From Dust to Dawn: Practically Efficient Two-Party Secure Function Evaluation Protocols and their Modular Design

(Full Version)

Vladimir Kolesnikov[1], Ahmad-Reza Sadeghi[2], and Thomas Schneider[3]

[1] Alcatel-Lucent Bell Laboratories, Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[2] CASED, Technical University Darmstadt, Germany
ahmad.sadeghi@cased.de
[3] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
thomas.schneider@trust.rub.de

Abstract. General two-party Secure Function Evaluation (SFE) allows mutually distrusting parties to (jointly) correctly compute *any* 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 privacy-preserving credit checking, medical classification, or face recognition. Today, SFE is subject of an immense amount of research in a variety of directions, and is not easy to navigate.

In this paper, we systematize the most *practically important* work of the vast research knowledge on *general* SFE. It turns out that the most efficient SFE protocols today are obtained by combining several basic techniques, such as garbled circuits and homomorphic encryption. We limit our detailed discussion to efficient general techniques. In particular, we do not discuss the details of currently *practically inefficient* techniques, such as fully homomorphic encryption (although we elaborate on its practical relevance), nor do we cover *specialized* techniques applicable only to small classes of functions.

As an important practical contribution, we present a framework in which today's practically most efficient techniques for general SFE can be viewed as building blocks with well-defined interfaces that can be easily combined to establish a complete efficient solution. Further, our approach naturally lends itself to automated protocol generation (compilation). This is evidenced by the implementation of (parts of) our framework in the TASTY SFE compiler (introduced at ACM CCS 2010).

In sum, our work is positioned as a comprehensive guide in state-of-the-art SFE, with the additional goal of extracting, systematizing and unifying the most relevant and promising general techniques from among the mass of SFE knowledge. We hope this guide would help developers of SFE libraries and privacy-preserving protocols in selecting the most efficient SFE components available today.

Keywords: framework; protocol design; privacy-preserving protocols; homomorphic encryption; garbled functions

Table of Contents

1 Introduction

The concept of two-party *Secure Function Evaluation* (SFE) was introduced in 1982 by Yao [Yao82]. The idea is to let two mutually mistrusting parties compute an arbitrary function on their private inputs without revealing any information about their inputs beyond the output of the function. Since then, this concept has been an appealing research subject in crypto and security communities, with many 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 SFE, its real-world deployment was believed to be very limited and expensive for a relatively long time. Fortunately, the cost of SFE has been dramatically reduced in the recent years thanks to many algorithmic improvements and automatic tools, as well as faster computing platforms and communication networks. (We note that SFE is only a part of a general task of *secure computing* as discussed in Appendix §A.)

Scope of this paper In this paper we survey and systematize the current state of the art of *practically efficient general* secure two-party computation. In particular, we do not discuss in detail currently practically inefficient techniques, such as fully homomorphic encryption (we elaborate on its practicality in §4.1), nor do we cover specialized techniques applicable only to small classes of functions. Moreover, we present a framework which allows to modularly combine the required techniques with well-defined interfaces to obtain highly efficient protocols suitable for practical applications. We build our presentation in the style of a tutorial, and aim for the paper to be both a reference on practically efficient SFE for experts, and an understandable area guide for non-experts in secure computation.

Efficient SFE techniques For several years, two different approaches for secure two-party computation have co-existed. One approach is based on *homomorphic encryption* (HE). Here one party sends its encrypted inputs to the other party, who then computes the intended function under encryption using the homomorphic properties of the cryptosystem, and sends back the encrypted result. Popular examples are the additively homomorphic cryptosystems of Paillier [Pai99] and Damgård-Jurik [DJ01], and the recent fully homomorphic schemes [Gen09, vDGHV10, SV10]. (We elaborate on the practicality of fully homomorphic schemes in §4.1). Alternatively, SFE can be done using *garbled functions* (GF), a generalization of Yao's garbled circuits (GC) [Yao86] that works as follows: one party (constructor) "encrypts" the function (using symmetric keys), the other party (evaluator) obliviously obtains the keys corresponding to both parties' inputs and the garbled function, and is able to decrypt the corresponding output value.

Both approaches have their respective advantages and disadvantages, i.e., GF requires to transfer the garbled function (communication complexity is at least linear in the size of the function) but allows to pre-compute almost all expensive operations resulting in a low latency of the online phase, whereas most HE schemes require relatively expensive public-key operations in the online phase but can result in a smaller overall communication complexity.

For a particular primitive, one of the techniques is usually more suitable than the other. For example, for comparison or maximum selection, GF [NPS99,KSS09] is better than HE [DGK07,DGK08b,DGK08a], whereas multiplication often benefits from using HE. Therefore, simply switching from one approach for secure computation to the other can result in substantial performance improvements. For instance, for privacy-preserving DNA matching based on secure evaluation of finite automatons, GC-based [Fri09] is more efficient than HE-based [TPKC07].

We also note that GCs are gaining popularity as a versatile tool whose applicability goes beyond the "simple" SFE [GGP10, KM10b, SS10]. Further, most published work (e.g., [GKR08, SS10]) do not give concrete GC instantiations, or constructions for how to hide the topology of the evaluated function (for which efficient universal circuit constructions are needed). We summarize today's most efficient constructions for garbled circuits and universal circuits in §4.3 of our paper.

Composition and Performance Evaluation of Efficient SFE Techniques Going one step further, we would like to benefit from the use the most efficient primitive for the respective sub-task even if they are based on different paradigms. Indeed, secure and efficient composition of sub-protocols based on HE and GF can result in performance improvements as shown for several privacy-preserving applications (see, e.g., [BPSW07, BS09, BFK$^+$09, SSW09]).

One of the goals of this work is a design and presentation of a unifying framework (§5), which allows for the above compositions in a modular way. The need for and the usefulness of this framework is illustrated by its recent (partial) implementation, "Tool for Automating Secure Two-partY computations" (TASTY) [HKS$^+$10]. TASTY allows a programmer to provide a high-level description of the computation to be performed on encrypted data in a domain-specific language, and automatically transforms this description into efficient protocols based on HE, GC, and arbitrary combinations of both. TASTY allows to compare the performance of different protocols with each other. For example, it may help determine (and quantify!) that GC-based multiplication has a more efficient online phase than HE-based multiplication for $\ell \leq 16$ bit values [HKS$^+$10].

Applications of SFE There is a large body of literature on SFE applications, in particular those with strong privacy requirements such as Privacy-Preserving Genomic Computation [TPKC07, JKS08, Fri09, KM10b], Remote Diagnostics [BPSW07], Graph Algorithms [BS05], Data Mining [LP02, LP09b], Credit Checking [FAZ05], Medical Diagnostics [BFK$^+$09], Face Recognition [EFG$^+$09,SSW09], or Policy Checking [FAL04, FAL06, FLA06], just to name a few. These applications are based on either HE or GF or a combination of both as explained before. Recently, verifiable outsourcing of computations for cloud-computing applications has been proposed, based on evaluating GCs under fully HE [GGP10]. Existence of a variety of SFE compilers, coming from both academia, e.g., [MOR03, MNPS04, HKS$^+$10], and industry [SKB$^+$09, SKM10], 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 [ST98] and GC [CCKM00], as well as privacy-preserving auction systems based on GC [NPS99] or HE [DGK07]. However, in this work, we do not address the issues of multi-party computation with more than two players. We mention, however, that this is a vibrant field with many efficient solutions [CDN01, BDNP08, BLW08, BCD$^+$09,IPS09,DO10,JMN10,DK10]. We note that in-depth conceptual and, where possible, performance comparison of two- and multi-party computation is an open problem.

Outline of the presentation We start our discussion in §2 with a few of most popular function representations and point 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 the construction of protocols. Then, in §3, we briefly discuss various notions of security and their relationship. In §4, we describe today's practically efficient SFE constructions for each of the function representations we consider. We handle the actual details of the composition, namely the techniques to convert encrypted intermediate values between the protocols in §5 for semi-honest players, a model which suits most client-server applications. We summarize the main techniques for achieving security in the malicious player setting in Appendix §D.

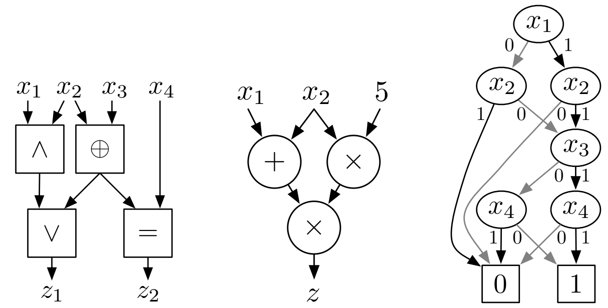
2 Function Representations

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 [BW96, KC00].

The following standard representations for functions are particularly useful for SFE: boolean circuits (cf. Fig. 1(a)), arithmetic circuits (cf. Fig. 1(b)) and ordered binary decision diagrams (OBDD) (cf. Fig. 1(c)).

In Appendix §B, we give their detailed descriptions and provide guidelines regarding efficiency choices. Here we stress that the cost of implementing SFE protocols varies greatly among the function representations. For example, the GC technique for SFE of boolean circuits is much more efficient than techniques for evaluating arithmetic circuits (e.g., using HE). However, some functions are represented much more compactly as an arithmetic circuit. As another example, some functions (e.g., decision strategies) are most compactly represented as OBDDs, while others (e.g., multiplication), require exponentially large OBDDs.

In this work (specifically, §4 and §5), we explain and advocate a hybrid approach, where function blocks can be evaluated using different techniques, and their encrypted intermediate results then glued together.



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

Fig. 1. Function Representations

3 SFE: Security Notions, Parameters, Notation

3.1 Security Notions

In this section, we give the intuition of the security notions we use. Due to the lack of space, we do not include the standard definitions here. However, we present the intuitive discussion of definitional approaches in Appendix §C and refer the reader to standard sources for formal definitions and further discussion, e.g., [Gol04, LP09b]. The definitions model *semi-honest*, *covert* and *malicious* behavior.

The strongest and most general (and, perhaps, the most natural) 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. Not surprisingly, protection against such attacks is relatively expensive, as we discuss later in §4.2 and Appendix §D. 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, e.g., as summarized later in §4.2.

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 the 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 designing protocols secure against covert and malicious adversaries (cf. §4.2).

Hybrid Security. It is often the case that players 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 cannot be made on bank clients, who may be much more willing to take the risks of 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 constructor or evaluator of garbled circuits) is almost free to secure against malicious evaluator, and much more expensive to secure against malicious constructor (details later in §4.2). Thus, GC-based protocols are good candidates for settings with corresponding trust relationships.

3.2 Computation under encryption

Before presenting the protocols in the next section, we find it instructive to present the following simple insight: each of the SFE techniques we consider can be viewed as *evaluation under encryption* with hints.

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 [Gen09]. We stress that this (and similar) schemes are currently prohibitively expensive, and are not likely to be considered for practice in the foreseeable future (see §4.1 for more discussion). In comparison, we propose extremely efficient solutions to a much simpler problem, where the computed function is fixed. Now, for example, the first player can send his encrypted input and additional function-specific "hints" to assist the second player with evaluation under encryption. This assistance can also be interactive. We further simplify our work by considering only elementary operations, e.g., boolean gates, and show how to compose their evaluation in a secure way.

3.3 Parameters and Notation

We denote the *symmetric security parameter* by t and the *asymmetric security parameter*, e.g., bitlength of RSA moduli, by T. From 2011 on, NIST recommends at least $t = 112$ and $T = 2048$. For detailed recommendations on the choice of security parameters we refer to [GQ09]. The *statistical security parameter* is denoted by σ and can be set to $\sigma = 112$. The *bitlength* of variable x is written $|x|$.

In the following, we 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 the 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.

4 SFE of Circuits and OBDDs in the Semi-honest Model

To reduce complexity, functions can be decomposed into several sub-functions (blocks). Each of these blocks can be represented in its own 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 OBDD.

In this section, we present the SFE protocols for the three representations of interest with semi-honest adversaries. We explain how to prevent/detect deviations from the protocol in Appendix §D.

It is our goal to be able to arbitrarily compose the three protocols. This, in particular, 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 (done in this section). (For completeness, we also discuss here the simple issue of how to appropriately encrypt the inputs and decrypt the outputs.) We will discuss the issues of composition of the protocols, such as conversions of encryptions, in §5. Overall, the protocol structure will look as follows: (i) encrypt the plaintext inputs, (ii) perform the computation under encryption (which may include a composition of encrypted computations), and, (iii) decrypt the output values.

4.1 Homomorphic Encryption: SFE 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 $\mathsf{Enc}(m, r)$.

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: $\forall a \in \mathbb{N}, x \in P : a[\![x]\!] = [\![ax]\!]$.

Popular instantiations for additively homomorphic encryption schemes are summarized in Table 1: The Paillier cryptosystem [Pai99] 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 [DJ01] is a generalization of the Paillier cryptosystem which provides a large plaintext space of size sT-bit for arbitrary $s \geq 1$. The cryptosystems of Damgård-Geisler-Krøigaard (DGK) [DGK07, DGK08b, DGK08a] and lifted EC-ElGamal [Gam85] (implemented over an elliptic curve group G with prime order p) have smaller ciphertexts, but are restricted to a small plaintext space $\mathbb{Z}_u$ (respectively a small subset of the plaintext space $\mathbb{Z}_p$) as decryption requires to solve a discrete log.

Table 1. Additively Homomorphic Encryption Schemes (N: RSA modulus, $s \geq 1$, u: small prime, p: large prime)

Scheme	P	C	$\mathsf{Enc}(m, r)$
Paillier [Pai99]	$\mathbb{Z}_N$	$\mathbb{Z}^*_{N^2}$	$g^m r^N \bmod N^2$
Damgård-Jurik [DJ01]	$\mathbb{Z}_{N^s}$	$\mathbb{Z}^*_{N^{s+1}}$	$g^m r^{N^s} \bmod N^{s+1}$
DGK [DGK07, DGK08b, DGK08a]	$\mathbb{Z}_u$	$\mathbb{Z}^*_N$	$g^m h^r \bmod N$
Lifted EC-ElGamal [Gam85]	$\mathbb{Z}_p$	G^2	$(g^r, g^m h^r)$

Fully Homomorphic Cryptosystems For completeness, we mention that some cryptosystems allow both 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* homomorphic.

Until recently, it was widely believed that such cryptosystems do not exist. Several works provided partial solutions: [BGN05, GHV10] allow for an arbitrary number of additions and one multiplication, and ciphertexts of [SYY99, AS08] grow exponentially in the number of multiplications. While one-multiplication schemes are relatively efficient, their use is limited due to their inherent restriction. Recent schemes [Gen09, vDGHV10, SV10] are fully homomorphic. However, the size of ciphertexts and computational cost of elementary steps in fully homomorphic schemes are *dramatically* larger than those of additively homomorphic schemes.

Recently, the first working implementation of fully homomorphic encryption was presented [GH10]. Its performance for reasonable security parameters is in the order of Gigabytes of communication and minutes of computation on high-end IBM System x3500 servers. Other recent implementation results of [SV10] show that even for very small parameters where the multiplicative depth of the evaluated circuit is $d = 2.5$, i.e., at most two multiplications, encrypting a single bit takes 3.7 s on 2.4GHz Intel Core2 (6600) CPU.

Although significant effort is underway in the theoretical community to improve its performance, it seems unlikely that fully homomorphic encryption would reach the efficiency of current public-key encryption schemes. Intuitively, this is because a fully homomorphic cryptosystem must provide the same strong security guarantees, while, at the same time, possessing extra algebraic structure to allow for homomorphic operations. The extra structure weakens security, and countermeasures (costing performance) are necessary. In this work, we do not rely on, but could use, (expensive) fully homomorphic schemes.

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 a 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) as described later in §4.1. Today, the interactive approach results in much faster SFE protocols than using fully homomorphic schemes. (The latter, however, allows non-interactive evaluation of private functions by $\mathcal{S}$; this can be done efficiently without fully HE, but with interaction, using universal circuits as shown in §4.3.)

Packing Often the plaintext space P of the homomorphic encryption scheme is substantially larger than the encrypted values. This allows for optimization of many HE-based protocols by packing together multiple ciphertexts into one before or after additive blinding and sending the single ciphertext from $\mathcal{S}$ to $\mathcal{C}$ instead. This substantially decreases the message size and 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 $[\![X]\!] = [\![x_n|| \dots ||x_1]\!]$ costs less than one full-range modular exponentiation by using Horner's scheme: $[\![X]\!] = [\![x_n]\!];$ for $i = n-1..1 : [\![X]\!] = 2^{|x_{i+1}|}[\![X]\!] \boxplus [\![x_i]\!]$.

Homomorphic Values and Conversions 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., homomorphic encryptions *held by* $\mathcal{S}$ encrypted under the public key of $\mathcal{C}$ (see Fig. 3 in §5). 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, i.e., $|x| = \ell$, into a homomorphic value $[\![x]\!]$, x is simply encrypted under $\mathcal{C}$'s public key. If x belongs to $\mathcal{C}$, $[\![x]\!]$ is sent to $\mathcal{S}$. 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 §D.1).

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.

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 $[\![xy]\!] = [\![z]\!] \boxplus (-r_x)[\![y]\!] \boxplus (-r_y)[\![x]\!] \boxplus [\![-r_x r_y]\!]$. Efficiency of parallel multiplications can be improved by *packing* multiple blinded ciphertexts together instead of sending them to $\mathcal{C}$ separately. Interactive multiplication with malicious players is given in Appendix §D.2.

4.2 Garbled Functions: SFE of Boolean Circuits and OBDDs

Efficient techniques for evaluating 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 a complete high-level treatment of the two protocols. We then present the corresponding details for SFE of boolean circuits in §4.3 and OBDDs in §4.4.

The idea for SFE, going back to Yao [Yao86], is to evaluate the function, step by basic step, under encryption. Yao's approach, which considered boolean circuits, is to encrypt (or *garble*) each wire with a symmetric encryption scheme. In contrast to homomorphic encryption (cf. §4.1), 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 §4.3, §4.4.

We now proceed to describe at the high level Yao's technique, and present the state of the art in the crypto primitives the method relies on. Following Yao's terminology, we talk about *garbled functions*, as the generalization of garbled (boolean) circuits and garbled OBDDs.

To securely evaluate a function f, the *constructor* (server $\mathcal{S}$) creates a garbled function $\widetilde{f}$ from f (a detailed description on how this is done is given later in §4.3 for boolean circuits and §4.4 for OBDDs). 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 in §4.2). $\mathcal{C}$ 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 give details on evaluating $\widetilde{f}$ later in §4.3 for boolean circuits and §4.4 for OBDDs.) Finally, the garbled output values $\widetilde{z}_j$ are translated into their corresponding plaintext values z_j (cf. §4.2).

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 pre-computation phase.

Garbled Values and Conversions For garbled functions, conversions between plaintext values and encryptions involve a number of subtleties and tricks. Recall, we first convert both players' plain inputs 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 §5). 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 [MNPS04]. Of course, a garbled ℓ-bit value can be viewed as a vector of ℓ garbled boolean values.

We show how to convert a plain value into its corresponding garbled value and back next.

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 $\mathcal{C}$ simply sends $\widetilde{x}_i$ (or π_i) to $\mathcal{S}$ who obtains the plain value by decrypting it, e.g., compare with $\widetilde{x}_i^0$ and $\widetilde{x}_i^1$. We note that malicious $\mathcal{C}$ cannot cheat in this conversion as he only knows either $\widetilde{x}_i^0$ or $\widetilde{x}_i^1$, but is unlikely to guess the other one.

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.

Oblivious Transfer Parallel 1-out-of-2 Oblivious Transfer (OT) of n t'-bit strings (where $t' = t + 1$ 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 pairs of t'-bit strings $s_i^0, s_i^1 \in \{0,1\}^{t'}$ and $\mathcal{C}$ inputs 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 of $\mathcal{C}$ into corresponding garbled values.

Efficient OT Protocols $\mathrm{OT}_{t'}^n$ can be instantiated efficiently with different protocols [NP01,AIR01,Lip03b]. For example the protocol of [AIR01] implemented over a suitably chosen elliptic curve using point compression 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 [KSS09]. Similarly, the protocol of [NP01] 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 ($\mathcal{C} \to \mathcal{S} \to \mathcal{C}$).

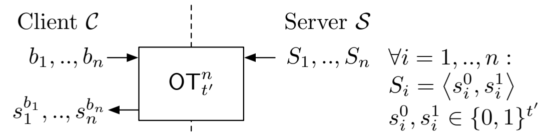


Fig. 2. Parallel Oblivious Transfer

Extending OT Efficiently The extensions of [IKNP03] can be used to reduce the number of computationally expensive public-key operations of $\mathrm{OT}^n_{t'}$ to be independent of n.[4] The transformation for semi-honest $\mathcal{C}$ reduces $\mathrm{OT}^n_{t'}$ 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. An OT extension for malicious $\mathcal{C}$ is given in [IKNP03] and improved in [Nie07].

In some computation-sensitive applications, the technique of [IKNP03] provides a critical performance improvement by getting rid of expensive public-key operations. We strongly recommend using it for functions with many/large inputs, possibly in conjunction with the following pre-computations.

Pre-Computing OT All computationally expensive operations for OT can be shifted into a setup phase by pre-computing OT [Bea95]: 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 $\mathrm{OT}^n_{t'}$ 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 $i = 1, \ldots, n$, $\mathcal{C}$ 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).

Covert and Malicious Adversaries SFE protocols based on garbled functions can be easily protected against covert or malicious client $\mathcal{C}$, by using an OT protocol with corresponding security.

Standard SFE protocols with garbled functions which additionally protect against covert [AL07, GMS08] or malicious [LP07] 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 the 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. Additionally, it must be guaranteed that $\mathcal{S}$'s input into OT is consistent with the GCs as pointed out in [KS06], e.g., using committed, committing, or cut-and-choose OT [LP10]. The practical performance of cut-and-choose-based GC protocols was investigated experimentally in [LPS08, PSSW09].[5]

For completeness, note that cut-and-choose may be avoided with SFE schemes, e.g., [JS07], which use zero-knowledge proofs of correctness of circuit construction, and operate on committed inputs [GMY04]. However, their elementary steps involve public-key operations. As estimated by [PSSW09], malicious-secure protocols [JS07, NO09] often require substantially more computation than garbled functions/cut-and-choose-based protocols.

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

[5] Secure evaluation of the AES functionality (boolean circuit with 33,880 gates) between two Intel Core 2 Duos running at 3.0 GHz, with 4 GB of RAM connected by gigabit ethernet takes approximately 0.5 MB data transfer and 7 s for semi-honest, 8.7 MB / 1 min for covert, and 400 MB / 19 min for malicious adversaries [PSSW09].

We further note that there are yet other approaches to malicious security, e.g., [IPS08] that compiles a secure multi-party computation protocol into a two-party SFE protocol. Their precise performance comparison is a desired but complicated undertaking, since, firstly, there are several performance measures, and, further, some schemes may work well only for certain classes of functions.

4.3 Garbled Circuits: SFE of Boolean Circuits

We now turn to presenting the boolean-circuit-specific details for SFE of garbled functions as introduced in [Yao86] and excellently presented in [LP09a]. Recall, in §4.2 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 random-looking 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 for each of the two inputs of the gate), we need to obtain the garbling of the output wire consistently with the gate function. Here the constructor $\mathcal{S}$ gives help to the evaluator $\mathcal{C}$ 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 [LP09a].

This method is composable s.t. the entire boolean circuit can be evaluated gate-by-gate. 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).

Table 2. Size of efficient GC techniques per garbled d-input gate (t: symmetric security parameter)

GC Technique	Size of garbled tables [bits]	Free XOR [KS08a]
Point-and-Permute [MNPS04]	$2^d t + 2^d$	yes
Garbled Row Reduction [NPS99]	$(2^d - 1)t + (2^d - 1)$	yes
Secret-Sharing [PSSW09]	$(2^d - 2)t + 2^d$	no

Efficient Garbled Circuits A summary of several techniques for garbled circuits is shown in Table 2. In the following we concentrate on the currently *most* efficient technique for garbled circuits, Garbled Row Reduced Free XOR (GRRFX) of [PSSW09], which combines free XOR gates of [KS08a] with garbled row reduction of [NPS99]. This technique requires less communication than the secret-sharing based technique of [PSSW09] as soon as more than 33% of the circuit's gates are XOR gates. This is achieved in almost all cases when applying the optimization techniques of [PSSW09] (see below). However, it can be proven secure only under a slightly stronger assumption than the standard model.

The GRRFX technique of [PSSW09] allows "free" evaluation of *XOR gates* from [KS08a], 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 [NPS99], i.e., each d-input non-XOR gate requires a garbled table of size $(2^d - 1)t + (2^d - 1)$ bit, where t is the symmetric security parameter. Creating this garbled table in the pre-computation phase requires 2^d invocations of a suitably chosen cryptographic hash function such as SHA-256 in the random oracle model. Later, for evaluation of a garbled d-input non-XOR gate, the evaluator needs 1 invocation of the hash function. If the cryptographic hash function is modeled to be correlation robust (CoR), a notion which is weaker than random oracles and was introduced in [IKNP03], the number of hash invocations is twice as high. Indeed, all known efficient GC constructions listed in Table 2 require exactly this number of hash invocations.

Circuit Optimizations As the costs of GC constructions for creating and transferring garbled tables grow exponentially in d, it is beneficial to optimize the circuit such that gates have small degree d while exploiting free XOR gates as much as possible. [LPS08] propose to encode circuit components with d inputs consisting of multiple 2-input gates by a single d-input gate. Afterwards, when XOR gates are "free", these d-input gates are decomposed into 2-input gates while minimizing the number of non-XOR gates [PSSW09].

Hardware-based SFE We note that the transfer of garbled tables can be avoided entirely when server $\mathcal{S}$ can send to client $\mathcal{C}$ a tamper-proof hardware token that generates the garbled circuit on behalf of $\mathcal{S}$. The token needs to compute only symmetric key primitives, processes the gates one-by-one using a constant amount of memory and does not need to be trusted by $\mathcal{C}$ [JKSS10b]. Another direction for improving SFE protocols is to use a cryptographic coprocessor for costly operations [IS10]. Using trusted hardware also allows to implement OT non-interactively, called *one-time programs* in combination with GC [GT08, GKR08, JKSS10c].

Pre-Computation vs. Streaming We note that most GC-based SFE implementations (e.g., [MNPS04, LPS08, PSSW09, HKS+10]) follow the compilation paradigm, in which the circuit is first compiled from a high-level description and then optimized for size (see above). Although this approach requires storage linear in the size of the circuit, it is beneficial when the function is fixed and the compilation (and possibly GC creation) can be done in a pre-computation phase. When pre-computation is not feasible (e.g., in scenarios where parties make ad-hoc decisions when and what to compute securely), it is also possible to generate the circuit and its garbling with constant storage/memory: Firstly, the circuit can be compiled on-the-fly using a constant amount of memory as implemented in [HKS+10] (see discussion in full version of [HKS+10]). Further, this stream of gates can be directly combined with the constant-memory GC creation technique of [JKSS10b], and the garbled tables can be streamed directly over the network to the evaluator who evaluates them on-the-fly [JKSS10c]. Finally, OT can be extended on-the-fly as mentioned in [IKNP03], s.t. only a constant (in the security parameter) number of public key operations is needed for an arbitrary (and unknown in advance) number of OTs. We note, however, that some circuits cannot be streamed as their evaluation requires memory linear in the circuit size [JKSS10a]. The recently proposed VMCrypt library [MK10] specifically aims to maximize GC streaming. The techniques described above as well as the "use cheapest SFE block" approach advocated in our work can be also used with their architecture, resulting in corresponding performance improvements.

Efficient Circuit Constructions with free XOR As XOR gates can be evaluated "for free", the circuits to be evaluated can be optimized so that the number of non-XOR gates is minimized as described above. These tricks can improve many basic functions, some of which are summarized in Table 3. For example, 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 [NPS99]) 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 relatively expensive circuit representation. Fast multiplication [KO62] with complexity $\mathcal{O}(\ell^{1.6})$ is more efficient than $\mathcal{O}(\ell^2)$ textbook multiplication for $\ell \geq 20$ [HKS+10].

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 [Val76, KS08b]. Generalized UCs of [SS08] 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 [SS08] with overhead $\mathcal{O}(k^2)$ gates, medium-size circuits benefit from the construction of [KS08b] with overhead $\mathcal{O}(k \log^2 k)$ gates and for very large circuits the construction of [Val76] with overhead $\mathcal{O}(k \log k)$ gates is most efficient. Explicit sizes and a

Table 3. Efficient circuit constructions for ℓ-bit values (optimized for free XOR)

Functionality	#non-XOR 2-input gates
Addition [BPP00]	ℓ
Subtraction, Comparison [KSS09]	ℓ
Multiplexer [KS08a]	ℓ
Minimum/Maximum Value + Index of n ℓ-bit values [KSS09]	$2\ell(n-1)+(n+1)$
Permute n bits [Wak68, KS08a]	$n \log n - n + 1$
Select v from $u \geq v$ bits [KS08b, KS08a]	$\frac{u+3v}{2} \log v + u - 2v + 1$
Textbook Multiplication [KSS09]	$2\ell^2 - \ell$
Fast Multiplication [HKS$^+$10]	$9\ell^{1.6} - 13\ell - 34$

detailed analysis of the break-even points between these constructions are given in [SS08]. The recent proposal of [KM10a] avoids the super-linear complexity of UCs, but requires much more expensive public-key operations.

While UCs entirely hide the structure of the evaluated functionality f, it is sometimes sufficient to hide f only within a class of topologically equivalent functionalities $\mathcal{F}$; this is 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 of [MNPS04] and [NPS99] completely hide the type of the gates from evaluator $\mathcal{C}$ [FAL04, FAZ05, FLA06, FAL06, PSS09].

4.4 Garbled OBDDs: SFE of OBDDs

OBDDs can be evaluated securely in a way analogous to garbled circuits, as first described in [KJGB06]. We base our presentation on the natural extension of [KJGB06] described in [Sch08, §3.4.1] and [BFK$^+$09], which also offers a (slight) improvement. Alternative approaches [IP07, Lip08] 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 for SFE of garbled functions. Recall, in §4.2 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 GCs described in §4.3.

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.

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 $\mathsf{Enc}_k^s(m) = m \oplus H(k||s)$, 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 [KJGB06] 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.

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 by appropriate padding with dummy-nodes.

5 Composition of SFE Blocks

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 entirely.

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.

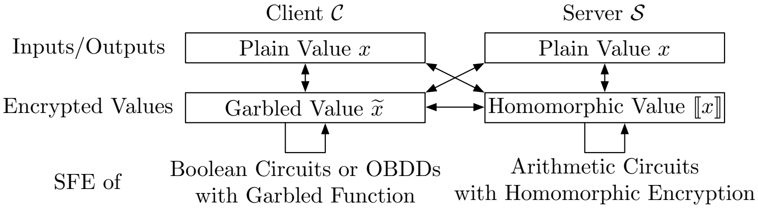

Client $\mathcal{C}$
Server $\mathcal{S}$
Inputs/Outputs
Plain Value x
Plain Value x
Encrypted Values
Garbled Value $\widetilde{x}$
Homomorphic Value $[\![x]\!]$
SFE of
Boolean Circuits or OBDDs with Garbled Function
Arithmetic Circuits with Homomorphic Encryption

Fig. 3. Composition of Secure Function Evaluation Protocols

5.1 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.

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 [KSS09].

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 §4.3, this requires in total $4(\ell+\sigma)(t+1)$ bits sent from $\mathcal{S}$ to $\mathcal{C}$ in the precomputation 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.

5.2 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 [BPSW07, BS09, JP09, BFK$^+$09].

$\mathcal{S}$ 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 §4.1 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 $\mathcal{C}$.

6 Conclusion

We conclude our survey with a summary of past, present, and possible future directions in practically efficient SFE.

Where we've come from Although the theoretical foundations of SFE have been laid over two decades ago, until recently, SFE was seen merely as a theoretical concept. Around ten years ago first SFE implementations were reported, and new primitives, such as efficient additively homomorphic encryption, have been proposed. About five years ago, coinciding with the availability of general SFE tools, a variety of privacy-preserving protocols started appearing in the research area of security, and real-life applications became within reach. In 2008 came a first major deployment of secure computation – its use in executing a nation-wide beets auction in Denmark [BCD$^+$09].

Where we are Today, we are on the verge of SFE gaining widespread recognition and use. Even now, the efficiency of existing protocols allows for business justification of its use in a number of scenarios. At the same time, both theoretical and applied research in SFE are experiencing a great surge in anticipation of its success.

A variety of SFE techniques and their prototype implementations already exist, each with its advantages and disadvantages – in this survey we have summarized today's most efficient approaches for generic SFE and presented a unified framework in which these can be arbitrarily combined.

Where we may be going With the growth of the web and social networking, came the realization of the value of privacy. Governments are introducing far-reaching restrictions on data collection and use, especially in the personal health domain. SFE is a clear candidate to help achieve privacy, while enabling a variety of applications. (Of course, no single technology, not even powerful primitives such as fully homomorphic encryption, can be used as a universal solution for private computing. This is due to both impossibility results [vDJ10] and the cost barriers raised by some SFE techniques. Instead, a comprehensive approach would include SFE, secure hardware, hardened code, legal agreements, etc.) With the political and business need in place, the Moore's-law performance improvements of hardware and expected algorithmic improvements, it is clear that SFE's use will be practically justified in more and more of security- and privacy-critical applications. In the longer term, fully homomorphic encryption may become practically efficient, and enable new opportunities.

We hope that our work serves to promote secure computation beyond theoretical research communities, and helps facilitate its earlier and broader practical use.

Acknowledgements The work described in this paper was supported in part by the European Commission through the ICT program under contract 216676 ECRYPT II and 216499 CACE. The authors would like to thank Juan Garay for helpful comments on the paper.

References

AIR01. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.

AL07. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC'07*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.

ALR99. E. Allender, M. C. Loui, and K. W. Regan. Complexity classes. In M. J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 27. CRC Press, 1999.

AS08. F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. Cryptology ePrint Archive, Report 2008/422, 2008.

BCD$^+$09. P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *FC'09*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

BDNP08. A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS'08*, pages 257–266. ACM, 2008.

Bea95. D. Beaver. Precomputing oblivious transfer. In *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

BFK$^+$09. 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 *ESORICS'09*, volume 5789 of *LNCS*, pages 424–439. Springer, 2009.

BGN05. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC'05*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005.

BLW95. B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. IWLS'95, 1995.

BLW08. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS'08*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

BPP00. J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235(1):43–57, 2000.

BPSW07. J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *CCS'07*, pages 498–507. ACM, 2007.

Bry86. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

Bry91. R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

BS05. J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT'05*, volume 3788 of *LNCS*, pages 236–252. Springer, 2005.

BS09. J. Brickell and V. Shmatikov. Privacy-preserving classifier learning. In *FC'09*, volume 5628 of *LNCS*, pages 128–147. Springer, 2009.

BW96. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

CCKM00. C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *ICALP'00*, volume 1853 of *LNCS*. Springer, 2000.

CDN01. R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT'01*, volume 2045 of *LNCS*, pages 280–299. Springer, 2001.

CDS94. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

CLRS01. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

CM99. J. Camenisch and M. Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 107–122. Springer, 1999.

Cra97. R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1997.

DBG96. R. Drechsler, B. Becker, and N. Gockel. Genetic algorithm for variable ordering of OBDDs. *IEE Proceedings on Computers and Digital Techniques*, 143(6):364–368, 1996.

DGK07. I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP'07*, volume 4586 of *LNCS*, pages 416–430. Springer, 2007.

DGK08a. I. Damgård, M. Geisler, and M. Krøigaard. A correction to "efficient and secure comparison for on-line auctions". Cryptology ePrint Archive, Report 2008/321, 2008.

DGK08b. I. Damgård, M. Geisler, and M. Krøigaard. Homomorphic encryption and secure comparison. *Journal of Applied Cryptology*, 1(1):22–31, 2008.

DJ01. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC'01*, volume 1992 of *LNCS*, pages 119–136. Springer, 2001.

DK10. I. Damgård and M. Keller. Secure multiparty AES. In *FC'10*, volume 6052 of *LNCS*, pages 367–374. Springer, 2010.

DO10. I. Damgård and C. Orlandi. Multiparty computation for dishonest majority: from passive to active security at low cost. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 558–576. Springer, 2010.

EFG$^+$09. Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies Symposium (PETS'09)*, volume 5672 of *LNCS*, pages 235–253. Springer, 2009.

FAL04. K. B. Frikken, M. J. Atallah, and J. Li. Hidden access control policies with hidden credentials. In *WPES'04*, pages 27–27. ACM, 2004.

FAL06. K. B. Frikken, M. J. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Transactions on Computers*, 55(10):1259–1270, 2006.

FAZ05. K. B. Frikken, M. J. Atallah, and C. Zhang. Privacy-preserving credit checking. In *EC'05*, pages 147–154. ACM, 2005.

FLA06. K. B. Frikken, J. Li, and M. J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *NDSS'06*, 2006.

FMK91. M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *EURO-DAC'91*, pages 50–54. IEEE, 1991.

Fri09. K. B. Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In *DBSec'09*, volume 5645 of *LNCS*, pages 81–94. Springer, 2009.

FS87. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

Für07. M. Fürer. Faster integer multiplication. In *STOC'07*, pages 57–66. ACM, 2007.

Gam85. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, 1985.

Gen09. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC'09*, pages 169–178. ACM, 2009.

GGP10. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 465–482. Springer, 2010.

GH10. C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010. `http://eprint.iacr.org/`.

GHV10. C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT'10*, volume 6110 of *LNCS*, pages 506–522. Springer, 2010.

GKR08. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *CRYPTO'08*, volume 5157 of *LNCS*, pages 39–56. Springer, 2008.

GMS08. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT'08*, volume 4965 of *LNCS*, pages 289–306. Springer, 2008.

GMY04. J. A. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *TCC'04*, volume 2951 of *LNCS*, pages 297–316. Springer, 2004.

Gol01. O. Goldreich. *Foundations of Cryptography*, volume 1: Basic Tools. Cambridge University Press, 2001. Draft available at `http://www.wisdom.weizmann.ac.il/~oded/foc-vol1.html`.

Gol04. O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004. Draft available at `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

GQ09. D. Giry and J.-J. Quisquater. Cryptographic key length recommendation, March 2009. `http://keylength.com`.

GT08. V. Gunupudi and S. Tate. Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In *FC'08*, volume 5143 of *LNCS*, pages 98–112. Springer, 2008.

HKS$^+$10. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS'10*. ACM, 2010.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

IP07. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC'07*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.

IPS08. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO'08*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.

IPS09. Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *TCC'09*, volume 5444 of *LNCS*, pages 294–314. Springer, 2009.

IS10. A. Iliev and S. W. Smith. Small, stupid, and scalable: Secure computing with Faerieplay. In *STC'10*, pages 41–51. ACM, 2010.

JKS08. S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *S&P'08*, pages 216–230. IEEE, 2008.

JKSS10a. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Efficient secure two-party computation with untrusted hardware tokens. In A.-R. Sadeghi and D. Naccache, editors, *Towards Hardware Intrinsic Security: Foundation and Practic e*, Information Security and Cryptography, pages 367–386. Springer, 2010.

JKSS10b. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading server and network using hardware tokens. In *FC'10*, volume 6052 of *LNCS*, pages 207–221. Springer, 2010.

JKSS10c. K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *CHES'10*, volume 6225 of *LNCS*, pages 383–397. Springer, 2010.

JMN10. T. P. Jakobsen, M. X. Makkes, and J. D. Nielsen. Efficient implementation of the Orlandi protocol. In *ACNS'10*, volume 6123 of *LNCS*, pages 255–272. Springer, 2010.

JP09. A. Jarrous and B. Pinkas. Secure hamming distance based computation and its applications. In *ACNS'09*, volume 5536 of *LNCS*, pages 107–124. Springer, 2009.

JS07. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT'07*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.

Jur03. M. Jurik. *Extensions to the Paillier Cryptosystem with Applications to Cryptological Protocols*. PhD thesis, Basic Research in Computer Science, August 2003.

Kah67. D. Kahn. *The Codebreakers — The Story of Secret Writing*. Macmillan Publishing Co, New York, USA, 1967.

KC00. V. Kabanets and J. Cai. Circuit minimization problem. In *STOC'00*, pages 73–79. ACM, 2000.

KJGB06. L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *CCS'06*, pages 410–420. ACM, 2006.

KM10a. J. Katz and L. Malka. Private function evaluation with linear complexity. Cryptology ePrint Archive, Report 2010/528, 2010. `http://eprint.iacr.org/`.

KM10b. J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *CCS'10*, pages 485–492. ACM, 2010.

KO62. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of the SSSR Academy of Sciences*, 145:293–294, 1962.

KS06. M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

KS08a. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP'08*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

KS08b. V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *FC'08*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008.

KSS09. V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS'09*, volume 5888 of *LNCS*, pages 1–20. Springer, 2009.

LB05. W. Lenders and C. Baier. Genetic algorithms for the variable ordering problem of binary decision diagrams. In *FOGA'05*, volume 3469 of *LNCS*, pages 1–20, 2005.

Lip03a. H. Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In *Advances on Cryptology - ASIACRYPT'03*, volume 2894 of *LNCS*, pages 398–415. Springer, 2003.

Lip03b. H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *ASIACRYPT'03*, volume 2894 of *LNCS*. Springer, 2003.

Lip08. H. Lipmaa. Private branching programs: On communication-efficient cryptocomputing. Cryptology ePrint Archive, Report 2008/107, 2008. `http://eprint.iacr.org/`.

LP02. Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

LP07. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT'07*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

LP09a. Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009. Cryptology ePrint Archive: Report 2004/175.

LP09b. Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

LP10. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. Cryptology ePrint Archive, Report 2010/284, 2010. `http://eprint.iacr.org/`.

LPS08. Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN'08*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

MK10. L. Malka and J. Katz. VMCrypt - modular software architecture for scalable secure computation. Cryptology ePrint Archive, Report 2010/584, 2010. `http://eprint.iacr.org/`.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security'04*, 2004. `http://fairplayproject.net`.

MOR03. P. D. MacKenzie, A. Oprea, and M. K. Reiter. Automatic generation of two-party computations. In *CCS'03*, pages 210–219. ACM, 2003.

Nie07. J. B. Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. `http://eprint.iacr.org/`.

NO09. J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *TCC'09*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

NP01. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, pages 448–457. ACM Society for Industrial and Applied Mathematics, 2001.

NPS99. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC'99*, pages 129–139. ACM, 1999.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

PSS09. A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *ACNS'09*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009. `http://www.trust.rub.de/FairplaySPF`.

PSSW09. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

Rud93. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD'93*, pages 42–47. IEEE, 1993.

Sch91. C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

Sch08. T. Schneider. Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, February 27, 2008. `http://thomaschneider.de/papers/S08Thesis.pdf`.

SKB$^+$09. 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.

SKM10. A. Schröpfer, F. Kerschbaum, and G. Müller. L1 - a programming language for mixed-protocol secure computation. Cryptology ePrint Archive, Report 2010/578, 2010. `http://eprint.iacr.org/`.

SS71. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen (Fast multiplication of large numbers). *Computing*, 7(3):281–292, 1971.

SS08. A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *ICISC'08*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

SS10. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *CCS'10*, pages 463–472. ACM, 2010.

SSW09. A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC'09*, LNCS. Springer, 2009.

ST98. T. Sander and C. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume 1419 of *LNCS*, pages 44–60. Springer, 1998.

SV10. N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC'10*, volume 6056 of *LNCS*, pages 420–443. Springer, 2010.

SYY99. T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS'99*, pages 554–566. IEEE, 1999.

TPKC07. J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. U. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *CCS'07*, pages 519–528. ACM, 2007.

Val76. L. G. Valiant. Universal circuits (preliminary report). In *STOC'76*, pages 196–203. ACM, 1976.

vDGHV10. M. v. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT'10*, volume 6110 of *LNCS*, pages 24–43. Springer, 2010.

vDJ10. M. v. Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *HotSec'10*. USENIX, 2010.

Vol99. H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer, Secaucus, NJ, USA, 1999.

Wak68. A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, 1968.

Woe05. P. Woelfel. Bounds on the OBDD-size of integer multiplication via universal hashing. *Journal of Computer and System Sciences*, 71(4):520–534, 2005.

Yao82. A. C. Yao. Protocols for secure computations. In *FOCS'82*, pages 160–164. IEEE, 1982.

Yao86. A. C. Yao. How to generate and exchange secrets. In *FOCS'86*, pages 162–167. IEEE, 1986.

A Where SFE Fits in Secure Computing

Cryptography (from Greek "secret writing") with thousands of years of history [Kah67] 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 Function Representations

B.1 Boolean Circuits

Boolean circuits are 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 [Vol99].

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 $O(|V| + |E|)$ [CLRS01, 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* [ALR99]. 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, e.g., $\oplus, \wedge, =, \vee$ would be possible as well.

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 [MNPS04]. Fairplay's *Secure Function Description Language* (SFDL) resembles a simplified version of a hardware description language, such as VHDL[6] , 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. [MNPS04, 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.2 Arithmetic Circuits

Arithmetic circuits are 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.

B.3 Ordered Binary Decision Diagrams

Another possibility to represent boolean functions are Ordered Binary Decision Diagrams (OBDDs) introduced by Bryant [Bry86].

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 node to a terminal node. 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 $x_1 < x_2 < x_3 < x_4$.[7] 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 $f(1, 1, 0, 0) = 1$.

[6] Very high speed integrated circuit Hardware Description Language

[7] 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.

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* [BPSW07] 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* [BFK$^+$09] 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.

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 [Bry86]. Even though finding an optimal variable ordering for OBDDs is NP-complete [BW96], 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 [Rud93], the window permutation algorithm [FMK91], genetic algorithms [DBG96, LB05], or algorithms based on simulated annealing [BLW95].

Nevertheless, some functions have a lower bound for the size of the smallest OBDD representation which is exponential. For example ℓ-bit integer multiplication has an exponential size OBDD [Bry91, Woe05] but requires only one multiplication gate in an arithmetic circuit over a sufficiently large ring. Multiplication within a boolean circuit has complexity $\mathcal{O}(\ell^2)$ using school method or $\mathcal{O}(\ell^{\log_2 3})$ with the method of [KO62] (indeed, for garbled circuits the latter is more efficient for $\ell \geq 20$ [HKS$^+$10]). 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: $\mathcal{O}(\ell \log \ell \log \log \ell)$ [SS71] and $\ell \log \ell 2^{\mathcal{O}(\mathsf{lg}^* \ell)}$ [Für07][8].

C Intuition Behind SFE Definitions

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 (i.e., the sequence of messages exchanged between the parties) 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 the simulated transcript was constructed without the knowledge of the input, the 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 simulators 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 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 the 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 [Gol04].

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

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

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 will be caught by the honest player with the specified probability. The formal definitions for covert security (three variations) were proposed in [AL07].

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

D 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 [Gol04], 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 §3.1, 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.

D.1 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 without revealing any further information on it. After the protocol execution, verifier decides whether it accepts or rejects the proof. A proof must be complete and sound. Completeness guarantees that the protocol works for any pair $(x, w) \in R$, i.e., for all $(x, w) \in R$ the verifier accepts the proof if both parties 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, an efficient knowledge extractor with black-box access to the possibly malicious prover can be constructed to compute a witness (cf., e.g., [Gol01]). 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 [CDS94, Cra97]. 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 [CDS94].

Σ-protocols can be made non-interactive using the standard Fiat-Shamir heuristic [FS87] by computing the challenge from the first message using a cryptographic 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 [DJ01]. It is also possible to prove various relations about the plaintexts encrypted within a ciphertext [Jur03], 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 [Lip03a].

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 [Pai99, DJ01] – is well-formed using the rather expensive zero-knowledge proof of [CM99].

D.2 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 §4.1 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.

D.3 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 §5.1 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 [Sch91], $\mathcal{C}$ 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.4 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 5.2: $\mathcal{S}$ 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 [Lip03b] 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 $\mathcal{S}$.