with such a property are called "fully" or "algebraically" homomorphic.

A possible instantiation is the cryptosystem of [52] which allows for an arbitrary number of additions and one multiplication. *Algebraically homomorphic* cryptosystems allow to evaluate an arbitrary number of additions and multiplications on ciphertexts. Possible candidates are the cryptosystem of [53] (boolean operations only) and [54] (size of ciphertexts grows exponentially in the number of multiplications for both). The most recent candidates are the schemes of [22], [55], [56] without such a restriction. We note that the size of ciphertexts and computational cost of elementary steps in fully homomorphic cryptosystems is *substantially* larger than that of the purely additively homomorphic schemes. Taking in consideration that for today's SFE applications, even a single public-key operation per gate is most often too expensive, we don't see fully homomorphic schemes being used for SFE in practical applications in the near future. Although we concentrate on the additively homomorphic Paillier cryptosystem in the following, our framework can be used together with fully homomorphic schemes as well.

2) Computing on Encrypted Data: Homomorphic encryption naturally allows to evaluate arithmetic circuits via computation on encrypted data, as follows. The client $\mathcal{C}$ generates a key pair for a homomorphic cryptosystem and sends his inputs encrypted under the public key to the server $\mathcal{S}$ together with the public key. With an fully homomorphic scheme, $\mathcal{S}$ can simply evaluate the arithmetic circuit by computing on the encrypted data and send back the (encrypted) result to $\mathcal{C}$, who then decrypts it to obtain the output. If the homomorphic encryption scheme only supports addition, one round of interaction between $\mathcal{C}$ and $\mathcal{S}$ is needed to evaluate each multiplication gate (or a layer of multiplication gates), e.g., as described later in this section (and also in §VI-B). We note that today, the latter is a much faster SFE approach than using fully homomorphic schemes.

3) Packing: Often the plaintext space P of the homomorphic encryption scheme is substantially larger than the size of encrypted numbers. This allows for optimization of many protocols based on homomorphic encryption by packing together multiple ciphertexts into one before or after additive blinding and sending back the single ciphertext from $\mathcal{S}$ to $\mathcal{C}$ instead. This substantially decreases the size of the messages sent from $\mathcal{S}$ to $\mathcal{C}$ as well as the number of decryptions performed by $\mathcal{C}$. The computational overhead for $\mathcal{S}$ is small as packing the ciphertexts $[\![x_1]\!], ..., [\![x_n]\!]$ into one ciphertext $[\![X]\!] = [\![x_n|| \ldots ||x_1]\!]$ costs less than one full-range modular exponentiation with Horner's scheme: $[\![X]\!] = [\![x_n]\!]$; for $i = n-1$ downto $1 : [\![X]\!] = 2^{|x_{i+1}|}[\![X]\!] \boxplus [\![x_i]\!]$.

4) Multiplication of Homomorphic Values with Additively-Homomorphic Encryption: To multiply two homomorphic ℓ-bit values $[\![x]\!]$ and $[\![y]\!]$ held by $\mathcal{S}$ the following standard protocol requires one single round of interaction with $\mathcal{C}$: $\mathcal{S}$ randomly chooses $r_x, r_y \in_R \{0,1\}^{\ell+\sigma}$, where σ is the statistical security parameter, computes the blinded values $[\![\bar{x}]\!] = [\![x + r_x]\!], [\![\bar{y}]\!] = [\![y + r_y]\!]$ and sends these to $\mathcal{C}$. $\mathcal{C}$ decrypts, multiplies and sends back $[\![z]\!] = [\![\bar{x}\bar{y}]\!]$. $\mathcal{S}$ obtains $[\![xy]\!]$ by computing $[\![xy]\!] = [\![z]\!] \boxplus (-r_x)[\![y]\!] \boxplus (-r_y)[\![x]\!] \boxplus [\![-r_x r_y]\!]$.

Packing can be used to improve efficiency of parallel multiplications by packing multiple blinded ciphertexts together instead of sending them to $\mathcal{C}$ separately.

5) Homomorphic Values and Conversions: Finally, we mention a few relatively simple issues and optimizations with encrypting the input, and decrypting the output of the homomorphic computation. Describing these procedures completes (at a high level) the description of SFE of arithmetic circuits.

The interface for SFE protocols based on homomorphic encryption are *homomorphic values*, i.e., a homomorphic encryption *held by* $\mathcal{S}$ encrypted under the public key of $\mathcal{C}$ (see Fig. 3 in §V). These homomorphic values can be converted from or to plaintext values as described next.

Plain Value to Homomorphic Value for Inputs: To convert a plain ℓ-bit value x into a homomorphic value $[\![x]\!]$, it is encrypted under $\mathcal{C}$'s public key (and sent to $\mathcal{S}$ if the plain value belonged to $\mathcal{C}$). If $\mathcal{C}$ is malicious he has to prove in zero-knowledge that the encryption was performed correctly and $[\![x]\!]$ indeed encrypts an ℓ-bit value (details later in §VI).

Homomorphic Value to Plain Value for Outputs: To convert a homomorphic value into a plain value for $\mathcal{C}$, $\mathcal{S}$ sends the homomorphic value to $\mathcal{C}$ who decrypts and obtains the plain value. If only $\mathcal{S}$ should learn the plain value corresponding to a homomorphic ℓ-bit value $[\![x]\!]$, $\mathcal{S}$ additively blinds the homomorphic value by choosing a random mask $r \in_R \{0,1\}^{\ell+\sigma}$, where σ is the statistical security parameter, and computing $[\![\bar{x}]\!] = [\![x]\!] \boxplus [\![r]\!]$. $\mathcal{S}$ sends this blinded value to $\mathcal{C}$ who decrypts and sends back $\bar{x}$ to $\mathcal{S}$. Finally, $\mathcal{S}$ computes $x = \bar{x} - r$. If $\mathcal{C}$ is malicious he has to prove in zero-knowledge that he correctly decrypted $\bar{x}$.

Packing can be used to improve efficiency of parallel output conversions.

B. Garbled Functions and Evaluation of OBDDs and Boolean Circuits

Efficient techniques for evaluation boolean circuits and OBDDs are quite similar; in fact the underlying idea is the same. In this section we will present the main idea and complete high-level treatment of the two protocols. We then present corresponding details for SFE of boolean circuits in §IV-C and OBDDs in §IV-D.

The idea for SFE, going back to Yao [2], is in fact to evaluate the function, step by basic step, under encryption. Yao's approach, which considered circuits, is to encrypt (or *garble*) each wire with a symmetric encryption scheme. In contrast with homomorphic encryption, described above in §IV-A, the encryptions/garblings here cannot be operated on without additional help. We will explain in detail how to operate under encryption on the basic function steps in §IV-C and §IV-D.

We now proceed with describing at the high level Yao's technique, and presenting the state of the art in the crypto primitives the method relies on. Following Yao's terminology, in this section, we talk about *garbled functions*, as the generalization of OBDD and boolean circuit.

To securely evaluate a function f, the *constructor* (server $\mathcal{S}$) creates a garbled function $\widetilde{f}$ from f. In $\widetilde{f}$, the garbled values of each wire W_i are two (random-looking) secrets $\widetilde{w}_i^0, \widetilde{w}_i^1$ that correspond to the values 0 or 1. We note that a garbled value $\widetilde{w}_i^j$ does not reveal its corresponding plain value j. $\mathcal{S}$ sends $\widetilde{f}$ to *evaluator* (client $\mathcal{C}$) and $\mathcal{C}$ additionally obtains both players' garbled input values $\widetilde{x}_1, \ldots, \widetilde{x}_u$ from $\mathcal{S}$ in an oblivious way (this requires further interaction as described later). $\mathcal{C}$ uses the garbled function and the garbled input values to obliviously compute the corresponding garbled output values $(\widetilde{z}_1, \ldots, \widetilde{z}_v) = \widetilde{f}(\widetilde{x}_1, \ldots, \widetilde{x}_u)$. We emphasize that during the step-by-step encrypted evaluation, all intermediate results are garbled values and hence do not reveal any additional information. (We remind the reader that we postpone the explanation of the encrypted evaluation techniques for circuits and OBDDs to §IV-C and §IV-D.) Finally, the garbled output values $\widetilde{z}_i$ can be translated into their corresponding plaintext values z_i.

We stress that a garbled function $\widetilde{f}$ cannot be re-used. Each secure evaluation requires construction and transfer of a new garbled function which can be done in a precomputation phase.

1) Garbled Values and Conversions: For garbled functions, conversions between the plaintext values and encryptions involve a number of subtleties and tricks. Recall, we need to convert both players' plaintext input values into their corresponding garbled values (encrypt inputs), then evaluate the garbled function (evaluate under encryption), and finally convert the garbled outputs back into plain values (decrypt result).

The interface for SFE protocols based on garbled functions are *garbled values* (see Fig. 3 in §V). A garbled boolean value $\widetilde{x}_i$ represents a bit x_i. Each garbled boolean value $\widetilde{x}_i = \langle k_i, \pi_i \rangle$ consists of a key $k_i \in \{0,1\}^t$, where t is the symmetric security parameter, and a permutation bit $\pi_i \in \{0,1\}$. The garbled value $\widetilde{x}_i$ is assigned to one of the two corresponding garbled values $\widetilde{x}_i^0 = \langle k_i^0, \pi_i^0 \rangle$ or $\widetilde{x}_i^1 = \langle k_i^1, \pi_i^1 \rangle$ with $\pi_i^1 = 1 - \pi_i^0$. The permutation bit π_i allows efficient evaluation of the garbled function using the so-called point-and-permute technique but does not reveal information about the corresponding plain value as it looks random. Of course, a garbled ℓ-bit value can be viewed as a vector of ℓ garbled boolean values.

In the following we show how to convert a plain value into its corresponding garbled value and back.

Garbled Value to Plain Value for Outputs: To convert a garbled value $\widetilde{x}_i = \langle k_i, \pi_i \rangle$ into its corresponding plain value x_i for evaluator $\mathcal{C}$, constructor $\mathcal{S}$ reveals the output permutation bit π_i^0 which was used during construction of the garbled wire and $\mathcal{C}$ obtains $x_i = \pi_i \oplus \pi_i^0$.

If the garbled value $\widetilde{x}_i$ should be converted into a plain value for constructor $\mathcal{S}$, evaluator $\mathcal{C}$ can simply send $\widetilde{x}_i$ to $\mathcal{S}$ who obtains the plain value by decrypting it. We note that malicious $\mathcal{C}$ cannot cheat in this conversion as he only knows one of the two garbled values.

Plain Value to Garbled Value for Inputs: To translate a plain value x_i held by $\mathcal{S}$ into a garbled value $\widetilde{x}_i$ for $\mathcal{C}$, $\mathcal{S}$ sends the corresponding garbled value $\widetilde{x}_i^0$ or $\widetilde{x}_i^1$ to $\mathcal{C}$ depending on the value of x_i.

To convert a plain value x_i held by $\mathcal{C}$ into a garbled value $\widetilde{x}_i$ for $\mathcal{C}$, both parties execute an oblivious transfer (OT) protocol where $\mathcal{C}$ inputs x_i, $\mathcal{S}$ inputs $\widetilde{x}_i^0$ and $\widetilde{x}_i^1$ and the output to $\mathcal{C}$ is $\widetilde{x}_i = \widetilde{x}_i^0$ if $x_i = 0$ or $\widetilde{x}_i^1$ otherwise. In the following we describe how OT can be implemented efficiently in practice.

2) Oblivious Transfer: Parallel 1-out-of-2 Oblivious Transfer (OT) of n t'-bit strings (recall, $t' = t + 1$ is the length of garbled values for symmetric security parameter t), denoted as $\mathrm{OT}_{t'}^n$, is a two-party protocol run between a chooser (client $\mathcal{C}$) and a sender (server $\mathcal{S}$) as shown in Fig. 2: For $i = 1, \ldots, n$, $\mathcal{S}$ inputs n pairs of t'-bit strings $s_i^0, s_i^1 \in \{0,1\}^{t'}$ and $\mathcal{C}$ inputs n 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 other strings $s_i^{1-b_i}$, whereas $\mathcal{S}$ learns nothing about $\mathcal{C}$'s choices b_i. As described above, OT is used to convert plain values held by $\mathcal{C}$ into corresponding garbled values, i.e., the strings have length $t' = t + 1$ bits for symmetric security parameter t.

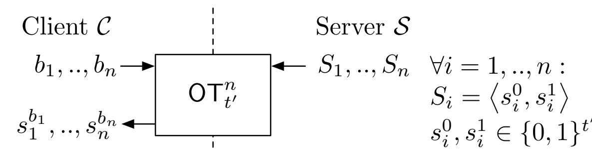


Figure 2. Parallel Oblivious Transfer

Efficient OT Protocols: $\mathrm{OT}_{t'}^n$ can be instantiated efficiently with different protocols [57]–[59]. For example the protocol of [58] implemented over a suitably chosen elliptic curve has communication complexity $n(6(2t+1)) + (2t+1) \sim 12nt$ bits and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the standard model as described in [60]. Similarly, the protocol of [57] implemented over a suitably chosen elliptic curve has communication complexity $n(2(2t+1)+2t') \sim 6nt$ bits and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the random oracle model. Both protocols require $\mathcal{O}(n)$ scalar point multiplications and two messages ($\mathcal{C} \to \mathcal{S} \to \mathcal{C}$).

Extending OT Efficiently: The extensions of [4] can be used to reduce the number of computationally expensive public-key operations of $\mathrm{OT}_{t'}^n$ to be independent of n.[5] The transformation for semi-honest $\mathcal{C}$ reduces $\mathrm{OT}_{t'}^n$ to OT_t^t and a small additional overhead: one additional message, $2n(t'+t)$ bits additional communication, and $\mathcal{O}(n)$ invocations of a correlation robust hash function such as SHA-256 ($2n$ for $\mathcal{S}$ and n for $\mathcal{C}$) which is substantially cheaper than $\mathcal{O}(n)$ asymmetric operations. Also a slightly less efficient extension for malicious $\mathcal{C}$ is given in [4].

In some computation-sensitive applications, the important technique of [4] provides a critical performance improvement by getting rid of expensive public-key operations. We strongly recommend using it for functions with large inputs, or in conjunction with OT pre-computation (see next).

Pre-Computing OT: All computationally expensive operations for OT can be shifted into a setup phase by pre-computing OT of [61]: In the setup phase the parallel OT protocol is run on randomly chosen values. Then, in the online phase, $\mathcal{C}$ uses its randomly chosen values r_i to mask his private inputs b_i, and sends them to $\mathcal{S}$. $\mathcal{S}$ replies with encryptions of his private inputs s_i^j using his random values m_i^j from the setup phase. Which input of $\mathcal{S}$ is masked with which random value is determined by $\mathcal{C}$'s message. Finally, $\mathcal{C}$ can use the masks m_i he received from the OT protocol in the setup phase to decrypt the correct output values $s_i^{b_i}$.

More precisely, the *setup phase* works as follows: for $i = 1, \ldots, n$, $\mathcal{C}$ chooses random bits $r_i \in_R \{0,1\}$ and $\mathcal{S}$ chooses random masks $m_i^0, m_i^1 \in_R \{0,1\}^{t'}$. Both parties run a $\mathrm{OT}_{t'}^n$ protocol on these randomly chosen values, where $\mathcal{S}$ inputs the pairs $\langle m_i^0, m_i^1 \rangle$ and $\mathcal{C}$ inputs r_i and obtains the masks $m_i = m_i^{r_i}$ as output. In the *online phase*, for each $i =$

[5]This is the reason for our choice of notation $\mathrm{OT}_{t'}^n$ instead of $n \times \mathsf{OT}^{t'}$.

$1, \ldots, n$, $\mathcal{C}$ masks its input bits b_i with r_i as $\bar{b}_i = b_i \oplus r_i$ and sends these masked bits to $\mathcal{S}$. $\mathcal{S}$ responds with the masked pair of t'-bit strings $\langle \bar{s}_i^0, \bar{s}_i^1 \rangle = \langle m_i^0 \oplus s_i^0, m_i^1 \oplus s_i^1 \rangle$ if $\bar{b}_i = 0$ or $\langle \bar{s}_i^0, \bar{s}_i^1 \rangle = \langle m_i^0 \oplus s_i^1, m_i^1 \oplus s_i^0 \rangle$ otherwise. $\mathcal{C}$ obtains $\langle \bar{s}_i^0, \bar{s}_i^1 \rangle$ and decrypts $s_i^{b_i} = \bar{s}_i^{r_i} \oplus m_i$. Overall, the online phase consists of two messages of size n bits and $2nt'$ bits and negligible computation (XOR of bitstrings).

3) Covert and Malicious Adversaries: It is relatively easy to protect SFE protocols based on garbled functions against covert or malicious client $\mathcal{C}$ by using an OT protocol which is secure against covert or malicious $\mathcal{C}$.

Standard SFE protocols with garbled functions which additionally protect against covert [62] or malicious [63] server $\mathcal{S}$ rely on the following cut-and-choose technique: $\mathcal{S}$ creates multiple garbled functions $\widetilde{f}_i$, deterministically derived from random seeds s_i, and commits to each, e.g., by sending $\widetilde{f}_i$ or $Hash(\widetilde{f}_i)$ to $\mathcal{C}$. In covert case, $\mathcal{C}$ asks $\mathcal{S}$ to open all but one garbled function I by revealing the corresponding $s_{i \neq I}$. For all opened functions, $\mathcal{C}$ computes $\widetilde{f}_i$ and checks that they match the commitments. The malicious case is similar, but $\mathcal{C}$ asks $\mathcal{S}$ to open half of the functions, evaluates the remaining ones and chooses the majority of their results. To ensure that $\mathcal{S}$'s inputs into OT are consistent with the garbled circuits one needs to resort to committed OT protocols as described in [64].

According to [6], the protocols of [65] and [66] which also achieve security against malicious adversaries require substantially more computation than protocols based on garbled functions and cut-and-choose as they need public-key operations rather than symmetric key operations for each gate of the circuit. Yet, a precise practical comparison between the different approaches has not been looked at so far and remains to be investigated in the future.

C. Garbled Circuits

We now turn to presenting the boolean-circuit-specific details of SFE of garbled functions as introduced in [2]. Recall, in §IV-B we left out the method of step-by-step creation of the garbled function $\widetilde{f}$ and its evaluation given the garblings of the input wires. In the following we describe how the garbled circuit is constructed and evaluated.

To construct the garbled circuit $\widetilde{C}$ for a given boolean circuit C, constructor $\mathcal{S}$ assigns to each wire W_i of the circuit two randomly chosen garbled values $\widetilde{w}_i^0, \widetilde{w}_i^1$ – encryptions of 0 and 1 on that wire. We now show how to perform a basic step – to evaluate a gate G_i under encryption. That is, given two garblings (one of each of the two of the gate's inputs), we need to obtain the garbling of the output wire consistently with the gate function. Here constructor $\mathcal{S}$ gives help to evaluator in the form of 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. This is easily done as follows. There are only four possible input combinations (and their garblings). The garbled table will consist of four entries, each of which is an encryption under a pair of input wire garblings of the corresponding output garbling. Clearly, this allows the evaluator to compute G_i under encryption, and it can be shown that $\widetilde{T}_i$ does not leak any information [67].

Table II
SIZE OF KNOWN GC TECHNIQUES IN BITS PER GARBLED 2-INPUT GATE. t: SYMMETRIC SECURITY PARAMETER

GC Technique	non-XOR gate	XOR gate
Point-and-Permute [23]	$4t + 4$	$4t + 4$
Garbled Row Reduction [27]	$3t + 3$	$3t + 3$
Secret-Sharing [6]	$2t + 4$	$2t + 4$
Free XOR [5]	$4t + 4$	0
Garbled Row Reduced Free XOR [6]	$3t + 3$	0

This method is composable, and the entire boolean circuit can be evaluated gate-by-gate in this manner. This technique also applies to gates with more than two inputs, but the size of garbled tables grows exponentially in the number of gate inputs.

The above is a simple description of Yao's technique. Today, a number of optimizations exist, which we survey next (but do not discuss in detail).

1) Efficient Garbled Circuits: A summary of several techniques for garbled circuits is shown in Table II. In the following we concentrate on the currently most efficient technique for garbled circuits, Garbled Row Reduced Free XOR of [6], which combines free XOR gates of [5] with garbled row reduction of [27]. As XOR gates occur frequently in most circuits, this technique results in better performance than the point-and-permute technique of [23] or the secret-sharing based technique of [6].

The garbled circuit technique of [6] allows "free" evaluation of *XOR gates* from [5], i.e., a garbled XOR gate has no garbled table (*no communication*) and its evaluation consists of XOR-ing its garbled input values to obtain the garbled output value (*negligible computation*).

The other gates, referred to as *non-XOR gates*, are evaluated with the garbled row reduction technique of [27], i.e., each 2-input non-XOR gate requires a garbled table of size $3t + 3$ bit, where t is the symmetric security parameter. Creating the garbled table for a 2-input non-XOR gate in the pre-computation phase requires 4 invocations of a suitably chosen cryptographic hash function such as SHA-256 in the random oracle model. Later, for evaluation of a garbled 2-input non-XOR gate, evaluator needs 1 invocation of the hash function. If the cryptographic hash function is modeled to be correlation robust (a notion which is weaker than random oracles and was introduced in [4]), the number of hash invocations is twice as high.

2) Efficient Circuit Constructions with free XOR: As XOR gates can be evaluated essentially for free, the circuits to be evaluated can be optimized such that the number of non-XOR gates is minimized. Such constructions which

Table III
EFFICIENT CIRCUIT CONSTRUCTIONS FOR ℓ-BIT VALUES (FREE XOR).

Functionality	[#non-XOR 2-input gates]	Reference
Addition	ℓ	[68]
Subtraction, Comparison	ℓ	[60]
Multiplexer	ℓ	[5]
Minimum/Maximum Value + Index of n ℓ-bit values	$2\ell(n-1)+(n+1)$	[60]
Permute n bits	$n \log n - n + 1$	[5], [69]
Select v from $u \geq v$ bits	$\frac{u+3v}{2} \log v + u - 2v + 1$	[5], [70]
Multiplication	$2\ell^2 - \ell$	[60]

are commonly used in many applications are summarized in Table III: Addition, Subtraction and Comparison have cheap circuit representations (linear in the size of the inputs). Also selecting the minimum or maximum value of n values together with its index (the function evaluated in a first-price auction [27]) has linear overhead. Permuting (without duplicates) or selecting (with duplicates) n bits grows like $\mathcal{O}(n \log n)$ and is hence feasible as well. In contrast, multiplication has a more expensive circuit representation.

3) Private Circuits: In some applications the evaluated function is known by one party only and should be kept secret from the other party. This can be achieved by securely evaluating a Universal Circuit (UC) which can be programmed to simulate any circuit C and hence entirely hides C (besides the number of inputs, number of gates and number of outputs). Efficient UC constructions to simulate circuits consisting of k 2-input gates are given in [70], [71]. Generalized UCs of [72] can simulate circuits consisting of d-input gates. Which UC construction is favorable depends on the size of the simulated functionality: Small circuits can be simulated with the UC construction of [72] with overhead $\mathcal{O}(k^2)$ gates, medium-size circuits benefit from the construction of [70] with overhead $\mathcal{O}(k \log^2 k)$ gates and for very large circuits the construction of [71] with overhead $\mathcal{O}(k \log k)$ gates is most efficient. Explicit sizes and a detailed analysis of the break-even points between these constructions are given in [72].

While universal circuits entirely hide the structure of the evaluated functionality f, it is sometimes sufficient to hide f only within a class of topologically equivalent functionalities $\mathcal{F}$, called secure evaluation of a *semi-private* function $f \in \mathcal{F}$. The circuits for many standard functionalities are topologically equivalent and differ only in the specific function tables, e.g., comparison $(<, >, =, \ldots)$ or addition/subtraction. It is possible to directly evaluate the circuit and avoid the overhead of UC for semi-private functions as GC constructions (without free XOR) hide the type of the gates from evaluator $\mathcal{C}$ [14], [18]–[20], [73].

D. Garbled OBDDs

OBDDs can be evaluated securely in a way analogous to garbled circuits, as first described in [74]. We base our presentation on the natural extension [15] of [74], which also offers a (slight) improvement. Alternative approaches [75], [76] based on homomorphic encryption have smaller communication overhead, but put more computational load on $\mathcal{S}$ (public key operations instead of symmetric operations for each decision node).

We now turn to presenting the OBDD-specific details of SFE of garbled functions. Recall, in §IV-B we left out the method of step-by-step creation of the garbled function $\widetilde{f}$ and its evaluation given the garblings of the input wires. In the following we describe how the garbled OBDD is constructed and evaluated. We note that the technique is somewhat similar to that of GC.

1) Create Garbled OBDD: In the pre-computation phase, $\mathcal{S}$ generates a garbled version $\widetilde{O}$ of the OBDD O. For this, the OBDD is first extended with dummy nodes to ensure that each evaluation path traverses the same number of variables in the same order resulting in evaluation paths of equal length. Further, OBDD nodes are randomly permuted to prevent leaking information from the sequence of steps taken by the evaluator (the start node P_1 remains the first node in $\widetilde{O}$). Then, each decision node P_i, labeled with boolean variable x_j, is converted into a garbled node $\widetilde{P}_i$ in $\widetilde{O}$, as follows. A randomly chosen key $\Delta_i \in_R \{0,1\}^t$ is associated with each node P_i. Node's information (pointers to the two successor nodes, and their encryption keys) is encrypted with the node's key Δ_i. To preserve security, we ensure that Δ_i is only revealed to the evaluator, if this node is reached by executing on the parties' inputs. Processing/evaluating an OBDD node is simply following the pointer to one of the two child nodes, depending on the input. Since we must prevent the evaluator from following both successor nodes, we additionally encrypt left (resp. right) successor information with the garbling of the 0-value (resp. 1-value) of P_i's decision variable x_j.

2) Evaluate Garbled OBDD: It is now easy to see the corresponding OBDD evaluation procedure. $\mathcal{C}$ receives the garbled OBDD $\widetilde{O}$ from $\mathcal{S}$, and evaluates it locally on the garbled values $\widetilde{x}_1, .., \widetilde{x}_n$ and obtains the garbled value $\widetilde{z}$ that corresponds to the result $z = O(x_1, \ldots, x_n)$, as follows.

$\mathcal{C}$ traverses the garbled OBDD $\widetilde{O}$ by decrypting garbled decision nodes along the evaluation path starting at $\widetilde{P}_1$. At each node $\widetilde{P}_i$, $\mathcal{C}$ takes the garbled input value $\widetilde{x}_i = \langle k_i, \pi_i \rangle$ together with the node's key Δ_i to decrypt the information needed to continue evaluation of the garbled successor node until the garbled output value $\widetilde{z}$ for the corresponding terminal node is obtained.

Implementation observations and optimizations: The employed semantically secure symmetric encryption scheme can be instantiated as $\mathsf{Enc}^s_k(m) = m \oplus H(k||s)$, where s is a unique identifier used once, and $H(k||s)$ is a pseudo-random function (PRF) evaluated on s and keyed with k, e.g., a cryptographic hash function from the SHA-2 family. Additionally the following technical improvement from [74] can be used: instead of encrypting twice (sequentially, with

Δ_i and k_i^j), the successor P_{i_j}'s data can be encrypted with $\Delta_i \oplus k_i^j$. The terminal nodes are garbled simply by including their corresponding garbled output value ($\widetilde{z}^0$ for the 0-terminal or $\widetilde{z}^1$ for the 1-terminal) into the parent's node (instead of the decryption key Δ_i).

Efficiency: To evaluate the garbled OBDD $\widetilde{O}$, the cryptographic hash function (e.g., SHA-256) is invoked once per decision node along the evaluation path.

The garbled OBDD $\widetilde{O}$ for an OBDD with d decision nodes (after extension to evaluation paths of equal length) contains d garbled nodes $\widetilde{P}_i$ consisting of two ciphertexts of size $\lceil \log d \rceil + t + 1$ bits each. The size of $\widetilde{O}$ is $2d(\lceil \log d \rceil + t + 1) \sim 2d(\log d + t)$ bits. Overall, creation of $\widetilde{O}$ requires $2d$ invocations of a cryptographic hash function.

3) Private OBDDs: The garbled OBDD reveals only a small amount of information about the evaluated OBDD to $\mathcal{C}$, namely the total number d of *decision* nodes. We note that in many cases this is acceptable. If not, this information can be hidden using appropriate padding with dummy-nodes.

V. Composition of SFE Protocols with Semi-Honest Parties

We now show how to convert encryptions of intermediate values between the different representations that are used in the three protocols we described. Done securely, this allows arbitrary compositions of the three techniques, and implies significant improvements to SFE.

We had already described the conversions between the plaintext values and encryptions. These conversions are only applicable for input encryption and output decryption. Intermediate values in the protocol must be converted without ever being decrypted.

Fig. 3 shows the types of conversions that may occur in the composed SFE protocol. Both parties have plain values as their inputs into the protocol. These plain values, denoted as x, are first encrypted by converting them into their corresponding encrypted value (garbled value, denoted as $\widetilde{x}$, or homomorphic value, denoted as $[\![x]\!]$, depending on which operations should be applied). After encryption the function is securely evaluated on the encrypted values, which may involve conversion of the encryptions between several representations. Finally, an encryption of the output is obtained. The encrypted outputs are decrypted by converting them into their corresponding plain output values. In the following we describe how to efficiently convert between the two types of encryptions.

A. Garbled Values to Homomorphic Values

A garbled ℓ-bit value $\widetilde{x}$ held by $\mathcal{C}$ (usually obtained from evaluating a garbled function) can be efficiently converted into a homomorphic value held by $\mathcal{S}$ by using additive blinding or bitwise encryption as described next.

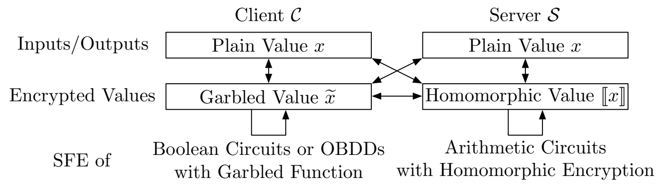


Figure 3. Composition of Secure Function Evaluation Protocols

1) Additive Blinding: $\mathcal{S}$ randomly chooses a random mask $r \in_R \{0,1\}^{\ell+\sigma}$, where σ is the statistical security parameter and $\ell + \sigma \leq |P|$ to avoid an overflow, and adds the random mask converted into garbled value $\widetilde{r}$ to $\widetilde{x}$ using a garbled $(\ell + \sigma)$-bit addition circuit that computes $\widetilde{\bar{x}}$ with $\bar{x} = x + r$. This value is converted into a plain output value $\bar{x}$ for $\mathcal{C}$ who homomorphically encrypts this value and sends the result $[\![\bar{x}]\!]$ to $\mathcal{S}$. Finally, $\mathcal{S}$ takes off the random mask under encryption as $[\![x]\!] = [\![\bar{x}]\!] \boxplus (-1)[\![r]\!]$. A detailed description of this conversion protocol is given in [60].

2) Bitwise Encryption: If the bitlength ℓ of $\widetilde{x}$ is small, a bitwise approach can be used as well in order to avoid the garbled addition circuit: $\mathcal{C}$ homomorphically encrypts the permutation bits π_i of the garbled boolean output values $\widetilde{x}_i = \langle k_i, \pi_i \rangle$ and sends $[\![\pi_i]\!]$ to $\mathcal{S}$. $\mathcal{S}$ flips those encrypted permutation bits for which the permutation bit was set as $\pi_i^0 = 1$ during creation to $[\![\pi_i']\!] = [\![1]\!] \boxplus (-1)[\![\pi_i']\!]$ or otherwise $[\![\pi_i']\!] = [\![\pi_i]\!]$. Then, $\mathcal{S}$ combines these potentially flipped bit encryptions using Horner's scheme as $[\![x]\!] = [\![\pi_\ell' || .. || \pi_1']\!]$.

Performance Comparison: The conversion based on additive blinding requires a garbled addition circuit for $(\ell + \sigma)$-bit values and the transfer of the $(\ell + \sigma)$-bit garbled value $\widetilde{r}$. When using the efficient GC technique described in §IV-C1, this requires in total $4(\ell + \sigma)(t + 1)$ bits sent from $\mathcal{S}$ to $\mathcal{C}$ in the pre-computation phase. In the online phase, the garbled circuit is evaluated and the result is homomorphically encrypted and sent to $\mathcal{S}$ (one ciphertext).

The conversion using bitwise encryption requires ℓ homomorphic encryptions and transfer of ℓ ciphertexts from $\mathcal{C}$ to $\mathcal{S}$ in the online phase. At least for converting a single bit, i.e., when $\ell = 1$, this technique results in better performance.

B. Homomorphic Values to Garbled Values

In the following we describe how to convert a homomorphic ℓ-bit value $[\![x]\!]$ into a garbled value $\widetilde{x}$. This protocol has been widely used to combine homomorphic encryption with garbled functions, e.g., in [10], [15], [77], [78].

$\mathcal{S}$ additively blinds $[\![x]\!]$ with a random pad $r \in_R \{0,1\}^{\ell+\sigma}$, where σ is the statistical security parameter and $\ell + \sigma \leq |P|$ to avoid an overflow, as $[\![\bar{x}]\!] = [\![x]\!] \boxplus [\![r]\!]$. $\mathcal{S}$ sends the blinded ciphertext $[\![\bar{x}]\!]$ to $\mathcal{C}$ who decrypts and inputs the ℓ least significant bits of $\bar{x}$, $\chi = \bar{x} \bmod 2^\ell$, to an ℓ-parallel OT protocol to obtain the corresponding garbled value $\widetilde{\chi}$. Then, the mask is taken off within a garbled ℓ-bit subtraction circuit which gets as inputs $\widetilde{\chi}$ and $\widetilde{\rho}$ converted from $\rho = r \bmod 2^\ell$ as input from $\mathcal{S}$. The output obtained by $\mathcal{C}$ is $\widetilde{x}$ which corresponds to $x = \chi - \rho$.

Again, *packing* as described in §IV-A3 can be used to improve efficiency of parallel conversions from homomorphic to garbled values by packing multiple ciphertexts together before additive blinding and sending them to $\mathcal{C}$.

VI. Efficient Techniques for Protection Against Malicious Actions

To achieve security against malicious parties, privacy-preserving protocols are usually designed in "layers". First a core protocol in the semi-honest model is constructed, and then, following the compilation paradigm of [45], each party needs to prove in zero-knowledge that it behaved honestly. (In the case of covert adversaries, each party needs to be convinced that a cheating opponent can be caught with certain probability, a weaker requirement.) As discussed in §III-A2, it is often necessary to achieve hybrid security against malicious client $\mathcal{C}$, while the server $\mathcal{S}$ is assumed to be semi-honest. In the following, we summarize standard methods for proving relations among homomorphically encrypted values in zero-knowledge and show how to avoid expensive zero-knowledge proofs for several standard tasks, such as multiplication of homomorphic values and conversion between homomorphic and garbled values.

A. Zero-Knowledge Proofs

A proof of knowledge for a relation $\mathcal{R} = \{(x, w)\}$ is a protocol between a prover and a verifier. Both parties get the public value x as common input while prover gets witness w as private input with $(x, w) \in \mathcal{R}$ and tries to convince the verifier that he knows a witness. After the protocol execution, verifier decides whether it accepts or rejects the proof. A proof must be complete and sound. Completeness guarantees that for any pair $(x, w) \in R$, the verifier accepts the proof if both prover and verifier follow the protocol. Soundness guarantees that a cheating prover cannot successfully convince a verifier if prover does not know a witness w for x. More formally, a knowledge extractor with black-box access to the prover can be constructed to compute a witness (cf., e.g., [79]). A proof is zero-knowledge, if a simulator can be constructed that, given access to x and the malicious verifier, produces a view of the protocol which is indistinguishable from verifier's view in a protocol execution with a real prover. In special honest-verifier zero-knowledge (SHVZK) proofs the verifier is assumed to be semi-honest and the simulator can produce views for a given challenge of the verifier.

Efficient SHVZK proofs of knowledge are the well-known Σ-protocols [80], [81]. These are 3-move protocols where prover starts with a commit message, verifier provides a randomly chosen challenge which is answered by the prover. Σ-protocols can be efficiently combined to prove an arbitrary AND/OR combination of underlying statements [80].

Σ-protocols can be made non-interactive using the standard Fiat-Shamir heuristic [82] of computing the challenge from the first message using a hash function. This can be proved secure in the random oracle model.

Zero-Knowledge Proofs for SFE: We summarize several efficient zero-knowledge protocols suited as building-blocks to secure SFE protocols against malicious behavior.

For the additively homomorphic Paillier and Damgård-Jurik cryptosystems one can efficiently prove knowledge of the plaintext encrypted within a ciphertext [49]. It is also possible to prove various relations about the plaintexts encrypted within a ciphertext [83], e.g., equality, linear, or multiplicative relations between two encrypted plaintexts, or that an encrypted plaintext indeed is an ℓ-bit value using efficient interval proofs of [84].

To achieve security against malicious client $\mathcal{C}$ in SFE protocols based on homomorphic encryption, it is necessary that $\mathcal{C}$'s public-key pk is well-formed. To achieve this, pk can be generated (or checked) and certified by a trusted third party. Alternatively, $\mathcal{C}$ can prove to $\mathcal{S}$ in zero-knowledge that pk – an RSA modulus in most commonly used additively homomorphic schemes of [48], [49] – is well-formed using the rather expensive zero-knowledge proof of [85].

B. Multiplication of Homomorphic Values

In the following, we discuss protocols for multiplying two homomorphic ℓ-bit values $[\![x]\!]$ and $[\![y]\!]$ with security against malicious $\mathcal{C}$. The obvious approach is to extend the semi-honest protocol of §IV-A4 which uses additively blinded values $[\![\bar{x}]\!], [\![\bar{y}]\!]$ such that $\mathcal{C}$ proves in zero-knowledge that he behaved honestly, i.e., that the multiplicative relation between $[\![\bar{x}]\!], [\![\bar{y}]\!]$, and $[\![\bar{x}\bar{y}]\!]$ holds.

Optimization: We show how to improve efficiency of this protocol by avoiding to prove the multiplicative relation in zero-knowledge: $\mathcal{S}$ chooses random multiplicative masks $m_x, m_y \in_R \{0,1\}^\sigma$ and additive masks $t_x, t_y \in_R \{0,1\}^{\ell+2\sigma}$, where σ is the statistical security parameter and $\ell + 2\sigma \leq |P|$ to avoid an overflow. Then, $\mathcal{S}$ blinds the values multiplicatively and additively by computing $[\![\bar{x}]\!] = [\![m_x x + t_x]\!]$ and $[\![\bar{y}]\!] = [\![m_y y + t_y]\!]$ and sends these blinded values to $\mathcal{C}$. $\mathcal{C}$ decrypts, multiplies and sends back $[\![c]\!] = [\![\bar{x}\bar{y}]\!]$. Finally, $\mathcal{S}$ obtains the intended result as $[\![xy]\!] = (m_x m_y)^{-1}[\![c]\!] \boxplus (-m_y t_x)[\![y]\!] \boxplus (-m_x t_y)[\![x]\!] \boxplus [\![-t_x t_y]\!]$.

It is easy to verify that if $\mathcal{C}$ cheats by sending back the encryption of a different value, then he modifies the result in an unpredictable way.

C. Garbled Values to Homomorphic Values

To convert a garbled value $\widetilde{x}$ into its corresponding homomorphic value $[\![x]\!]$ with malicious client $\mathcal{C}$ we extend the bitwise conversion protocol of §V-A2 as follows: When $\mathcal{C}$ sends the homomorphically encrypted values of the output bits to $\mathcal{S}$ he additionally has to prove in zero-knowledge that the encrypted bit is consistent with the garbled output value which is either $\widetilde{x}_i^0 = \langle k_i^0, \pi_i^0 \rangle$ or $\widetilde{x}_i^1 = \langle k_i^1, \pi_i^1 \rangle$. For this, $\mathcal{S}$ provides $\mathcal{C}$ with deterministic commitments for the two possible garblings c_i^0, c_i^1, where $c_i^{\pi_i^0} = g^{\widetilde{x}_i^0}$, $c_i^{\pi_i^1} = g^{\widetilde{x}_i^1}$, and g is the generator of a prime-order group in which the discrete logarithm problem is hard (e.g., an elliptic curve group for maximal efficiency). Using the efficient zero-knowledge proofs for knowledge of a discrete logarithm in a prime-order group of [86], $\mathcal{C}$ can efficiently prove the following statements in zero-knowledge: ($\mathcal{C}$ knows the discrete log of c_i^0 AND the homomorphic ciphertext encrypts 0) OR ($\mathcal{C}$ knows the discrete log of c_i^1 AND the homomorphic ciphertext encrypts 1).

D. Homomorphic Values to Garbled Values

Finally, we describe how to efficiently convert a homomorphic ℓ-bit value $[\![x]\!]$ into a garbled value $[\![x]\!]$ with malicious client $\mathcal{C}$. The high-level structure is the same as the conversion for semi-honest parties described in V-B: $\mathcal{S}$ blinds the homomorphic value with a randomly chosen mask $r \in_R \{0,1\}^{\ell+\sigma}$ as $[\![\bar{x}]\!] = [\![x]\!] \boxplus [\![r]\!]$ and sends this to $\mathcal{C}$. $\mathcal{C}$ decrypts and obtains the $(\ell+\sigma)$-bit representation $\bar{x}_i$. Now, $\mathcal{C}$ must be guaranteed that he decrypted correctly and the inputs in the following OT protocol match this decrypted value. For this, $\mathcal{C}$ decomposes $\bar{x}$ into its bit-representation $\bar{x}_i$ and sends homomorphic encryptions of each bit $[\![\bar{x}_i]\!]$ to $\mathcal{S}$. Additionally, $\mathcal{C}$ proves in zero-knowledge that these homomorphically encrypted bits when added together as $\sum_{i=1}^{\ell+\sigma} 2^{i-1}[\![\bar{x}_i]\!]$ encrypt the same value as $[\![\bar{x}]\!]$. This corresponds essentially to proving equality of two encrypted plaintexts as the encryption scheme is homomorphic. Additionally, $\mathcal{C}$ has to prove that each encrypted bit $[\![\bar{x}_i]\!]$ is indeed an encryption of either 0 or 1. We show how to avoid this rather expensive proof later. $\mathcal{S}$ uses $[\![\bar{x}_i]\!]$ as first message in the Paillier-based OT protocol of [59] to obliviously transfer the corresponding garbled values of $\widetilde{\bar{x}}$ to $\mathcal{C}$. Then, $\mathcal{C}$ evaluates a garbled subtraction circuit to take off the random mask. This circuit gets inputs $\widetilde{\bar{x}}$ and $\widetilde{r}$ and computes the garbled value $\widetilde{x}$ corresponding to $x = \bar{x} - r$.

Optimization: In the following we try to optimize such that $\mathcal{C}$ does not need to prove in zero-knowledge that he indeed sent homomorphic encryptions of bits. We note that if $\mathcal{C}$ tries to cheat by sending an encryption of neither 0 nor 1 he will obtain a random string instead of a valid garbled input value corresponding to this bit as output of the OT protocol. Due to this property of OT it would be sufficient if $\mathcal{C}$ proves in zero-knowledge that he obtained correctly the garbled input values $\widetilde{\bar{x}}_i$ which implies that he did not cheat with the inputs of the OT protocol (the probability that $\mathcal{C}$ guesses a valid garbled value is negligible). Instead of proving this in zero-knowledge we reduce the costs even more. For this we observe that the most significant output bit of the subtraction circuit depends on *all* input bits $\widetilde{\bar{x}}_i$. $\mathcal{C}$ can obtain one of the two valid garbled output values for this most-significant bit only if he knows all garbled input bits. We connect a garbled 1-input zero-gate to this wire which maps both possible garbled input values to the single garbled output value $\widetilde{c}^0$ (invalid garbled inputs are mapped to different values with high probability). Finally, $\mathcal{C}$ only needs to send $\widetilde{c}^0$ to $\mathcal{S}$ to prove that it behaved correctly. As the zero-gate always evaluates to the same value, no additional information is leaked to $\mathcal{S}$.

Acknowledgments

Ahmad-Reza Sadeghi and Thomas Schneider have been supported by EU FP6 project SPEED, EU FP7 project CACE and ECRYPT II.

References

[1] A. C. Yao, "Protocols for secure computations," in *IEEE Symposium on Foundations of Computer Science (FOCS'82)*. IEEE, 1982, pp. 160–164.

[2] ——, "How to generate and exchange secrets," in *IEEE Symposium on Foundations of Computer Science (FOCS'86)*. IEEE, 1986, pp. 162–167.

[3] K. B. Frikken, "Practical private DNA string searching and matching through efficient oblivious automata evaluation," in *Data and Applications Security (DBSec'09)*, ser. LNCS, vol. 5645. Springer, 2009, pp. 81–94.

[4] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology – CRYPTO'03*, ser. LNCS, vol. 2729. Springer, 2003, pp. 145–161.

[5] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *International Colloquium on Automata, Languages and Programming (ICALP'08)*, ser. LNCS, vol. 5126. Springer, 2008, pp. 486–498.

[6] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Advances in Cryptology – ASIACRYPT'09*, ser. LNCS, vol. 5912. Springer, 2009, pp. 250–267.

[7] D. Kahn, *The Codebreakers — The Story of Secret Writing*. New York, USA: Macmillan Publishing Co, 1967.

[8] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. U. Celik, "Privacy preserving error resilient DNA searching through oblivious automata," in *ACM Conference on Computer and Communications Security (CCS'07)*. ACM, 2007, pp. 519–528.

[9] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *IEEE Symposium on Security and Privacy (S&P'08)*. IEEE, 2008, pp. 216–230.

[10] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *ACM Conference on Computer and Communications Security (CCS'07)*. ACM, 2007, pp. 498–507.

[11] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *Advances in Cryptology – ASIACRYPT'05*, ser. LNCS, vol. 3788. Springer, 2005, pp. 236–252.

[12] Y. Lindell and B. Pinkas, "Privacy preserving data mining," *J. Cryptology*, vol. 15, no. 3, pp. 177–206, 2002.

[13] ——, “Secure multiparty computation for privacy-preserving data mining,” *Journal of Privacy and Confidentiality*, vol. 1, no. 1, pp. 59–98, 2009.

[14] K. B. Frikken, M. J. Atallah, and C. Zhang, “Privacy-preserving credit checking,” in *ACM conference on Electronic Commerce (EC’05)*. ACM, 2005, pp. 147–154.

[15] 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 *European Symposium on Research in Computer Security (ESORICS ’09)*, ser. LNCS, vol. 5789. Springer, 2009, pp. 424–439.

[16] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, “Privacy-preserving face recognition,” in *Privacy Enhancing Technologies Symposium (PETS’09)*, ser. LNCS, vol. 5672. Springer, 2009, pp. 235–253.

[17] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, “Efficient privacy-preserving face recognition,” in *International Conference on Information Security and Cryptology (ICISC’09)*, ser. LNCS. Springer, 2009, full version available at http://eprint.iacr.org/2009/507.

[18] K. B. Frikken, M. J. Atallah, and J. Li, “Hidden access control policies with hidden credentials,” in *ACM Workshop on Privacy in the Electronic Society (WPES’04)*. ACM, 2004, pp. 27–27.

[19] ——, “Attribute-based access control with hidden policies and hidden credentials,” *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1259–1270, 2006.

[20] K. B. Frikken, J. Li, and M. J. Atallah, “Trust negotiation with hidden credentials, hidden policies, and policy cycles,” in *Network and Distributed System Security Symposium (NDSS’06)*, 2006.

[21] R. Gennaro, C. Gentry, and B. Parno, “Non-interactive verifiable computing: Outsourcing computation to untrusted workers,” Cryptology ePrint Archive, Report 2009/547, 2009, http://eprint.iacr.org/.

[22] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *ACM Symposium on Theory of Computing (STOC’09)*. ACM, 2009, pp. 169–178.

[23] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, “Fairplay — a secure two-party computation system,” in *USENIX*, 2004, http://fairplayproject.net.

[24] A. Schröpfer, F. Kerschbaum, D. Biswas, S. Geißinger, and C. Schütz, “L1 – faster development and benchmarking of cryptographic protocols,” in *ECRYPT Workshop on Software Performance Enhancements for Encryption and Decryption and Cryptographic Compilers (SPEED-CC’09)*, 2009.

[25] T. Sander and C. Tschudin, “Protecting mobile agents against malicious hosts,” in *Mobile Agents and Security*, ser. LNCS, vol. 1419. Springer, 1998, pp. 44–60.

[26] C. Cachin, J. Camenisch, J. Kilian, and J. Müller, “One-round secure computation and secure autonomous mobile agents,” in *International Colloquium on Automata, Languages and Programming (ICALP’00)*, ser. LNCS, vol. 1853. Springer, 2000.

[27] M. Naor, B. Pinkas, and R. Sumner, “Privacy preserving auctions and mechanism design,” in *ACM Conference on Electronic Commerce*, 1999, pp. 129–139.

[28] 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)*, ser. LNCS, vol. 4586. Springer, 2007, pp. 416–430.

[29] B. Bollig and I. Wegener, “Improving the variable ordering of OBDDs is NP-complete,” *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996.

[30] V. Kabanets and J. Cai, “Circuit minimization problem,” in *ACM Symposium on Theory of Computing (STOC’00)*. ACM, 2000, pp. 73–79.

[31] H. Vollmer, *Introduction to Circuit Complexity: A Uniform Approach*. Secaucus, NJ, USA: Springer, 1999.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[33] E. Allender, M. C. Loui, and K. W. Regan, “Complexity classes,” in *Algorithms and Theory of Computation Handbook*, M. J. Atallah, Ed. CRC Press, 1999, ch. 27.

[34] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[35] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD’93)*. IEEE, 1993, pp. 42–47.

[36] M. Fujita, Y. Matsunaga, and T. Kakuda, “On variable ordering of binary decision diagrams for the application of multi-level logic synthesis,” in *Conference on European Design Automation (EURO-DAC’91)*. IEEE, 1991, pp. 50–54.

[37] R. Drechsler, B. Becker, and N. Gockel, “Genetic algorithm for variable ordering of OBDDs,” *IEE Proceedings on Computers and Digital Techniques*, vol. 143, no. 6, pp. 364–368, 1996.

[38] W. Lenders and C. Baier, “Genetic algorithms for the variable ordering problem of binary decision diagrams,” in *Foundations Of Genetic Algorithms (FOGA’05)*, ser. LNCS, vol. 3469, 2005, pp. 1–20.

[39] B. Bollig, M. Löbbing, and I. Wegener, “Simulated annealing to improve variable orderings for OBDDs,” IEEE/ACM International Workshop on Logic Synthesis (IWLS’95), 1995.

[40] R. E. Bryant, “On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication,” *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 205–213, 1991.

[41] P. Woelfel, “Bounds on the OBDD-size of integer multiplication via universal hashing,” *Journal of Computer and System Sciences*, vol. 71, no. 4, pp. 520–534, 2005.

[42] A. Karatsuba and Y. Ofman, “Multiplication of many-digital numbers by automatic computers,” *Proceedings of the SSSR Academy of Sciences*, vol. 145, pp. 293–294, 1962.

[43] A. Schönhage and V. Strassen, “Schnelle Multiplikation großer Zahlen (Fast multiplication of large numbers),” *Computing*, vol. 7, no. 3, pp. 281–292, 1971.

[44] M. Fürer, “Faster integer multiplication,” in *ACM Symposium on Theory Of Computing (STOC’07)*. ACM, 2007, pp. 57–66.

[45] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2: Basic Applications, draft available at http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html.

[46] Y. Aumann and Y. Lindell, “Security against covert adversaries: Efficient protocols for realistic adversaries,” in *Theory of Cryptography (TCC’07)*, ser. LNCS, vol. 4392. Springer, 2007, pp. 137–156.

[47] D. Giry and J.-J. Quisquater, "Cryptographic key length recommendation," March 2009, http://keylength.com.

[48] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT'99*, ser. LNCS, vol. 1592. Springer, 1999, pp. 223–238.

[49] 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)*, ser. LNCS. Springer, 2001, pp. 119–136.

[50] I. Damgård, M. Geisler, and M. Krøigaard, "Homomorphic encryption and secure comparison," *Journal of Applied Cryptology*, vol. 1, no. 1, pp. 22–31, 2008.

[51] ——, "A correction to "efficient and secure comparison for on-line auctions"," Cryptology ePrint Archive, Report 2008/321, 2008.

[52] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of Cryptography (TCC'05)*, ser. LNCS, vol. 3378. Springer, 2005, pp. 325–341.

[53] T. Sander, A. Young, and M. Yung, "Non-interactive cryptocomputing for NC^1," in *IEEE Symposium on Foundations of Computer Science (FOCS'99)*. IEEE, 1999, pp. 554–566.

[54] F. Armknecht and A.-R. Sadeghi, "A new approach for algebraically homomorphic encryption," Cryptology ePrint Archive, Report 2008/422, 2008.

[55] N. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," Cryptology ePrint Archive, Report 2009/571, 2009, http://eprint.iacr.org/.

[56] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," Cryptology ePrint Archive, Report 2009/616, 2009, http://eprint.iacr.org/.

[57] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," in *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*. Society for Industrial and Applied Mathematics, 2001, pp. 448–457.

[58] W. Aiello, Y. Ishai, and O. Reingold, "Priced oblivious transfer: How to sell digital goods," in *Advances in Cryptology – EUROCRYPT'01*, ser. LNCS, vol. 2045. Springer, 2001, pp. 119–135.

[59] H. Lipmaa, "Verifiable homomorphic oblivious transfer and private equality test," in *Advances in Cryptology – ASIACRYPT'03*, ser. LNCS, vol. 2894. Springer, 2003.

[60] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Cryptology and Network Security (CANS'09)*, ser. LNCS, vol. 5888. Springer, 2009, pp. 1–20, full version available at http://eprint.iacr.org/2009/411.

[61] D. Beaver, "Precomputing oblivious transfer," in *Advances in Cryptology – CRYPTO'95*, ser. LNCS, vol. 963. Springer, 1995, pp. 97–109.

[62] V. Goyal, P. Mohassel, and A. Smith, "Efficient two party and multi party computation against covert adversaries," in *Advances in Cryptology – EUROCRYPT'08*, ser. LNCS, vol. 4965. Springer, 2008, pp. 289–306.

[63] 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*, ser. LNCS, vol. 4515. Springer, 2007, pp. 52–78.

[64] M. S. Kiraz and B. Schoenmakers, "A protocol issue for the malicious case of Yaos garbled circuit construction," in *27th Symposium on Information Theory in the Benelux*, 2006, pp. 283–290.

[65] S. Jarecki and V. Shmatikov, "Efficient two-party secure computation on committed inputs," in *Advances in Cryptology – EUROCRYPT'07*, ser. LNCS, vol. 4515. Springer, 2007, pp. 97–114.

[66] J. B. Nielsen and C. Orlandi, "Lego for two-party secure computation," in *Theory of Cryptography (TCC'09)*, ser. LNCS, vol. 5444. Springer, 2009, pp. 368–386.

[67] Y. Lindell and B. Pinkas, "A proof of Yao's protocol for secure two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009, cryptology ePrint Archive: Report 2004/175.

[68] J. Boyar, R. Peralta, and D. Pochuev, "On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$," *Theoretical Computer Science*, vol. 235, no. 1, pp. 43–57, 2000.

[69] A. Waksman, "A permutation network," *Journal of the ACM (JACM)*, vol. 15, no. 1, pp. 159–163, 1968.

[70] V. Kolesnikov and T. Schneider, "A practical universal circuit construction and secure evaluation of private functions," in *Financial Cryptography and Data Security (FC'08)*, ser. LNCS, vol. 5143. Springer, 2008, pp. 83–97.

[71] L. G. Valiant, "Universal circuits (preliminary report)," in *ACM Symposium on Theory of Computing (STOC'76)*. ACM, 1976, pp. 196–203.

[72] A.-R. Sadeghi and T. Schneider, "Generalized universal circuits for secure evaluation of private functions with application to data classification," in *International Conference on Information Security and Cryptology (ICISC'08)*, ser. LNCS, vol. 5461. Springer, 2008, pp. 336–353.

[73] A. Paus, A.-R. Sadeghi, and T. Schneider, "Practical secure evaluation of semi-private functions," in *Applied Cryptography and Network Security (ACNS'09)*, ser. LNCS, vol. 5536. Springer, 2009, pp. 89–106, http://www.trust.rub.de/FairplaySPF.

[74] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh, "Secure function evaluation with ordered binary decision diagrams," in *ACM Conference on Computer and Communications Security (CCS'06)*. ACM Press, 2006, pp. 410–420.

[75] Y. Ishai and A. Paskin, "Evaluating branching programs on encrypted data," in *Theory of Cryptography (TCC'07)*, ser. LNCS, vol. 4392. Springer, 2007, pp. 575–594.

[76] H. Lipmaa, "Private branching programs: On communication-efficient cryptocomputing," Cryptology ePrint Archive, Report 2008/107, 2008, http://eprint.iacr.org/.

[77] J. Brickell and V. Shmatikov, "Privacy-preserving classifier learning," in *Financial Cryptography and Data Security (FC'09)*, ser. LNCS, vol. 5628. Springer, 2009, pp. 128–147.

[78] A. Jarrous and B. Pinkas, "Secure hamming distance based computation and its applications," in *Applied Cryptography and Network Security (ACNS'09)*, ser. LNCS, vol. 5536. Springer, 2009, pp. 107–124.

[79] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2001, vol. 1: Basic Tools, draft available at http://www.wisdom.weizmann.ac.il/~oded/foc-vol1.html.

[80] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology – CRYPTO'94*, ser. LNCS, vol. 839. Springer, 1994, pp. 174–187.

[81] R. Cramer, "Modular design of secure yet practical cryptographic protocols," Ph.D. dissertation, CWI and University of Amsterdam, 1997.

[82] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," in *Advances in Cryptology – CRYPTO'86*, ser. LNCS, vol. 263. Springer, 1987, pp. 186–194.

[83] M. Jurik, "Extensions to the paillier cryptosystem with applications to cryptological protocols," Ph.D. dissertation, Basic Research in Computer Science, August 2003.

[84] H. Lipmaa, "On diophantine complexity and statistical zero-knowledge arguments," in *Advances on Cryptology - ASIACRYPT'03*, ser. LNCS, vol. 2894. Springer, 2003, pp. 398–415.

[85] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number is the product of two safe primes," in *Advances in Cryptology – EUROCRYPT'99*, ser. LNCS, vol. 1592. Springer, 1999, pp. 107–122.

[86] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.