Complexity of Multi-Party Computation Functionalities

Hemanta K. Maji* Manoj Prabhakaran† Mike Rosulek‡

October 16, 2012

Abstract

The central objects of secure multiparty computation are the "multiparty functions" (or functionalities) that it seeks to securely realize. In this chapter we survey a set of results that constitute a *Cryptographic Complexity Theory*. This theory classifies and compares multiparty functions according to their secure computability and reducibility to each other. The basic questions studied, under various notions of security and reducibility, include:

- Which functionalities are securely realizable (or are "trivial" – i.e., can be reduced to any functionality)?
- Which functionalities are "complete" – i.e., those to which any functionality can be reduced?
- More generally, which functionalities are reducible to which? Outside of triviality and completeness, this question is relatively less explored.

Reductions yield relative measures of complexity among various functionalities. In the information-theoretic setting, absolute complexity measures have also been considered. In particular, we discuss results regarding which functions have t-private protocols (in which security is required against a passive adversary corrupting t out of n players) and how this set changes as t increases from 1 to n.

We treat separately the results on two-party functionalities, for which the cryptographic complexity is much better understood. In particular, we present unified combinatorial characterizations of completeness and triviality for secure function evaluation using notions of isomorphism and the common information functionality (called the kernel) of a given functionality. Beyond completeness and triviality, we also discuss results on general reducibility, and, in the computationally bounded setting, the connection between these reductions and computational hardness assumptions.

We briefly discuss results on reactive functionalities, which are much less studied than non-reactive (secure function evaluation) functionalities. Finally, we conclude with a selection of open problems.

*Department of Computer Science, University of California, Los Angeles. hmaji@cs.ucla.edu.

†Department of Computer Science, University of Illinois, Urbana-Champaign, mmp@uiuc.edu. Supported by NSF grants CNS 07-47027 and CNS 12-28856.

‡Department of Computer Science, University of Montana. mikero@cs.umt.edu. Supported by NSF grant CCF-1149647

Contents

1 Introduction

Computational complexity studies the amount of computational resources needed for a computing device to evaluate various functions. In this setting, a function takes a single input and produces a single output. (Indeed, multiple input and output values can simply be encoded as a single values.) However, in the cryptographic context of *multiparty computation*, it is also significant which inputs are provided by which parties, and which outputs are delivered to which parties. Quite apart from the question of computational complexity, new interesting parameters emerge — regarding how the various inputs and outputs are related to each other. Some important aspects of multiparty functions are studied, for instance, under the umbrella of *communication complexity* (number of bits, rounds of interaction, etc. required for evaluating the function). In this chapter we survey *cryptographic complexity* of multiparty functions that deals with yet another aspect of how intricately the inputs and outputs are related to each other.

We illustrate cryptographic complexity with an example. Consider two 2-party functions, XOR and OR, both of which take two bits of input — x from Alice and y from Bob — and output a single bit to both parties; XOR outputs $x \oplus y$, and OR outputs $x \vee y$. In terms of communication complexity, both these functions are of similar complexity, requiring (in the worst case) the parties to exchange their inputs with each other. If anything, OR is slightly simpler, as the average amount of communication needed over random inputs is lesser. However, in a cryptographic sense OR is much more complex than XOR: XOR hides no information about the parties' inputs from each other, in that Alice can infer Bob's input y from $(x, x \oplus y)$; similarly Bob can infer Alice's input x. However OR hides information in a peculiar manner: if Alice's input is 1, she does not learn Bob's input; but if her input is 0, she does learn Bob's input; but note that Bob (if his input is 1) should not learn whether Alice learned this information or not! As it turns out, there is no way[1] for Alice and Bob to carry out the computation of OR, such that they learn *nothing but the value of the function* (in addition to their own respective inputs). That is to say, OR *is not computable* in a cryptographic sense. On the other hand, XOR is computable using a straightforward protocol in which the parties simply exchange their bits.[2]

As it turns out, most of the interesting 2-party functions are not "trivial" (i.e., securely computable) unless corrupt parties' computational resources are limited. Then, to distinguish among cryptographically uncomputable functions, one needs a finer gradation of complexity. Following the lead of Turing, who introduced the notion of oracles and (what is now called) Turing reductions, it is possible to differentiate between cryptographically uncomputable functions using an appropriate notions of a reduction. We shall say that a multiparty function F "reduces to" another multiparty function G (written $F \sqsubseteq G$), if there is a protocol that securely computes F, in which the parties have access to an "oracle" for G. Typically, the parties are allowed to access the oracle any number of times during the protocol; each time they access the oracle, they feed inputs of their choice to the oracle (without seeing the other parties' inputs), and the oracle returns to each party its prescribed G-output alone. Intuitively, if F reduces to G, then F is no more complex than G is. Technically, we can choose a few different but natural notions of reductions, corresponding to the security requirements on the protocols (analogous to how reductions in computational complexity theory can be deterministic, non-deterministic, probabilistic etc.), and classify the functions into different complexity classes accordingly.

We offer a simple description of the subject of this chapter:

> *Cryptographic Complexity Theory* is the study of multiparty functionalities, classified according to their cryptographic computability and reducibility to each other.

Note that we consider the objects of study in this theory to be *multiparty functionalities*, which are more general than multiparty functions as described above. In particular, we shall be interested in the following cryptographic tasks:

- Securely sampling values for two or more parties, that are correlated with each other. More generally this falls under the scope of securely evaluating *randomized functions*.

[1] This refers to the information-theoretic setting.

[2] This protocol is secure in the "honest but curious" adversary model. In stronger security models, XOR too becomes an uncomputable function.

- Computations that involve persistent state being maintained across several calls, where parties provide inputs over multiple rounds and outputs may depend on the inputs from past rounds. In the language of multi-party computation, such objects are called *reactive functionalities*. The "canonical" nonreactive functionality is the two-round "commitment" function: in the first round only Alice has an input x, and no party gets any output; in the second round, neither party has an input, and Bob receives x as an output. The significance of this function is that the parties get to schedule other interaction between these two rounds, during which Alice stays "committed" to her input x without Bob learning anything about it. Such reactive functionalities, apart from being interesting in their own right, turn out to be important in understanding the complexity of single-round functions too.

In general, a *functionality* is defined by the code of a possibly interactive "oracle" who receives inputs, carries out the computation, and gives outputs.

A Brief History of Cryptographic Complexity. While cryptographic computation of multiparty functions (more commonly referred to as secure multiparty computation) was studied starting in late 1970's [SRA79, Rab81, Blu81, Yao82], perhaps the first explicit reference to a cryptographic complexity aspect of a function was not until Kilian's work that identified a "complete" function [Kil88]. Previous works focused on studying which multiparty functions are *computable* under various definitions of cryptographic computability prevalent then, and under suitable computational intractability assumptions [Yao86, GMW87, BGW88, CCD88]. Indeed, in the settings in these works, all functions were computable and hence all were complete too, blurring any distinctions in complexity of the different functions. Since Kilian's result, completeness was more fully investigated [Kil91, KKMO00, Kil00, FGMO05, KM11, Ros12]. Meanwhile the characterization of cryptographically computable functions was also carried out for several more settings [CK89, Kus89, Bea89, CKL03, PR08, KMR09, MPR09]. However, functions of intermediate complexity — those which are neither computable, nor complete — were known to exist in some settings [Bea89, Kus89] but by and large not explored until recently. The term *cryptographic complexity* was introduced in [PR08].

The above works on cryptographic complexity deal with the computationally unbounded setting. The computational aspect of cryptographic complexity was investigated more recently. Beimel et al. [BMM99] were perhaps the first to investigate the computational intractability requirements *necessary and sufficient for* cryptographic computation of multiparty functions. For a certain interesting class of 2-party functions, they show that *any* function that is information-theoretically uncomputable is computable against computationally bounded adversaries if and only if there exists a semi-honest oblivious transfer protocol (what we call the sh-OT assumption in this survey). Since then, this problem has been extended in several dimensions: class of functions, types of security, and reducibility rather than just computability (for instance, in [MPR10a, MPR10b, MOPR11, MP11]).

2 Preliminaries

As described in earlier chapters, the security of a multi-party computation protocol is defined by comparison to an ideal *functionality*. A functionality is a trusted party that takes inputs from a fixed number of parties,[3] $P_1, \ldots, P_n$, carries out a computation, and gives output to parties. A *reactive functionality* can interact with the parties over multiple rounds, sending them outputs at each round computed from the inputs and outputs in all previous rounds. Invariably, we will consider functionalities which take time at most polynomial in their input and a security parameter. The *real-ideal* paradigm for security [GMW87] is used to define security for multi-party computation. Informally, *a protocol securely realizes a functionality* if for every adversary attacking the protocol, there is an adversary which can achieve the same effect when the computation is carried out (honestly) by the functionality. Depending on the type of the security required, the capabilities of the adversary vary. A functionality's specification (say, as an interactive Turing machine) plus the security

[3] This number is called the *cardinality* of the functionality. While it is possible to consider functionalities with variable cardinality, all the results we survey concern themselves with fixed cardinality functionalities.

model (say, security against active adversaries) completely determines all of the implicit and explicit security requirements of a multi-party computation task.

2.1 Functionalities

We briefly summarize the different kinds of functionalities in this survey. There are a few important classes of functionalities that have been studied significantly more than the others. **Secure function evaluation (SFE)** functionalities accept inputs from all parties, then performs a computation and gives outputs to all parties. An SFE functionality $\mathcal{F}$ of cardinality n is defined by n functions $f_1, \ldots, f_n$; this functionality, denoted more explicitly as $\mathcal{F}(f_1, \ldots, f_n)$, accepts inputs $(x_1, \ldots, x_n)$ from the n parties respectively, (optionally) randomly samples an element r from some domain, and then provides $f_i(x_1, \ldots, x_n; r)$ to the P_i. The following classification of SFE functionalities is useful to describe the results in the literature:

- **Symmetric SFE (SSFE),** for which $f_1 = \cdots = f_n$. That is, all parties get the same output.
- **Asymmetric SFE,** for which (say) $f_1 = \cdots = f_{n-1}$ are constant functions. In other words, only P_n gets an output. This is of most interest when $n = 2$.
- There are SFE functionalities which fall into neither of these classes. Sometimes we will use the term general SFE to stress that we are considering an SFE which is not necessarily of the above two types.

The universe of functionalities considered is that of "well-behaved" and "regular" functionalities: a well-behaved functionality does not behave differently based on which parties are corrupt (except possibly in allowing the adversary some control over the timing of message delivery to the honest parties); a regular functionality does not communicate with the adversary directly (except via corrupt parties).[4] We make the following distinctions based on how the adversary can control the delivery of messages from the functionality.

• An *unfair functionality* will deliver the output first to the corrupt parties, and then deliver output to remaining honest parties only if the adversary instructs it to (and only to the subset of honest parties specified by the adversary).

• In a *fair functionality* the adversary can instruct the functionality to deliver the output to all parties or to none, without first seeing the corrupt parties' outputs.

• *Functionalities with guaranteed output delivery* offer an even stronger notion of fairness: the adversary cannot control the output delivery at all; even if the adversary refuses to provide an input to the functionality, it will be replaced by a pre-determined dummy input, and the functionality proceeds to carry out the computation and deliver the output to all parties.

Among 2-party functionalities, the literature pre-dominantly deals with unfair functionalities; among functionalities with larger cardinality, when information-theoretic security is considered, often functionalities with guaranteed output delivery are considered.

Many of the results we consider are for the universe of *finite functionalities*. In a finite functionality, the input and output spaces and (in the case of reactive functionalities) the state-space of the functionality are all finite — that is, have size $O(1)$ as a function of the security parameter. In particular, the maximum number of bits needed to represent the input to the parties (and, in the case of randomized functionalities, the number of bits in the random tape of the functionality) does not grow with the security parameter.

For clarity, we shall also define *bounded functionalities* wherein for each value of the security parameter (or more generally, some common parameter in the input), the functionality restricted to that security parameter is finite.[5] Note that as the security parameter grows, the input space of the functionality could grow with it, and so a bounded functionality need not be a finite functionality as defined above. Most of the secure protocols in the literature are for bounded functionalities. We shall always require that for bounded

[4] It is often useful to securely realize functionalities that are not well-behaved or regular and use them in realizing other functionalities. However, the universe of functionalities dealt with in complexity classifications is invariably that of well-behaved and regular functionalities. For instance, for a functionality $\mathcal{F}$ to be complete, it is enough that every well-behaved and regular functionality reduces to $\mathcal{F}$.

[5] Thus, a bounded functionality can be implemented by a circuit family.

functionalities that can be implemented in polynomial time (polynomial in the security parameter) the secure protocols be polynomial time themselves. In particular, note that the input domain and state space of such a functionality should be at most exponential in the security parameter. On the other hand, many of the negative results for finite functionalities in fact extend to bounded functionalities where the input domain and state space are polynomially bounded in the security parameter. A few characterizations that deal with non-finite 2-party functionalities are discussed in Section 5.2.

2.2 Protocols and Reductions

In a protocol for a functionality $\mathcal{F}$, the participating parties receive inputs (from the input domain of $\mathcal{F}$) from an external environment, interact with each other, and produce outputs for the environment. If $\mathcal{F}$ is a reactive functionality, the parties in the protocol may accept inputs from and produce outputs for the environment at multiple points in the protocol. In Section 2.3 we briefly mention the various security definitions dealt with in this survey.

Two standard models for communication among the parties provide *a public-discussion medium* (in which case the adversary can listen to all the messages) and *private point-to-point channels*, respectively. In either model, the adversary fully controls the *scheduling of the delivery of messages*. An important exception to this is the standard model used for protocols that securely realize fair functionalities or functionalities with guaranteed output delivery: in this case the communication is "synchronous" (proceeding in rounds, allowing a party to time-out when a message does not arrive in its specified round), but with a "rushing adversary" (which allows the corrupt parties to receive messages sent to them in a round, before sending out their own messages in that round).

A protocol π for a functionality $\mathcal{F}$ may use a functionality $\mathcal{G}$ as a *setup*:[6] in such a protocol the parties can invoke one or more instances of the ideal functionality $\mathcal{G}$ (implemented by an uncorruptible external party) and interact with it (i.e., send inputs and obtain outputs). The instantiation of the protocol with access to $\mathcal{G}$ will be indicated as $\pi^{\mathcal{G}}$. If there is a protocol $\pi^{\mathcal{G}}$ that securely realizes a functionality $\mathcal{F}$, then we say $\mathcal{F}$ reduces to $\mathcal{G}$ (written as $\mathcal{F} \sqsubseteq \mathcal{G}$); the exact notion of reduction corresponds to the specific notion of security provided by the protocol. Among non-reactive functionalities, $\sqsubseteq$ is a transitive relation; for passive security and UC security, it remains transitive even when considering reactive functionalities.

2.3 Security Models

As mentioned above, a protocol is said to securely realize a functionality $\mathcal{F}$ if, informally, for every adversary attacking the protocol, there is an adversary (called a simulator) which corrupts the same set of parties can achieve an indistinguishable effect (from the point of view of the environment) when the computation is carried out by the functionality. The advantage in distinguishing is typically required to be negligible in the security parameter. (In *perfect* security, this advantage is required to be 0.) The actual security notion depends on the adversary's capabilities, as discussed below.

There are three main dimensions to security definitions that we consider.

- *Adversarial behavior:* The adversary could be passive (a.k.a. honest-but-curious) or active (a.k.a. byzantine, or malicious). In the latter case, the adversary could be standalone (does not interact with the environment during the course of the protocol, except via protocol input/output for reactive functionalities), or not. Correspondingly, we have three notions of security: **passive security**, **standalone security** and **UC security**.

- *Computational power:* We consider either the information-theoretic (a.k.a. statistical, or computationally unbounded) setting or the probabilistic polynomial time (PPT) (a.k.a. computationally bounded, or, simply, computational) setting. Typically all entities (in particular, the adversary, simulator, and

[6] In many works in the PPT setting, when π has a setup like the "common random string" (which takes no inputs), π can access only a single instance of the setup functionality, and further, polynomially many instances of π should access this same instance. While there are important positive results in such a model, it is outside the scope of this survey.

environment) have the same kind of computational power, except the functionalities and the protocols (honest parties) which are required to be PPT (w.r.t. the security parameter and inputs).

- *Adversary structures:* The set of parties that are allowed to be corrupted at once could be restricted (especially when results in the information-theoretic setting are desired). In particular, in a threshold adversary structure with a threshold t, no more than t parties can be corrupted at once. Such restrictions are typically considered in the information-theoretic setting.

Any combination of choices in each dimension is possible, though some combinations have received more attention in literature than the others.

Above we introduced the symbol $\sqsubseteq$ to indicate reducibility (without being specific about the security model). By default we use this symbol to denote reducibility in the information-theoretic setting, and write $\sqsubseteq_{\mathsf{ppt}}$ for the PPT setting. The relations for reducibility in the passive, standalone and UC security models will be denoted by $\sqsubseteq^{\mathsf{sh}}$, $\sqsubseteq^{\mathsf{sa}}$ and $\sqsubseteq^{\mathsf{uc}}$ respectively, and ($\sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}}$ etc. in the PPT setting).

Omitted Security Models. Apart from the above dimensions, another dimension of the security definition is whether the adversary is allowed to corrupt parties over the course of the protocol (adaptive corruption) or only at the beginning (static corruption). In this survey, we focus on the static corruption setting, though many of the results do extend to the adaptive corruption setting. Also, along the first dimension, several models intermediate to standalone security and UC security are considered in the literature (e.g. concurrent security and non-malleability); the known positive results (existence of reductions) in this case are often specialized for specific functionalities (e.g. zero-knowledge proofs and commitment), and otherwise often the negative results (separations) for UC security extend to these security notions. For the sake of focus, we omit these intermediate notions of security from this survey.

2.4 Cryptographic Complexity

The *cryptographic complexity* of a multi-party computation functionality can be thought of as either "the difficulty of creating a secure protocol for the functionality" (hardness) or "how much does access to the functionality help in creating secure protocols for other functionalities" (usefulness). Quantitative measures of complexity like privacy-threshold (see Section 3.1.1) are best interpreted in the former manner;[7] the qualitative, reduction-based notion of complexity, on the other hand, explicitly relates the usefulness of one functionality to the hardness of another.

We highlight two natural extremes of complexity, where the two interpretations — hardness and usefulness — coincide. Firstly, if a functionality has a secure protocol (in some security model) without any setup (i.e., the functionality is *realizable* in that security model), then it is of no value as a setup, as the access to such an ideal functionality can be replaced by an implementation. We shall refer to such functionalities as **trivial functionalities** (for the corresponding security model or reduction). Trivial functionalities are neither hard nor useful in the sense defined above. In terms of reducibility, a trivial functionality reduces to every functionality. At the other extreme, we can consider functionalities to which every functionality reduces. Such functionalities, any of which can replace any other setup, are called **complete functionalities** (for the corresponding security model or reduction). These most useful functionalities are also essentially the hardest functionalities to realize, since once a complete functionality is realized somehow, any other functionality can be realized using a reduction to the former (provided that the security model allows secure composition). Of course, other intermediate levels of complexity are possible between triviality and completeness, and we shall see results regarding them as well. But much of the current literature on cryptographic complexity focuses on these two natural extremes.

For brevity, we shall write "**passive-trivial**," "**UC-complete**," etc. to stand for "trivial w.r.t. reductions that are secure against passive adversaries," "complete w.r.t. reductions that are UC-secure" etc.

[7] Though not covered in this survey, there are also useful quantitative complexity measures related to reducibility. In particular, one can measure complexity in terms of how many instances of a setup $\mathcal{G}$ are needed per copy of $\mathcal{F}$ in a protocol that reduces $\mathcal{F}$ to $\mathcal{G}$— a complexity measure that has connections with circuit complexity.

3 Information-Theoretic Results

In this section we discuss several results in the computationally unbounded setting — that is, when information-theoretic security is required. In general, unless otherwise specified we assume point-to-point channels between all pairs of parties.

3.1 Triviality of Functionalities

Recall that we say that a functionality is trivial (under a certain reduction) if it is securely realizable without any setups (under the corresponding security model). In the information-theoretic setting, one studies the constraints on adversarial structures (see Section 2) under which a functionality becomes trivial. In Section 3.1.1 we survey results in the passive corruption model, and in Section 3.1.2, those in the active corruption model.

3.1.1 Passive-Triviality and t-privacy

In the passive corruption model with point-to-point channels, the most commonly studied complexity measure of functionalities is called **t-privacy**. (In this context, the term privacy signifies security against passive corruption.) Unlike the notion of complexity based on reductions, this is an absolute measure rather than a relative measure of complexity. A functionality f is t-private if there exists a protocol for f that is secure against any passive corruption of at most t parties. In other words, the combined view of any coalition of t parties reveals no more information about the honest parties' inputs than the prescribed output to the coalition.

For convenience, we shall say that a functionality has **privacy-threshold** t if it is t-private, but not $(t+1)$-private. Note that every n-party functionality has a well defined privacy-threshold t, with $0 \leq t \leq n$. (Throughout this section, we let n denote the cardinality of a functionality.) A higher value of the privacy-threshold indicates the functionality is *less* complex: in particular, functionalities with privacy-threshold n are precisely the ones which are passive-trivial (i.e., passive-securely realizable, without any restriction on the number of parties that can be corrupted). Lower values of privacy-threshold indicate difficulty to securely realize the functionalities (rather than usefulness as a setup in realizing other functionalities).

The *privacy hierarchy*, for each value of n, refers to the ordering of n-party functionalities according to their privacy-threshold. The primary question regarding privacy-threshold has been the structure of the privacy hierarchy. We point out that $(n-1)$-privacy implies n-privacy, and hence no functionality can have privacy-threshold $(n-1)$.

The results on the privacy hierarchy have largely focused on symmetric SFE functionalities (SSFE). Such functionalities are completely described by a deterministic function $f : X^n \to Y$, where X is the *input space* and Y is the *output space*. Each party i has input x_i, and all parties learn $f(x_1, \ldots, x_n)$.

The first broad characterizations for t-privacy were by Ben-Or, Goldwasser, and Wigderson [BGW88], and independently Chaum, Crépau, and Damgård [CCD88], who showed that all bounded n-party deterministic SFE functionalities are $\lfloor \frac{n-1}{2} \rfloor$-private. That is, all such functionalities have secure protocols against passive adversaries in the presence of an honest (strict) majority. The restriction to bounded functionalities is necessary; Chor, Geréb-Graus, and Kushilevitz [CGK95] showed that if the input domain is allowed to be unbounded, then there are functionalities (e.g., the n-argument addition function over $\mathbb{Z}$) which are not even 1-private. (In contrast, addition over *non-negative* integers is $\lfloor \frac{n-1}{2} \rfloor$-private, albeit using an inefficient protocol.)

The restriction to deterministic SFE functionalities can be removed. Any randomized reactive functionality (with possibly different outputs to the different parties) can be passive-securely (in fact, UC-securely) reduced to a deterministic SSFE functionality, using standard techniques.[8] Thus we have the following

[8] A randomized SFE can be reduced to deterministic SFE which accepts shares of random-tapes for the original functionality as input from the n parties. A general SFE can be reduced to a symmetric SFE in which each party inputs an additional string that is used as a one-time pad to mask its part of the output. A reactive functionality can be reduced to a non-reactive functionality by secret-sharing the internal state of the functionality among all n parties. (For standalone and UC-security this

theorem.

Theorem 1 *All bounded n-party functionalities are $\lfloor \frac{n-1}{2} \rfloor$-private.*

Effect of the size of the output space. Characterizations of privacy-threshold depend on the output space of the functionalities in question. At one extreme, among functionalities with a *binary* output space (i.e., 2 possible outputs), the privacy hierarchy is not dense but has only two non-empty levels [CK89]. Kreitz [Kre11] showed that this holds for ternary output spaces as well.

Theorem 2 *Every bounded deterministic n-party SSFE functionality with* a ternary output space *has privacy-threshold either $\lfloor \frac{n-1}{2} \rfloor$ or n.*

Further, [CK89, Kre11] also gave combinatorial characterizations of the functionalities in the two levels of the privacy hierarchy in these cases. In particular, in the case of binary outputs, the n-private SSFE functionalities have a particularly simple form: they are exactly those which evaluate functions of the form $B_1(x_1) \oplus \cdots \oplus B_n(x_n)$, where each B_i is a boolean predicate, and x_i is the input of party P_i. If the input space is unbounded and the output space is binary, then in addition to the above two classes, there is exactly one additional possible level of privacy: functionalities which have privacy-threshold 0 (i.e., which are not even 1-private) [CGK95]. Chor and Shani [CS95] showed that the above privacy dichotomy extends to a subset (called "dense symmetric functions") of the class of functions in which each party's input space is binary and the output is a function of the Hamming weight of the set of inputs (and hence the output space is $\{0, \ldots, n\}$).

In the case that the output space is unrestricted, complete characterizations of t-privacy are not known. However, it is known that Theorem 2 does not hold in general: Chor, Geréb-Graus, and Kushilevitz [CGK94] showed that for every n and every $t \in \{\lfloor \frac{n-1}{2} \rfloor, \ldots, n-2\}$, there exists an n-party SFE functionality that has privacy-threshold t. (The functionality that is constructed in [CGK94] has an output space of 2^{t+2} elements. In particular, there is a 4 party functionality with an output space of size 16 with privacy-threshold 2.)

While complete characterizations are not known for the t-privacy of general multi-party functionalities, some necessary conditions are known. Let f be an n-party, t-private function and denote its inputs as $(x_1, \ldots, x_n)$. Consider a set $S \subseteq \{1, \ldots, n\}$ such that $|S| \leq t$ and $|\overline{S}| \leq t$ (note that the question of t-privacy in general is only interesting for $t \geq n/2$), and let $f_{S,\overline{S}}$ denote the interpretation of f as a 2-party function of inputs $\{x_i \mid i \in S\}$ and $\{x_i \mid i \notin S\}$. Then for any such S, $f_{S,\overline{S}}$ must be 1-private. In general, one may also consider partitions of the set of parties into any number of groups. While this technique yields a necessary condition, Chor and Ishai [CI01] proved that it does not give a sufficient condition for t-privacy. In particular, there are n-party functionalities f for which every partition into k parties, for all $k < n$, is fully private (i.e., k-private), yet f itself is not fully private (i.e., n-private).

For the special case of **two-party** functionalities, there are only two levels of the privacy hierarchy: privacy-threshold being 2 or 0, corresponding to functionalities that are passive-trivial or not. In fact, a complete combinatorial characterization is known for the two-party functionalities that are 2-private [Kus89, Bea89, KMR09, MPR09]. We discuss this further in Section 5.1.2. In the *public discussion model*, this characterization generalizes to n-party functionalities as well (with no restriction on the number of parties corrupted) [KMR09].

3.1.2 Triviality for active security

In this section we consider the standalone and UC security models in which corrupt parties are allowed to arbitrarily (a.k.a. actively, or maliciously) deviate from the protocol.

When considering active adversaries, one of the most important considerations is ensuring that honest parties can output consistent values, even when corrupt parties deviate from the protocol. The multi-party results in this section consider a security model that includes the requirement of *guaranteed output delivery*. In this model, honest parties in the ideal world always receive output from the functionality (i.e.,

must be done in a way that is robust to cheating parties; a *non-malleable* secret-sharing scheme can be used for this [IPS08].)

the adversary cannot block the functionality's output message); thus, the real/ideal notion of security entails that honest parties give consistent output when running the protocol as well.

Ben-Or, Goldwasser, and Wigderson [BGW88], and indpendently Chaum, Crépau, and Damgård [CCD88] showed that every bounded n-party SFE functionality with guaranteed output delivery is securely realizable against active adversaries who corrupt strictly less than $n/3$ of the parties.[9] In fact, the protocol in [BGW88] achieves perfect security. Security of these protocols was proven in the standalone setting, and was later extended to UC security by Canetti [Can01]. As in Footnote 8, the restriction to SFE functionalities can be removed.

Theorem 3 *All bounded n-party functionalities with guaranteed output delivery are UC-trivial against adversaries that corrupt at most $\lfloor \frac{n-1}{3} \rfloor$ parties.*

The above $\lfloor \frac{n-1}{3} \rfloor$ bound is tight, in the sense that there exist functionalities that are insecure against adversaries who actively corrupt $\lceil \frac{n}{3} \rceil$ parties. Pease, Shostak, and Lamport [PSL80] showed that *perfectly secure* "Byzantine agreement" — and by extension, perfectly secure "broadcast"[10] — is possible if and only if less than one-third of the parties are actively corrupted. The impossibility was later extended to the case of vanishing security error (in fact, any error less than 1/3) by Karlin and Yao [KY86] (cf. [GY89]).

Mixed & generalized adversaries. Standard security models consider adversaries who either corrupt a set of parties passively, or corrupt a set of parties actively. In an attempt to gain a unified understanding of the relationship between passive and active corruption, *mixed* adversaries have been considered. A mixed (threshold) adversary is allowed to corrupt t_a of the parties actively and additionally corrupt t_p of the parties passively. Additionally, a mixed adversary can corrupt t_f additional parties in a fail-stop manner: the adversary can make these parties stop responding at any time, but cannot see their view of the protocol or make them deviate from the prescribed protocol.

Fitzi, Hirt, and Maurer [FHM98] showed that perfectly secure n-party computation is possible if and only if $3t_a + 2t_p + t_f < n$, with point-to-point channels. If a negligible error is allowed, then n-party computation is possible if and only if $2(t_a + t_p) + t_f < n$ and $3t_a + t_f < n$. The second inequality is not needed if a broadcast channel is available. As above, guaranteed output delivery is required.

Beyond adversaries that corrupt a threshold number of parties, we may also consider *generalized* adversaries. Let $\mathcal{S}$ denote a family of subsets of $\{1, \ldots, n\}$. Then a generalized adversary parameterized by $\mathcal{S}$ can corrupt a set S of parties, for any $S \in \mathcal{S}$. Generalized adversaries can also be mixed, so that $\mathcal{S}$ is a set of (say) triples of subsets of $\{1, \ldots, n\}$, to indicate the allowable sets of parties that can be simultaneously actively, passively, and fail-stop corrupted. Several works have generalized the previous tight threshold bounds described above to the setting of generalized adversaries [Cha89, HM00, FHM98, FHM99, HLMR12]. As an illustrative example, consider an adversary who corrupts only actively (thus, $t_p = t_f = 0$). Then a threshold condition of $3t_a < n$ is translated into the analogous (generalized) condition that no three elements of $\mathcal{S}$ cover the set of all parties. We leave the detailed characterizations outside the scope of this survey.

3.2 Complete Functionalities

First we shall consider completeness for functionalities without guaranteed output delivery. The case of functionalities with guaranteed output delivery is considered in Section 3.2.1.

The first functionality to be identified as *complete* in the information-theoretic setting is the **oblivious transfer (OT)** functionality. The simplest form of OT is the *1-out-of-2* variant [Wie83, EGL85]: A sender prepares two bits x_0, x_1 as input, and a receiver prepares an input choice bit b. The receiver then learns x_b. The sender learns nothing about b, and the receiver learns nothing about x_{1-b}. More generally, in k-out-of-n

[9] Here synchronous, point-to-point channels among all parties are assumed.

[10] The broadcast functionality simply accepts a message from one party and delivers it to all the other parties. Once the functionality is invoked it is guaranteed that all the honest parties will get the output. While non-trivial, this is still a relatively simple functionality (there are no secrecy requirements, for instance) and can often be physically implemented in a system. See Section 3.2.1 for the completeness of this functionality.

OT, the sender prepares bits (or strings) $x_1, \ldots, x_n$, the receiver prepares an input $S \subseteq \{1, \ldots, n\}$ with $|S| = k$, and the receiver learns $\{x_i \mid i \in S\}$.

To see why OT could be (passive) complete, consider the case of finite 2-party deterministic SFE functionalities. It is easy to see that the following protocol passive-securely reduces any such functionality to 1-out-of-k OT (which in turn can be relatively easily reduced to 1-out-of-2 OT), where k is the size of the input domain of P_2. P_1 prepares a vector $(f(x,1), \ldots, f(x,k))$ and P_2 uses 1-out-of-k OT to receive $f(x,y)$, which he sends back to P_1.

For more general functionalities, it follows from the protocol of [GMW87] (and simplifications from [HM86, GV87]) that 1-out-of-2 OT is complete against passive adversaries, in the information theoretic setting for any number of parties. These protocols proceed by performing a gate-by-gate evaluation of a boolean circuit for the function, in which the parties maintain a secret sharing of the values of the wires in the circuit. For AND and OR gates, a sharing of the output can be computed from the shares of the input wires with the help of pairwise OT among the parties. For instance, in the 2-party setting, z_1, z_2 such that $z_1 \oplus z_2 = (x_1 \oplus x_2)(y_1 \oplus y_2)$ is computed as follows: the party P_1 picks a random bit r_1, and uses OT to transfer $r_2 = r_1 \oplus x_1 y_2$ to P_2 (i.e., P_1 inputs $(r_1, r_1 \oplus x_1)$ as the sender in OT, and P_2 inputs y_2 as the receiver); similarly, they pick s_1 and s_2 respectively, such that $s_1 \oplus s_2 = y_1 x_2$; then the two parties locally compute $z_1 = x_1 y_1 + r_1 + s_1$ and $z_2 = x_2 y_2 + r_2 + s_2$, which form a random share of the value $(x_1 \oplus x_2)(y_1 \oplus y_2)$. To compute NOT gates, one of the parties can locally flip her share to obtain new shares of the output wire.

In a seminal work, Kilian [Kil88] showed that the OT functionality is complete also against active adversaries, in the information-theoretic setting. The security of the protocol was shown in the standalone setting, but does extend to a composable security setting [Kil89]. A simpler and more efficient construction, which explicitly showed that OT is complete with respect to UC security was given in [IPS08].

Combinatorial Characterizations. Many varied relaxations of OT were known to be equivalent to standard 1-out-of-2 OT under information-theoretic reductions [Rab81, EGL85, Cré87], and thus were shown complete when OT was shown to be complete. Subsequently, detailed combinatorial characterizations of complete two-party functionalities were developed [Kil88, Kil91, Kil00, KM11]. A unified description of these results is presented in Section 5.

For SFE functionalities of larger cardinalities, characterizations of completeness are known for only certain classes of functionalities. Kilian et al. [Kil91, KMO94, KKMO00] showed that every finite SSFE functionality with *boolean* output is either fully n-private, or else passive-complete. This result complements the earlier combinatorial characterizations of n- and $\lfloor \frac{n-1}{2} \rfloor$-privacy of boolean-output SSFE functionalities by Chor and Kushilevitz [CK89]. Thus, any SSFE functionality that evaluates a boolean function that is not of the form $B_1(x_1) \oplus \cdots \oplus B_n(x_n)$, where each B_i is a boolean predicate, is passive-complete.

3.2.1 Completeness for Functionalities with Guaranteed Output Delivery

Since all bounded functionalities with guaranteed output delivery are trivial (and hence complete) against adversaries that corrupt at most $\lfloor \frac{n-1}{3} \rfloor$ parties, the question of completeness has primarily dealt with larger corruption thresholds. The first such result showed that the broadcast functionality (see Footnote 10) is standalone-complete (for the class of bounded functionalities with guaranteed output delivery) against adversaries that corrupt at most $\lfloor \frac{n-1}{2} \rfloor$ parties [RB89].[11] The threshold $\lfloor \frac{n-1}{2} \rfloor$ is tight: for instance, Cleve [Cle86] showed that a multiparty coin-tossing functionality cannot be reduced to any communication functionality (including broadcasts and multicasts) in the presence of $\lceil n/2 \rceil$ corrupt parties.

Fitzi and Maurer [FM00] later showed that a much simpler functionality is complete (for the same threshold): this functionality lets a party broadcast to any two receivers (rather than broadcasting to all other $n-1$ parties). Further, Fitzi et al. [FGMO05] showed that for this threshold, it is necessary that a complete functionality has cardinality (number of parties) of 3 or more: in fact, no functionality with

[11] Unlike in the case of the protocol in [BGW88], this is not a *perfectly secure* reduction. Indeed, when up to $\lfloor \frac{n-1}{2} \rfloor$ parties can be corrupted, there is no perfectly secure reduction of every functionality to the broadcast functionality. citation?

cardinality two is complete if $t \geq n/3$ parties could be corrupt. They conjecture that a functionality must have cardinality n to be complete for secure computation against $\lceil n/2 \rceil$ or more corrupt parties (they prove this when security is required against corruption of $n-2$ players). Further, they point out that for any threshold, one can indeed construct a complete function with cardinality n, namely a "Universal Black-Box" function.

3.3 Intermediate Levels of Complexity

By comparing the characterizations of complete and trivial functionalities (for the settings where they are known), it can be seen that there are functionalities that do not fall into either of these classes. Concrete examples of such 2-party functionalities are discussed in Section 5.1.5. There is relatively less work trying to differentiate among the various functionalities of intermediate complexity. Section 5.1.6 describes some known structure including the existence of infinitely many distinct "degrees" of reduction for 2-party functionalities. Functionalities of intermediate complexity with more than two parties have not been well-studied.

4 Results in the Computational Setting

The results in the computational or PPT setting (which do not hold in the information theoretic setting) depend on *assumptions* regarding what computational problems can be solved by PPT machines. While these assumptions are widely believed, given the state-of-the-art in computational complexity theory, we do not have the means to prove any of them (all these assumptions typically imply $\mathbf{P} \neq \mathbf{NP}$). Two assumptions in particular are important to our discussion.

1. **OWF assumption**: There exists a one-way function (see [Gol01]).
2. **sh-OT assumption**: There exists a semi-honest OT protocol. That is, in the PPT setting, there is a passive-secure protocol for the oblivious transfer functionality $\mathcal{F}_{\text{OT}}$.

These are increasingly stronger assumptions: i.e., sh-OT assumption $\Rightarrow$ OWF assumption [IL89].[12] In fact, in a formal sense — namely that of "fully black-box reductions" [IR89, RTV04] — these assumptions are "distinct;" i.e., the reverse implication *does not* hold.

The high-level complexity question in the PPT setting, which bridges cryptographic complexity and computational complexity, is how the "cryptographic complexity landscape" changes as we go from assuming no computational hardness (say, $\mathbf{P} = \mathbf{PSPACE}$), through making relatively weak hardness assumptions (like the OWF assumption) to stronger ones like the sh-OT assumption.

4.1 Trivial Functionalities in the PPT Setting

Early motivating examples of secure multiparty computation problems included "mental poker" [SRA79], coin flipping by telephone [Blu81] and the "millionaires' problem" [Yao82]. Moving beyond such specific problems, a few years later Yao showed that *any* bounded two party SFE functionality is passive-trivial in the PPT setting, under specific computational assumptions (originally, the hardness of factoring) [Yao86].[13] In fact, this construction, called the Yao's Garbled Circuit construction, could be seen as a passive-secure reduction of any SFE to the oblivious transfer functionality $\mathcal{F}_{\text{OT}}$, assuming only the existence of a pseudo-random generator (or, equivalently, under the OWF assumption [HILL99]). In other words, under the OWF assumption, Yao's Garbled Circuit construction shows that in the PPT setting, $\mathcal{F}_{\text{OT}}$ is complete for

[12] [IL89] shows that the existence of a private-key protocol for agreeing on a longer key implies the OWF assumption. On the other hand, the sh-OT assumption yields a key agreement protocol (which does not even use a private-key).

[13] The proceedings version of [Yao86] does not include a description of this protocol; later works described this construction based on Yao's presentations. Detailed descriptions and proofs of security of various versions of this construction appear in [Rog91, TX03, AIK06, LP09].

passive-security.[14] Further, under the sh-OT assumption, this shows that all SFE functionalities are passive-trivial in the PPT setting.

For the standalone security case, Goldreich, Micali and Wigderson [GMW87] showed that all bounded functionalities are standalone-trivial if the sh-OT assumption holds.[15] These results, also implied by the later result of [Kil88], can be summarized as follows.

Theorem 4 *All bounded functionalities are passive-trivial as well as standalone-trivial in the PPT setting, if the sh-OT assumption holds.*

The sh-OT assumption is also *necessary*, because $\mathcal{F}_{\text{OT}}$ being passive-trivial is the sh-OT assumption, and if $\mathcal{F}_{\text{OT}}$ is standalone-trivial it can be seen that the same protocol would establish it to be passive-trivial as well.[16] The usage of sh-OT assumption as a hardness assumption in Theorem 4 can be made black-box without loss of generality [HIK+11].

It remains open how the sets of passive-trivial and standalone-trivial functionalities change with the various assumptions made in the PPT setting. In Section 5.2 we discuss this for finite 2-party deterministic SFE functionalities. In brief, we know that under the OWF assumption the set of standalone-trivial functionalities does expand compared to that in the information-theoretic setting; but the set of passive-trivial functionalities in this class remains unchanged, as long as the protocols use the one-way function in a "fully blackbox manner." Further restricted to *asymmetric* SFE, neither the set of passive-trivial functionalities nor that of standalone-trivial functionalities changes, unless the sh-OT assumption holds.

A result such as Theorem 4 does not extend to UC-triviality, even under stronger computational assumptions. Indeed, in the case of finite two-party SFE functionalities, the set of UC-trivial functionalities remains identical in the PPT and information-theoretic settings (see Theorem 11).

4.2 Completeness in the PPT Setting

Under the sh-OT assumption, since all functionalities are passive- and standalone-trivial in the PPT setting (Theorem 4), all of them are passive- and standalone-complete as well. On the other hand, finite functionalities which are complete in the information-theoretic setting continue to be complete — unconditionally — in the PPT setting. It remains open to understand how the sets of passive-complete and standalone-complete functionalities change with the assumptions.

It was shown in [CLOS02] that the coin-flipping functionality $\mathcal{F}_{\text{COIN}}$ is complete for UC security under specific hardness assumptions.[17] Later, this result was shown under the sh-OT assumption [DNO10, MPR10b]. Further, [MPR10b, Ros12] proved the following (with the latter providing the extension to randomized functionalities using a "splittability" criterion).

Theorem 5 *Every (possibly reactive and randomized) finite 2-party functionality that is not UC-trivial in the information-theoretic setting is UC-complete in the PPT setting under the sh-OT assumption.*

In Section 5.2 we discuss further results, which relate to two-party functionalities. In particular, going beyond the question of triviality and completeness, one could consider the computational complexity assumptions necessary and sufficient for a reduction $\mathcal{F} \sqsubseteq_{\mathsf{ppt}} \mathcal{G}$ to hold (for various security models).

[14] As mentioned above, later results showed that oblivious transfer is complete unconditionally, first for passive-security [GMW87] and later for active-security [Kil88], for both standalone and UC-security (latter explicit in [IPS08]). Nevertheless, from a practical point of view, Yao's construction is very efficient and remains of great importance.

[15] The original result was stated in terms of the assumption that (enhanced) trapdoor permutations exist (also see [Gol04]). This assumption is used to realize the sh-OT assumption. The construction used a compiler — now called the GMW compiler — to transform a passive-secure protocol into a standalone-secure protocol using zero-knowledge proofs [GMR85] for languages in **NP** [GMW86]; such proofs can be constructed based on the OWF assumption (in turn implied by the sh-OT assumption).

[16] This is not true for all functionalities. But for $\mathcal{F}_{\text{OT}}$ and other "deviation-revealing" functionalities [PR08] a standalone-secure protocol is also passive-secure.

[17] The results of [CLOS02, DNO10] are somewhat stronger: all bounded functionalities are securely realizable in the "common random string model," in which polynomially many sessions can use the same fixed string. Further, the protocol of [CLOS02] is secure against adaptive adversaries.

5 The Universe of Two Party Functionalities

We have a much deeper understanding of the cryptographic complexity of 2-party functionalities than of general MPC functionalities. This section will focus on these results. The unified presentation here borrows from [MPR12].

For the two party case, we denote the two parties by Alice and Bob (instead of P_1 and P_2), the functions associated with a functionality by f_A, f_B (instead of f_1, f_2) and the inputs from the parties by x, y (instead of x_1, x_2). In this section we consider only *finite* SFE functionalities (see Section 2). Also, we restrict ourselves to unfair functionalities.[18] Note that a finite two-party *deterministic* SSFE functionality $\mathcal{F}(f, f)$ can be fully specified by a matrix, with rows indexed by Alice's inputs and columns indexed by Bob's inputs, and entry $f(x, y)$ at position (x, y).

Graph of an SFE Functionality. It is useful to represent a 2-party (possibly randomized) SFE functionality $\mathcal{F}$ as a (weighted) bipartite graph $\mathrm{G}(\mathcal{F})$, as follows. The set of nodes on the left are indexed by pairs of the form (x, a) for each possible input value $x \in X$ for Alice and output value a for Alice; similarly, the set of nodes on the right are indexed by (y, b) for all possible inputs $y \in Y$ and outputs b for Bob. There is an edge between (x, a) and (y, b) if $\Pr[a, b|x, y] > 0$, (i.e., there is a non-zero probability that Alice and Bob get outputs a and b respectively, when they send x and y as their respective inputs to the functionality). The weight on this edge $((x, a), (y, b))$ will be $\Pr[a, b|x, y]$, or in a normalized version, $\Pr[a, b|x, y]/(|X||Y|)$.[19]

The *connected components* of this graph are interesting to study. As we shall see below, we can use it to define the kernel of the given functionality.

Isomorphic Functionalities. It will be very useful to define "isomorphism" between SFE functionalities. The definitions presented here are variants of the definitions in [MPR09, MOPR11]. Crucial to defining this is the notion of a "local protocol" for a functionality $\mathcal{F}$ using a functionality $\mathcal{G}$ as a setup. In a local protocol, each party (deterministically) maps its input to inputs for the functionality $\mathcal{G}$, calls $\mathcal{G}$ once with that input and, based on their private input, the output obtained from $\mathcal{G}$, and possibly additional private random coins, locally computes the final output, without any other communication.

Definition 1 *We say $\mathcal{F}$ and $\mathcal{G}$ are* strongly isomorphic *to each other if there exist two local protocols $\pi_{\mathcal{F}}$ and $\pi_{\mathcal{G}}$ such that*

1. *$\pi_{\mathcal{F}}^{\mathcal{G}}$ UC-securely realizes $\mathcal{F}$ and $\pi_{\mathcal{G}}^{\mathcal{F}}$ UC-securely realizes $\mathcal{G}$.*
2. *$\pi_{\mathcal{F}}^{\mathcal{G}}$ passive-securely realizes $\mathcal{F}$ and $\pi_{\mathcal{G}}^{\mathcal{F}}$ passive-securely realizes $\mathcal{G}$.*
3. *$\mathcal{F}$ and $\mathcal{G}$ have the same input domains, and in $\pi_{\mathcal{F}}^{\mathcal{G}}$ and $\pi_{\mathcal{G}}^{\mathcal{F}}$, the parties invoke the given functionality with the same input as they get from the environment.*

$\mathcal{F}$ and $\mathcal{G}$ are said to be isomorphic *to each other if conditions 1 and 2 are satisfied. $\mathcal{F}$ and $\mathcal{G}$ are said to be* weakly isomorphic *to each other if condition 1 is satisfied.*

It is not hard to see that Condition 1 above (required by all three definitions of isomorphism) is equivalent to the (seemingly weaker) condition obtained by replacing UC-security with standalone security.

All the above notions of isomorphism are equivalence relations. Isomorphism (and hence strong isomorphism) preserves UC and standalone reducibility, as well as reducibility against passive adversaries, between functionalities. Weak isomorphism preserves UC and standalone reducibility between functionalities.

[18] For the case of fair two-party functionalities, several impossibility results, and more recently, a few positive results are known [Cle86, CI93, GHKL11, MNS09, GIM+10]. We leave these results outside the scope of this survey, except to remark that the protocols in [GHKL11, MNS09] show that *certain two-party fair functionalities reduce to OT in the information-theoretic setting.*

[19] More generally, the weight on the edge $((x, a), (y, b))$ is $\Pr[a, b|x, y]p_X(x)p_Y(y)$ where $p_X(x)p_Y(y)$ is the weight of the input pair (x, y) according to some product distribution over the inputs.

Kernel: Common Information in an SFE functionality. Given an SFE functionality $\mathcal{F}$, it is convenient to define an associated SSFE functionality as the "common information" that Alice and Bob both get from $\mathcal{F}$ [MOPR11]. The *kernel of* $\mathcal{F}$ is defined as an SSFE functionality $\mathcal{F}'$ which takes x and y from the parties, samples the outcome (a, b) according to $\mathcal{F}$, and returns to both parties the *connected component* containing the edge $((x, a), (y, b))$ in $\mathrm{G}(\mathcal{F})$ (defined above). Observe that $\mathcal{F}'$ gives the same output to both parties.

As we shall see shortly, an important property of an SFE functionality is whether or not it is (strongly) isomorphic to its kernel.

Definition 2 *A (possibly randomized) SFE functionality $\mathcal{F}$ is said to be simple if it is strongly isomorphic*[20] *to its kernel.*

This definition unifies several definitions in the literature for special cases. We note these below. Firstly, it can be shown that the above condition holds if and only if the following two equivalent combinatorial conditions hold.

- For every four nodes A_0, A_1, B_0, B_1 in $\mathrm{G}(\mathcal{F})$, with A_0, A_1 on the LHS and B_0, B_1 on the RHS, it holds that $p(A_0, B_0)p(A_1, B_1) = p(A_0, B_1)p(A_1, B_0)$, where $p(A, B)$ denotes the weight on the edge (A, B) in $\mathrm{G}(\mathcal{F})$.
- Equivalently, the weights over the edges *in each connected component* of $\mathrm{G}(\mathcal{F})$ is a "product distribution": that is, there is a weight function α on the left hand nodes, and a function β on the right hand nodes such that if A and B are in the same connected component, then $p(A, B) = \alpha(A)\beta(B)$.

It is easy to see that if the connected components are all product distributions then the functionality is strongly isomorphic to its kernel, by means of a protocol/simulation which takes a connected component and samples a node for a party conditioned on its input. To see the converse, note that just the correctness of a *strongly* local reduction from the function to its kernel implies product distribution for each connected component.

As mentioned above the definition of simple functionalities that we presented above unifies several definitions that appeared in the literature for special classes of functionalities.

- *Deterministic symmetric SFE.* The first instance where simple functionalities were identified was for the special case of deterministic symmetric SFE: in this case a functionality is *not* simple if and only if the matrix representing the function f has an "OR minor" (i.e., $\exists x_0, x_1, y_0, y_1, z$ such that $f(x_0, y_0) = f(x_0, y_1) = f(x_1, y_0) = z$ and $f(x_1, y_1) \neq z$) [Kil91].
- *Randomized symmetric SFE.* In [Kil00] this was generalized to randomized symmetric SFE functionality: in this case a functionality is not simple iff $\exists x_0, x_1, y_0, y_1, z$ such that

$$\Pr[f(x_0, y_0) = z] > 0, \text{ and } \Pr[f(x_0, y_1) = z] > 0$$
$$\Pr[f(x_0, y_0) = z] \cdot \Pr[f(x_1, y_1) = z] \neq \Pr[f(x_1, y_0) = z] \cdot \Pr[f(x_0, y_1) = z].$$

- *Randomized asymmetric SFE.* In [Kil00], the characterization of simple functionalities, specialized to the case of randomized *asymmetric* SFE too appears. Kilian gives a combinatorial condition for being non-simple, but also notes (the more intuitive characterization) that the condition does not hold (i.e., the functionality *is* simple) if and only if the functionality has a passive-secure protocol which involves a single deterministic message from Alice to Bob. Equivalently, a (possibly randomized) asymmetric SFE is simple if and only if it is strongly isomorphic to a deterministic functionality in which Bob has no input.

[20] The definition is unaltered if isomorphism, instead of strong isomorphism is used. Indeed, if $\mathcal{F}$ is not strongly isomorphic to its kernel, then by virtue of say, Theorem 6, $\mathcal{F}$ is complete, and hence $\mathcal{F}$ cannot be isomorphic to its kernel (which, being simple, is not complete). However, for *deriving* the completeness characterizations, it is convenient to use such a stricter notion of isomorphism.

- *Deterministic SFE.* Another generalization, this time to deterministic, but general (not necessarily symmetric or asymmetric) SFE, appears in [KM11]: a deterministic SFE functionality is not simple iff it has an "OT-core": i.e., there are inputs x, x' for Alice and y, y' for Bob such that $f_A(x,y) = f_A(x,y')$, $f_B(x,y) = f_B(x',y)$ and either $f_A(x',y) \neq f_A(x',y')$ or $f_B(x,y') \neq f_B(x',y')$ or both.

All these special cases of the definition of simple functionalities were identified for characterizing complete functionalities (see Theorem 6 and Theorem 7).

Redundancy-Free SFE functionality. To study security against active adversaries alone (i.e., not also against passive adversaries) it is useful to have a notion of "redundant" inputs to an SFE functionality that will never be needed by an active adversary. We will restrict ourselves to deterministic SFE functionalities.[21] Alice's input x to a 2-party deterministic SFE functionality is said to be redundant if there is an input $x' \neq x$ that *dominates* x: i.e., Alice can substitute x' for x without Bob noticing (i.e., for all inputs y of Bob, $f_B(x,y) = f_B(x',y)$) while still allowing her to calculate her correct output (i.e., there is a deterministic mapping $T_{x,x'}$ such that for all inputs y of Bob, $f_A(x,y) = T_{x,x'}(f_A(x',y))$). Similarly one can define redundant inputs for Bob. A deterministic SFE functionality is called *redundancy-free* if none of Alice's and Bob's inputs is redundant.

Definition 3 *A deterministic SFE functionality $\mathcal{F}'$ is said to be a* core *of a deterministic SFE functionality $\mathcal{F}$ if $\mathcal{F}$ and $\mathcal{F}'$ are weakly isomorphic to each other and $\mathcal{F}'$ is redundancy-free.*

For any deterministic SFE functionality, one can find a core by successively removing from its domain, one at a time, inputs that are redundant (based on the set of inputs that have not been removed yet). To see this, it is enough to verify that $\mathcal{F}$ and $\mathcal{F}'$, where the latter is obtained by removing a single redundant input from the domain of $\mathcal{F}$, are weakly isomorphic to each other; this is because weak isomorphism is transitive. A local protocol for $\mathcal{F}'$ using $\mathcal{F}$ is to simply have parties feed their input for $\mathcal{F}'$ directly to $\mathcal{F}$; this protocol is secure against active (as well as passive) adversaries because even if a corrupt player feeds to $\mathcal{F}$ the input that is not in the domain of $\mathcal{F}'$ (and hence redundant), the simulator can send an input for $\mathcal{F}'$ that dominates the adversary's input. In the other direction, a local protocol for $\mathcal{F}$ using $\mathcal{F}'$ would be to instruct parties to feed their input for $\mathcal{F}$ to $\mathcal{F}'$ unless it is the removed input x, in which case they should send to $\mathcal{F}'$ an input x' that dominates x; this protocol is easily seen to be secure against active adversaries (using a simulator which forwards to the ideal functionality $\mathcal{F}$, the input that the adversary sends to $\mathcal{F}'$ in the protocol), but *passive adversaries may indeed learn more information* than $\mathcal{F}$ would give them when they use an input that dominates their actual input. Hence we can only ensure that $\mathcal{F}$ and $\mathcal{F}'$ are weakly isomorphic to each other.

Due to the above connection, while studying active security, it is convenient to study only redundancy-free functionalities, and to understand problems relating to functionalities with redundancy, relate them to a redundancy-free functionality that is weakly isomorphic to it. Redundancy-free functionalities are more well-behaved with respect to how their complexity under active security relates to that under passive security. In particular, a protocol for a redundancy-free functionality that is secure against active adversaries is also secure against passive adversaries.[22]

5.1 Complexity of SFE in the Information-Theoretic Setting

As an aid for keeping track of the various levels of complexity that we shall discuss in this section, it is useful to consider the "map" of complexity classes in Figure 1. The examples shown are all deterministic 2-party SSFE functionalities that belong to various classes. These examples are used to illustrate some of the theorems in the sequel.

[21] In [MPR10b] the notion of redundant inputs is generalized to *reactive* functionalities (where it is called dominated inputs).

[22] Redundancy-free functionalities are a special case of what are called "deviation-revealing functionalities" [PR08], a notion that is defined more generally for randomized and reactive functionalities.

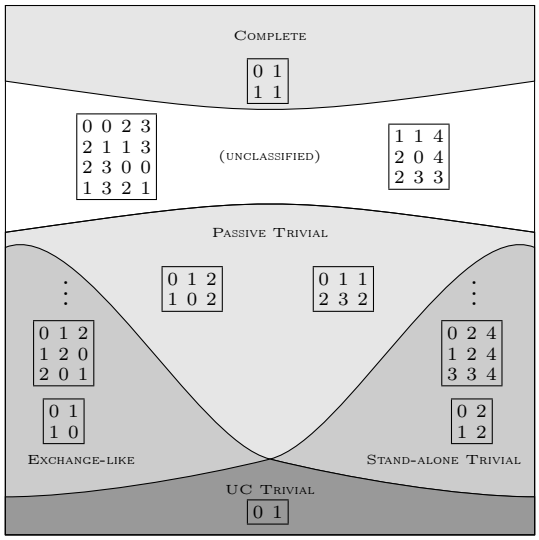


Figure 1 Some cryptographic complexity classes of deterministic 2-party SSFE functionalities with examples. The classes shown are trivial functionalities for UC security, stand-alone security against active adversaries and security against passive adversaries, the class of complete functionalities (which, for the case of SSFE functionalities, turns out to be the same for all three types of security), and exchange-like functionalities. $\mathcal{G}(f, f)$ is an exchange-like functionality if it is isomorphic to an exchange functionality $\mathcal{G}'(f_A, f_B)$ where $f_A(x, y) = y$ and $f_B(x, y) = y$ (over some finite input domains).

5.1.1 Complete SFE functionalities

The first complete functionality that was discovered was Oblivious Transfer. It was shown to be complete against passive adversaries in [GMW87, GV87, HM86], and against active adversaries in [Kil88] (and explicitly extended to UC security in [IPS08]). All subsequent completeness results build on this.

We have a full understanding of SFE functionalities that are complete under security against *passive* adversaries.

Theorem 6 *A finite (possibly randomized) 2-party SFE functionality is passive-complete in the information theoretic setting if and only if it is not simple.*

The first step towards such a characterization was taken by Kilian, for the special case of deterministic symmetric SFE [Kil91]. As mentioned before, for this case the complete functionalities are those with an OR minor. Later, Kilian extended it to the setting of randomized, *symmetric* SFE functionalities, and also for randomized *asymmetric* SFE functionalities [Kil00]. [KM11] includes the case of deterministic general SFE. Building on techniques in [Kil00], the characterization was completed (implicitly) in [MOPR11].[23]

For the case of active corruption, in standalone as well as the UC setting, a characterization of complete functionalities is known restricted to deterministic SFE.

Theorem 7 *A finite deterministic 2-party SFE functionality is standalone-complete in the information theoretic setting if and only if it has a core that is not simple.*

This was first shown for the special case of deterministic, *asymmetric* SFE (in which f_A is the constant function) by Kilian [Kil00]. In this case the condition simplifies to the following: a deterministic asymmetric SFE is simple iff it has an input y_0 for Bob such that $f_B(x, y_0)$ determines the function $f_B(x, \cdot)$ (i.e., $\forall x_0, x_1, y_1, f_B(x_0, y_0) = f_B(x_1, y_0) \Rightarrow f_B(x_0, y_1) = f_B(x_1, y_1)$). The complete characterization in Theorem 7

[23] [MOPR11] extends the protocol in [Kil00] for asymmetric SFE to show that if an SFE functionality $\mathcal{F}$ is not (strongly) isomorphic to its kernel, then it is complete for security against passive adversaries. (Though the statement in [MOPR11] is not in terms of strong isomorphism, the protocols that establish completeness of $\mathcal{F}$ only uses the condition of $\mathcal{F}$ not being *strongly* isomorphic to its kernel.) On the other hand, a functionality which is (strongly) isomorphic to its kernel is not complete, since the kernel (which is an SSFE) is itself simple and hence not complete by one of the characterizations in [Kil00].

— including the extension to UC security — is due to Kraschewski and Müller-Quade [KM11], who phrased it in terms of the presence of an OT-core (as described earlier). Extending this characterization to cover randomized SFE remains an open problem. Partial progress on this problem was made by Crépeau, Morozov and Wolf [CMW04] who showed that any non-trivial channel is standalone complete.

5.1.2 Passive Trivial SFE

We consider three classes of trivial SFE functions, depending on the type of security. The simplest 2-party functionalities are the ones which are trivial under UC security (Section 5.1.4). A much richer class of functionalities is obtained by considering triviality under information theoretic *passive security* (this section), and triviality under information theoretic *standalone active security* (Section 5.1.3. The latter two have been characterized only restricted to deterministic functionalities. We elaborate on these three low-complexity classes below. Trivial functionalities in the PPT setting (for two-party functionalities) are considered in Section 5.2.

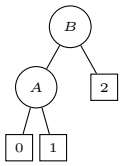
An important combinatorial structure of deterministic 2-party symmetric SFE functionalities, called decomposability, was identified in [Kus89].

Definition 4 (Decomposable Functions [Kus89]) *A function $f : X \times Y \to Z$ is* decomposable *if at least one of the following (recursively defined) conditions holds:*

- *f is a constant function (i.e., $|range(f)| = 1$). (This is the base case.)*
- *f is* row-decomposable*: i.e., there exists a partition $X = X_1 \cup \cdots \cup X_t$, with $t > 1$, $|X_i| > 0$, such that:*
  - *For each i, $f\big|_{X_i \times Y}$ is either constant or column-decomposable.*
  - *For all $y \in Y$, all i, j such that $1 \le i < j \le t$, and $x_1 \in X_i$, $x_2 \in X_j$, we have $f(x_1, y) \ne f(x_2, y)$.*
- *f is* column-decomposable*: i.e., there exists a partition $Y = Y_1 \cup \cdots \cup Y_t$, with $t > 1$, $|Y_i| > 0$, such that:*
  - *For each i, $f\big|_{X \times Y_i}$ is either constant or row-decomposable.*
  - *For all $x \in X$, all i, j such that $1 \le i < j \le t$, and $y_1 \in Y_i$, $y_2 \in Y_j$, we have $f(x, y_1) \ne f(x, y_2)$.*

Here $f\big|_{A \times B}$ denotes the restriction of f to the domain $A \times B$, where $A \subseteq X$ and $B \subseteq Y$.

As an example, consider f defined by $\begin{smallmatrix} 0 & 2 \\ 1 & 2 \end{smallmatrix}$ (i.e., with $X = Y = \{0, 1\}$, $f(0,0) = 0, f(1,0) = 1, f(0,1) = f(1,1) = 2$). A decomposition of f is shown to the right (the column decomposition node is labeled B (for Bob) and the row decomposition node is labeled A (for Alice)).



Definition 5 (Decomposable SFE) *We shall call a deterministic SSFE functionality $\mathcal{F}(f, f)$ decomposable if the function f is decomposable. We shall call a deterministic general SFE functionality $\mathcal{F}$ decomposable if it is simple and its kernel (which is an SSFE functionality to which $\mathcal{F}$ is strongly isomorphic) is decomposable.*

Note that an asymmetric SFE is decomposable if and only if it is simple (since its kernel is simply a constant functionality).

Kushilevitz [Kus89] and Beaver [Bea89] proved that a deterministic SSFE functionality is decomposable if and only if it has a *perfectly secure* protocol. Later, this characterization was extended to the case (of our standard security definition) where a secure protocol is allowed to be only statistically secure [MPR09, KMR09].

Theorem 8 *A finite deterministic 2-party SFE functionality $\mathcal{F}$ is passive-trivial in the information theoretic setting if and only if it is decomposable.*

The "if" direction of this theorem is easy. It is enough to consider symmetric SFE functionalities, since a decomposable SFE functionality is (strongly) isomorphic to a decomposable SSFE functionality (namely its kernel). If $\mathcal{F}(f,f)$ is an SSFE functionality that is decomposable, then a *canonical protocol* for evaluating f is a deterministic protocol defined inductively as follows [Kus89]:

- If f is a constant function, both parties output the output value of f, without interaction.
- If f is row-decomposable with $X = X_1 \cup \cdots \cup X_t$, then Alice announces the unique i such that her input $x \in X_i$. Then both parties run a canonical protocol for evaluating $f\big|_{X_i \times Y}$.
- If f is decomposable as $Y = Y_1 \cup \cdots \cup Y_t$, then Bob announces the unique i such that his input $y \in Y_i$. Then both parties run a canonical protocol for evaluating $f\big|_{X \times Y_i}$.

It is a simple exercise to see that a canonical protocol is a *perfectly* secure protocol for $\mathcal{F}$ against passive adversaries.

5.1.3 Standalone Trivial SFE Functionalities

To characterize standalone trivial SFE functionalities, like in the case for passive trivial SFE functionalities, it will be convenient to first restrict to SSFE functionalities, and then extend the characterization. The combinatorial characterization of standalone trivial SSFE functionalities in [MPR09, KMR09] is a further restriction on decomposability mentioned above. Below we follow [MPR09] in defining this characterization.

Decomposition strategies. Given a *uniquely* decomposable SSFE functionality $\mathcal{F}(f,f)$ for $f : X \times Y \to Z$,[24] we define a new function f^* over Alice's and Bob's *strategies* in the canonical protocol for computing $\mathcal{F}$ (using the unique decomposition), as follows. A strategy for Alice specifies, at each row-decomposition step $X' = X'_1 \cup \cdots \cup X'_t$, an index $i \in \{1, \ldots, t\}$, and similarly a strategy for Bob specifies an index at each column-decomposition step. If A, B are Alice and Bob strategies for $\mathcal{F}$, respectively, then we define $f^*(A,B)$ to be the subset $X' \times Y' \subseteq X \times Y$ obtained by "traversing" the decomposition of f according to the choices of A and B.

Essentially, the function f^* maps the strategies of (possibly corrupt) Alice and Bob in a canonical protocol for $\mathcal{F}$, to the outcome of the protocol. We define an SSFE functionality $\mathcal{F}^*(f^*, f^*)$.

Definition 6 (Saturation) *Let $\mathcal{F}$ be a uniquely decomposable SSFE functionality, and $\mathcal{F}^*$ be as defined above. We say that $\mathcal{F}$ is* saturated *if $\mathcal{F}$ and $\mathcal{F}^*$ are isomorphic to each other.*

To understand this condition further, we provide an alternate description for it. For every $x \in X$ we define an Alice-strategy A_x such that at any row decomposition step $X = X_1 \cup \cdots \cup X_t$ where $x \in X$, it chooses X_i such that $x \in X_i$. (For X such that $x \notin X$, the choice is arbitrary, say X_1.) Similarly for $y \in Y$ we define a Bob-strategy B_y. Note that in the canonical protocol, on inputs x and y, Alice and Bob traverse the decomposition of f according to the strategy (A_x, B_y), to compute the set $f^*(A_x, B_y)$ (where f is constant). If $\mathcal{F}$ is saturated, then all Alice strategies should correspond to some x that Alice can use as an input to $\mathcal{F}$. That is, for all Alice-strategies A, there exists a $x \in X$ such that for all $y \in Y$, we have $f^*(A, B_y) = f^*(A_x, B_y)$; similarly each Bob strategy B is equivalent to some B_y.

As examples, consider the SSFE functionalities $\boxed{\begin{smallmatrix}0&1\\2&3\end{smallmatrix}}$, $\boxed{\begin{smallmatrix}0&1&1\\2&3&2\end{smallmatrix}}$ and $\boxed{\begin{smallmatrix}0&1&1&0\\2&3&2&3\end{smallmatrix}}$. The first one is not uniquely decomposable (as it is both row-decomposable and column-decomposable). The second one is uniquely decomposable, but not saturated. Finally, the last one is saturated.

The above characterization — that an SSFE functionality is standalone-trivial if and only if it is saturated — can be extended to general SFE functionalities as follows.

[24] $\mathcal{F}(f,f)$ is uniquely decomposable if, in Definition 4, *exactly* one of the three conditions holds with the recursive references to decomposability replaced by *unique* decomposability, and with the partitions required to be unique (up to reordering of the parts).

Theorem 9 *A finite deterministic 2-party SFE functionality is standalone-trivial in the information theoretic setting if and only if it is weakly isomorphic to a saturated functionality.*

Note that if an SFE functionality $\mathcal{F}$ is weakly isomorphic to a saturated functionality (which is standalone trivial), then it is standalone trivial itself. To see the converse, suppose $\mathcal{F}$ is a standalone trivial SFE. Let $\mathcal{F}'$ be a core of $\mathcal{F}$. Then $\mathcal{F}'$ is simple: otherwise, by Theorem 7, $\mathcal{F}$ would be standalone complete and hence not standalone trivial. Let $\mathcal{F}''$ be the kernel of $\mathcal{F}'$. Since $\mathcal{F}$ is weakly isomorphic to $\mathcal{F}'$ (because $\mathcal{F}'$ is a core of $\mathcal{F}$) and $\mathcal{F}'$ is strongly isomorphic to $\mathcal{F}''$ (because $\mathcal{F}'$ is simple), $\mathcal{F}$ is weakly isomorphic to $\mathcal{F}''$. This implies that $\mathcal{F}''$ is standalone trivial, and being an SSFE, saturated. Thus $\mathcal{F}$ is indeed weakly isomorphic to a saturated functionality.

5.1.4 UC Trivial SFE Functionalities

Right from the introduction of the notion of UC security by Canetti [Can01], it was observed that many interesting functionalities are *not* trivial for UC security [Can01, CF01]. Shortly, these impossibility results were extended into an almost complete combinatorial characterization of all trivial deterministic 2-party SFE functionalities by Canetti et al. [CKL03, CKL06]. Here we follow the presentation in [PR08], which addresses the setting of randomized and reactive functionalities as well. At the heart of this characterization is a *behavioral property* of a functionality (as opposed to structural properties like the presence of a minor), called *splittability*.

Given a 2-party functionality $\mathcal{F}$ and a "splitter" $\mathcal{T}$ (which is a virtual party), we define another 2-party "split functionality" $\langle \mathcal{F}|\mathcal{T}|\mathcal{F}\rangle$ as follows: it internally simulates two independent instances of $\mathcal{F}$, which we call $\mathcal{F}_L$ and $\mathcal{F}_R$, and an instance of $\mathcal{T}$. $\langle \mathcal{F}|\mathcal{T}|\mathcal{F}\rangle$ lets Alice directly interact with $\mathcal{F}_L$ as Party 1 and lets Bob directly interact with $\mathcal{F}_R$ as Party 2; $\mathcal{T}$ interacts with $\mathcal{F}_L$ as Party 2 and with $\mathcal{F}_R$ as Party 1 (see Figure 2).

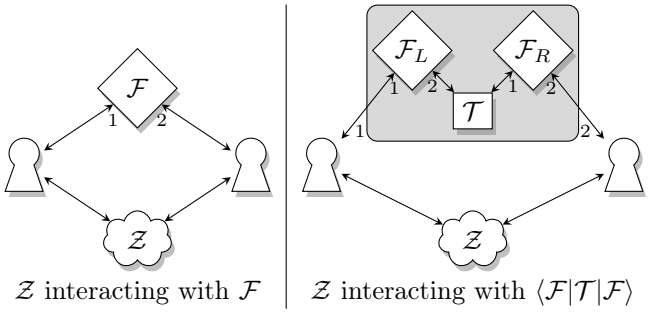


Figure 2 Environment $\mathcal{Z}$ interacting with $\mathcal{F}$ and with $\langle \mathcal{F}|\mathcal{T}|\mathcal{F}\rangle$ (shown in the shaded box).

We say $\mathcal{F}$ is splittable if there exists a machine $\mathcal{T}$ such that no environment can distinguish between interacting with $\mathcal{F}$ and interacting with $\langle \mathcal{F}|\mathcal{T}|\mathcal{F}\rangle$.

Theorem 10 *A (possibly randomized, reactive) 2-party functionality $\mathcal{F}$ is UC-trivial in the computational or information theoretic setting if and only if it is splittable.*

It is easy, using the above, to explicitly characterize which finite 2-party SFE functionalities are trivial and which are not. If a finite SFE $\mathcal{F}$ is splittable, then (upto the reordering of the two parties) there must be an input for Bob y^* which the splitter $\mathcal{T}$ can use to extract an input x' that is *equivalent* to Alice's actual input x, and from Alice's point of view, all inputs y for Bob should be equivalent. This is formalized in the following.

Theorem 11 *A finite 2-party SFE $\mathcal{F}(f_A, f_B)$ is UC-trivial in the information theoretic (or PPT) setting if and only if (after swapping Alice and Bob, if necessary) there are functions f_1, f_2, g, h over finite domains, a special input for Bob y^*, and an isomorphism R mapping a pair of strings to a single string, such that for*

all x, y, r in the input domain of f_A, f_B, and r_1, r_2 such that $r = R(r_1, r_2)$, we have $f_A(x, y, r) = f_1(x, r_1)$, $f_B(x, y, r) = f_2(g(x), y, r_2)$ and $g(x) = h(f_B(x, y, r))$.

Alternatively, a finite 2-party SFE is UC-trivial in the information theoretic (or PPT) setting if and only if it is isomorphic to a deterministic asymmetric SFE which takes input from only one party.

The second characterization above follows from the first one by considering the deterministic asymmetric SFE which takes input x from Alice and outputs $g(x)$ to Bob. All such functionalities have single-message protocols that UC securely realize them.

In the case where $\mathcal{F}$ is a deterministic SSFE functionality, $\mathcal{F}$ is UC-trivial if and only if $\mathcal{F}$ is decomposable, with a canonical protocol that exchanges only one message.

5.1.5 Functionalities of Intermediate Complexity

As shown in Figure 1, there is a gap between complete functionalities and trivial functionalities. That is, an SFE functionality that is simple is not necessarily trivial for any of our notions of security. Such a dichotomy does hold for *asymmetric* SFE (in both passive adversary and standalone active adversary settings),[25] but more generally there are SFE functionalities of intermediate complexity levels.

In [Bea89], Beaver described a symmetric SFE functionality $\begin{bmatrix} 1 & 1 & 4 \\ 2 & 0 & 4 \\ 2 & 3 & 3 \end{bmatrix}$ (called SPIRAL), in the region labeled "unclassified" in Figure 1, which is neither complete nor trivial with respect to passive security. That is, the function does not have an OR-minor, and is not decomposable. In Figure 1, we give another such symmetric SFE functionality, which we call WEAVE: $\begin{bmatrix} 0 & 0 & 2 & 3 \\ 2 & 1 & 1 & 3 \\ 2 & 3 & 0 & 0 \\ 1 & 3 & 2 & 1 \end{bmatrix}$ (in fact, any 3×4 or 4×3 minor of this function has the same property). Note that WEAVE has an output space of size 4. From a structural characterization of functions with a ternary output space in [Kre11], it follows that a 2-party deterministic SSFE functionality with an output space of size at most 3 is either complete or is trivial against passive corruption. So WEAVE is a deterministic SSFE which is neither complete nor trivial against passive corruption that is minimal in terms of the size of the output space.

5.1.6 Finer Classification

UC-trivial functionalities form the simplest class of interest in our classification. An instructive question to ask is which are *the simplest functionalities that are not UC-trivial*. Interestingly, this can indeed be answered with two specific SSFE functionalities [MPR10b]: the XOR functionality $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ (denoted by $\mathcal{F}_{\text{XOR}}$) and the "Cut-and-Choose" (or CC) functionality $\begin{bmatrix} 0 & 2 \\ 1 & 2 \end{bmatrix}$ (denoted by $\mathcal{F}_{\text{CC}}$). These are the simplest in the sense that any finite deterministic SFE functionality $\mathcal{F}$ (or even reactive functionality – see Section 6) that is not UC trivial yields at least one of these functionalities: i.e., either $\mathcal{F}_{\text{XOR}} \sqsubseteq^{\text{uc}} \mathcal{F}$ or $\mathcal{F}_{\text{CC}} \sqsubseteq^{\text{uc}} \mathcal{F}$. In Figure 1 these functionalities are shown just above the class of UC-trivial functionalities.

These two functionalities could be considered the simplest ones in two *hierarchies* of SSFE functionalities. These are hierarchies of strictly increasing cryptographic complexity in the sense that a functionality lower down in the hierarchy UC securely reduces to a functionality above it (in the computationally unbounded setting), but not vice versa. The first hierarchy is of *exchange-like* functionalities, which are deterministic SSFE functionalities isomorphic to the exchange functionality $\mathcal{F}_{\text{EXCH}}^{m,n}(f_A, f_B)$ with $f_A(x, y) = y$ and $f_B(x, y) = x$ over domains of size $|X| = m$ and $|Y| = n$. Another such hierarchy can be described in terms of decomposition trees, which can be identified with saturated SSFE functionalities (up to isomorphism, there is a unique saturated functionality with a given decomposition tree). Consider decomposition trees which are binary trees of depth d, with at least one child at every node being a leaf. $\mathcal{F}_{\text{CC}}$, with depth $d = 2$,

[25] This follows from the observation in [Kil00] that if an asymmetric SFE is simple, then it is strongly isomorphic to a deterministic functionality in which Bob has no input. Such functionalities are trivial by means of a simple protocol in which Alice sends a function of her input to Bob.

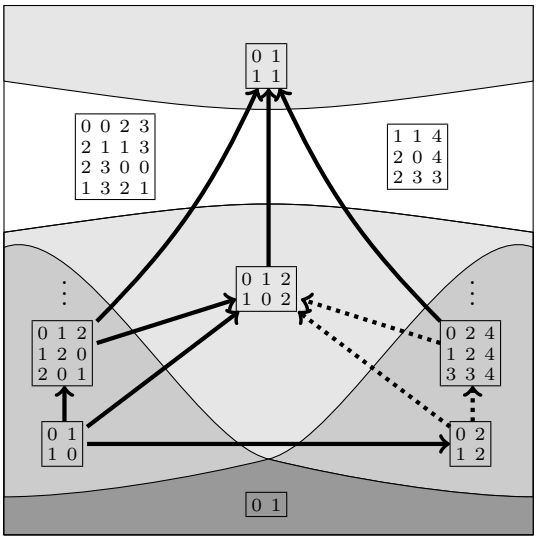


Figure 3 A solid arrow from $\mathcal{G}$ to $\mathcal{F}$ indicates that $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is equivalent to the sh-OT assumption and a dotted arrow indicates that it is equivalent to the OWF assumption.

has the smallest such tree (illustrated after Definition 4) among functionalities that are not UC trivial. A functionality with decomposition depth $d = 4$ is also shown in Figure 1.[26]

5.2 Complexity of 2-Party SFE in the Computationally Bounded Setting

Triviality. In the computationally-bounded setting, the existence of a secure protocol is likely to depend on the truth of various computational intractibility assumptions. Here we give an overview of how the class of trivial functionalities is affected by such assumptions.

Restricted to finite 2-party deterministic *asymmetric* SFE, [BMM99] showed that the sets of passive-trivial and standalone-trivial functionalities in the PPT setting are the same as those in the information-theoretic setting, *unless the sh-OT assumption holds.* However, on removing the restriction of being *asymmetric*, several functionalities which are not *standalone*-trivial in the information theoretic setting do become standalone-trivial in the PPT setting under the weaker OWF assumption. As a concrete example, the commitment functionality can be standalone-securely realized under this assumption. We note that many SFE functionalities unconditionally reduce to commitment (see the discussion in Section 6), hence these also become standalone-trivial under the sh-OT assumption.

In fact, in the case of finite two-party deterministic SFE functionalities, both for passive and standalone security, a fully blackbox use of the OWF assumption is "useful only" to securely realize the commitment functionality and those functionalities that reduce unconditionally to commitment [MMP12]. In particular, among finite two-party deterministic SFE functionalities, the set of *passive*-trivial functionalities in the PPT setting under a fully blackbox use of the OWF assumption remains the same as that in the information-theoretic setting, as commitment itself is passive-trivial.

In general, the property of UC-triviality also depends on the truth of intractibility assumptions, as it does for other security models. However, when restricted to the special case of *finite, two-party* functionalities, UC-triviality is *the same* in the computationally-bounded setting as in the information-theoretic setting (see Theorem 11).

[26] The strictness of the second hierarchy follows from a more general result in [MPR09]: a functionality with a larger tree (measured in terms of the depth of the tree) cannot be UC securely reduced to one with a smaller tree, in the computationally unbounded setting. (The proof in [MPR09] is for the case when the first functionality has a decomposition depth at least two more than that of the second functionality.) In fact, it is enough for the former functionality to be uniquely decomposable, and the latter functionality to be decomposable (and not necessarily saturated).

Reductions. More generally, the truth value of a reduction such as $\mathcal{F} \sqsubseteq_{\mathsf{ppt}} \mathcal{G}$ may likely depend on computational intractibility assumptions such as the OWF assumption or sh-OT assumption (defined in Section 4). In fact, one may view a statement such as "$\mathcal{F} \sqsubseteq_{\mathsf{ppt}} \mathcal{G}$" as a (admittedly non-standard) computational assumption in itself. By considering all "assumptions" of the form $\mathcal{F} \sqsubseteq_{\mathsf{ppt}} \mathcal{G}$, for all functionalities $\mathcal{F}$ and $\mathcal{G}$, we arrive at a set of assumptions which are the most fundamental for secure multi-party computation (at least in terms of feasibility).

This point of view of "fundamental" assumptions for multi-party computation was suggested in [MPR10a], especially for UC-secure reductions. Despite there being infinitely many pairs of functionalities $(\mathcal{F}, \mathcal{G})$, these pairs appear to induce only four distinct "assumptions" of the form $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$. That is, for all finite two-party SSFE functionalities $\mathcal{F}$ and $\mathcal{G}$ which have been classified in [MPR10a, MPR10b], one of the following holds:

1. $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is known to be unconditionally true (i.e., the reduction is achieved by an information-theoretic secure protocol).

2. $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is known to be unconditionally false. This is the case if $\mathcal{G}$ is UC-trivial but $\mathcal{F}$ is not.

3. $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is **equivalent** to the sh-OT assumption. There are two such cases:

   (a) $\mathcal{F}$ is complete and $\mathcal{G}$ is passive-trivial.

   (b) $\mathcal{F} \not\sqsubseteq^{\mathsf{uc}}_{\mathsf{it}} \mathcal{G}$ and $\mathcal{G}$ is "exchange-like" (see Figure 1). This was later partly extended to randomized SSFE: if $\mathcal{G}$ is $\mathcal{F}_{\text{COIN}}$, and $\mathcal{F} \not\sqsubseteq^{\mathsf{uc}}_{\mathsf{it}} \mathcal{G}$ then $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is equivalent to the sh-OT assumption [MOPR11, MP11].

4. $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is **equivalent** to the OWF assumption. In fact, if the above cases do not hold, and if neither $\mathcal{F}$ nor $\mathcal{G}$ fall into the "unclassified" region in Figure 1 (neither complete nor passive-trivial), then the OWF assumption *implies* that $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$. Further, it is known to be equivalent to the OWF assumption for a large subset of such pairs (and is conjectured to be so for the rest) [MPR10a].

In particular, this demonstrates the (possibly unique) fundamental nature of the OWF assumption and the sh-OT assumption to two-party computation.

Completeness. As mentioned previously in Section 4.2, all functionalities are complete for passive and standalone security in the computational setting under the sh-OT assumption (in a degenerate way, since all functionalities are also *trivial* under the same assumption). The completeness results known for computationally bounded *UC security* (also described in the previous section) are all restricted to the two-party case.

Beyond Finite Functionalities. There have been relatively fewer results about the complexity of 2-party functionalities that are not finite. Harnik et al. [HNRR06] investigated two-party *asymmetric* deterministic SFE functionalities that are not finite. They define two disjoint classes of such functionalities — row-transitive and row non-transitive[27] — which for the case of *finite* asymmetric deterministic SFE become simple and non-simple functionalities. They show that in the PPT setting, the functionalities in the first class are passive-trivial (and under the OWF assumption, standalone-trivial as well, using the GMW-compiler). The functionalities in the second class, on the other hand, are passive-complete in the PPT setting (and consequently, passive-trivial or standalone-trivial if and only if the sh-OT assumption holds).

However, this stops short of being a complete characterization of passive-trivial and passive-complete 2-party asymmetric deterministic SFE functionalities, because some of them could be neither row-transitive nor row non-transitive. A similar gap appears in [Ros12], in the case of non-finite functionalities, between functionalities which are UC-trivial and those which are UC-complete.

[27] For the functionality $\mathcal{F}(\perp, f_B)$ to be row-transitive a certain computation — computing $f_B(x_1, y)$ given only $(x_0, x_1, f_B(x_0, y))$ — has to be "easy," and to be non row-transitive it has to be "hard," but being easy and being hard are not exactly complements of each other.

6 Reactive Functionalities

Nearly all work related to cryptographic complexity has been restricted to non-reactive (that is, SFE) functionalities. However, many *positive* results for SFE immediately carry over to the case of reactive functionalities. Being able to securely compute arbitrary SFEs suffices to securely compute arbitrary reactive functionalities (see Footnote 8). The picture is less clear in security models that do not admit secure protocols for all tasks.

The ideal bit commitment functionality is a quite natural reactive functionality. A picture of commitment's cryptographic complexity arises from many protocols that use commitment. Notably, Canetti et al. [CLOS02] showed (among other things) that the commitment functionality is complete for computational UC-secure reductions under suitable computational assumptions. As was explicitly noted in [DNO10, MPR10b], the exact computational assumption corresponding to the completeness of commitment is the sh-OT assumption; further, the corresponding reductions can use the sh-OT assumption in a "fully black-box" manner [CDMW09].

In [MPR09] it was shown that the commitment functionality can be used to give a natural, alternate characterization of the class of 2-party passive-trivial, deterministic SFE. That is, they showed that a symmetric (in fact, deviation revealing) $\mathcal{F}$ belongs to this class if and only if $\mathcal{F}$ has a UC-secure reduction to the commitment functionality in the information theoretic setting.

A comprehensive treatment of arbitrary reactive functionalities (beyond commitment) first appears in [MPR10b]. There the reactive functionalities are modeled as finite automata. This model was used in [MPR10b, Ros12] to prove a zero-one law (Theorem 5) of triviality and completeness for reactive as well as non-reactive functionalities. Further, in [MPR10b], this model was used to show a structural result about finite deterministic reactive functionalities: informally, it was shown that for any (possibly reactive) $\mathcal{F}$ that is not UC-trivial, the non-triviality of $\mathcal{F}$ is either due to $\mathcal{F}$ behaving "like a non-trivial SFE" functionality, or from $\mathcal{F}$ behaving "like commitment."

Another instance of a result that extends to reactive functionalities is the characterization of UC-trivial functionalities in terms of splittability (Section 5.1.4).

But it remains the case that beyond a few such instances, the literature lacks a thorough study of the complexity of reactive functionalities.

Different complexities for reactive & non-reactive functionalities. Under very specific circumstances, reactive and non-reactive (SFE) functionalities can exhibit surprisingly different qualitative complexities. Specifically, Ishai et al. [IKLP06] considered a model in which a (polynomial-time) adversary is allowed to *either* actively corrupt a strict minority, *or* passively corrupt any number of parties (but no mixture of corruption types). In this setting, arbitrary SFE is possible, but there are reactive functionalities that cannot be securely realized. Hirt, Maurer, and Zikas [HMZ08] showed an analogous separation between SFE and reactive functionalities in the information-theoretic setting. In the simpler case where the adversary is a *mixed threshold adversary* (i.e., the adversary can actively corrupt up to t_a parties, passively corrupt up to t_p parties, and so on; see Section 3.1.2) there is no such qualitative separation between SFE and reactive functionalities.

7 Open Problems

We list a few open problems on cryptographic complexity (but omit several other important questions as the related results were not covered above.)

Privacy hierarchy. It remains open to fully characterize the different levels of the privacy hierarchy (Section 3.1.1). Importantly, it is not known which functions are n-private. A curious open problem is to find the size of the smallest output space (known to be between 4 and 16) for which Theorem 2 breaks down.

Reduction-based cryptographic complexity. The map in Figure 1 indicates a fairly detailed picture of the complexity landscape of deterministic 2-party SSFE functionalities (except for the region marked as "unclassified" in Figure 1). But extending this to the universe of functionalities that are randomized or has cardinality larger than 2 remains largely open. In particular, it remains open to extend the characterizations of passive and complete functionalities (Theorem 6, Theorem 7 and Theorem 8) to randomized 2-party SFE functionalities.

Computational Complexity and Cryptographic Complexity. The map in Figure 3 suggested the conjecture that among 2-party deterministic SSFE functionalities, any assumption of the form $\mathcal{F} \sqsubseteq^{\mathsf{uc}}_{\mathsf{ppt}} \mathcal{G}$ is equivalent to one of OWF assumption and sh-OT assumption. This conjecture remains open. On the other hand, reductions of the form $\mathcal{F} \sqsubseteq^{\mathsf{sh}}_{\mathsf{ppt}} \mathcal{G}$ and $\mathcal{F} \sqsubseteq^{\mathsf{sa}}_{\mathsf{ppt}} \mathcal{G}$ are less understood, and perhaps there are numerous "distinct" (blackbox-separated) assumptions of this form. In particular, the assumptions of the form that a functionality is passive-trivial or standalone-trivial have not been characterized; similarly the completeness of (non-trivial) functionalities have not been characterized for passive, standalone or UC reductions.

All these forms of reductions are much less understood for randomized functionalities and functionalities with cardinality larger than 2.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[Bea89] Donald Beaver. Perfect privacy for two-party protocols. In Joan Feigenbaum and Michael Merritt, editors, *Proceedings of DIMACS Workshop on Distributed Computing and Cryptography*, volume 2, pages 65–77. American Mathematical Society, 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *STOC*, pages 1–10. ACM, 1988.

[Blu81] Manuel Blum. Three applications of the oblivious transfer: Part I: Coin flipping by telephone; part II: How to exchange secrets; part III: How to send certified electronic mail. Technical report, University of California, Berkeley, 1981.

[BMM99] Amos Beimel, Tal Malkin, and Silvio Micali. The all-or-nothing nature of two-party secure computation. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 1999.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" available at the ECCC archive TR01-016. Extended abstract in FOCS 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Janos Simon, editor, *STOC*, pages 11–19. ACM, 1988.

[CDMW09] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2009.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.

[CGK94] Benny Chor, Mihály Geréb-Graus, and Eyal Kushilevitz. On the structure of the privacy hierarchy. *J. Cryptology*, 7(1):53–60, 1994.

[CGK95] Benny Chor, Mihály Geréb-Graus, and Eyal Kushilevitz. Private computations over the integers. *SIAM J. Comput.*, 24(2):376–386, 1995.

[Cha89] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 591–602. Springer, 1989.

[CI93] Richard Cleve and Russel Impagliazzo. Martingales, collective coin flipping and discrete control processes. Manuscript, 1993. http://www.cpsc.ucalgary.ca/~cleve/pubs/martingales.ps.

[CI01] Benny Chor and Yuval Ishai. On privacy and partition arguments. *Information and Computation*, 167(1):2–9, 2001.

[CK89] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy (extended abstract). In David S. Johnson, editor, *STOC*, pages 62–72. ACM, 1989.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*. Springer, 2003.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In Juris Hartmanis, editor, *STOC*, pages 364–369. ACM, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party computation. In John H. Reif, editor, *STOC*, pages 494–503. ACM, 2002.

[CMW04] Claude Crépeau, Kirill Morozov, and Stefan Wolf. Efficient unconditional oblivious transfer from almost any noisy channel. In Carlo Blundo and Stelvio Cimato, editors, *SCN*, volume 3352 of *Lecture Notes in Computer Science*, pages 47–59. Springer, 2004.

[Cré87] Claude Crépeau. Equivalence between two flavours of oblivious transfers. In Carl Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer, 1987.

[CS95] Benny Chor and Netta Shani. The privacy of dense symmetric functions. *Computational Complexity*, 5(1):43–59, 1995.

[DNO10] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. On the necessary and sufficient assumptions for UC computation. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2010.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FGMO05] Matthias Fitzi, Juan A. Garay, Ueli M. Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1998.

[FHM99] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. General adversaries in unconditional multi-party computation. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *ASIACRYPT*, volume 1716 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1999.

[FM00] Matthias Fitzi and Ueli M. Maurer. From partial consistency to global broadcast. In F. Frances Yao and Eugene M. Luks, editors, *STOC*, pages 494–503. ACM, 2000.

[GHKL11] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. ACM*, 58(6):24, 2011.

[GIM+10] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2010.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In Robert Sedgewick, editor, *STOC*, pages 291–304. ACM, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In Alfred V. Aho, editor, *STOC*, pages 218–229. ACM, 1987. See [Gol04, Chap. 7] for more details.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on http://www.wisdom.weizmann.ac.il/~oded/frag.html .

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[GV87] Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In Carl Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 1987.

[GY89] Ronald L. Graham and Andrew Chi-Chih Yao. On the improbability of reaching byzantine agreements (preliminary version). In David S. Johnson, editor, *STOC*, pages 467–478. ACM, 1989.

[HIK+11] Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions in STOC'89 and STOC'90.

[HLMR12] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Passive corruption in statistical multi-party computation. In Adam Smith, editor, *The 6th International Conference on Information Theoretic Security - ICITS 2012*, Lecture Notes in Computer Science. Springer-Verlag, 2012. Full Version available from http://eprint.iacr.org/2012/272.

[HM86] Stuart Haber and Silvio Micali. Unpublished manuscript, 1986.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptology*, 13(1):31–60, 2000. Extended abstract in Proc. 16th of ACM PODC'97.

[HMZ08] Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE : Unconditional and computational security. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[HNRR06] Danny Harnik, Moni Naor, Omer Reingold, and Alon Rosen. Completeness in two-party secure computation: A computational view. *J. Cryptology*, 19(4):521–552, 2006.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 483–500. Springer, 2006.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *Proc. 30th FOCS*, pages 230–235. IEEE, 1989.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In David S. Johnson, editor, *STOC*, pages 44–61. ACM, 1989.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, *STOC*, pages 20–31. ACM, 1988.

[Kil89] Joe Kilian. *Uses of Randomness in Algorithms and Protocols.* PhD thesis, Department of Electrical Engineering and Computer Science, Mas sachusetts Institute of Technology, 1989.

[Kil91] Joe Kilian. A general completeness theorem for two-party games. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *STOC*, pages 553–560. ACM, 1991.

[Kil00] Joe Kilian. More general completeness theorems for secure two-party computation. In F. Frances Yao and Eugene M. Luks, editors, *STOC*, pages 316–324. ACM, 2000.

[KKMO00] Joe Kilian, Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.

[KM11] Daniel Kraschewski and Jörn Müller-Quade. Completeness theorems with constructive proofs for finite deterministic 2-party functions. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2011.

[KMO94] Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. Reducibility and completeness in multiparty private computations. In *FOCS*, pages 478–489. IEEE Computer Society, 1994.

[KMR09] Robin Künzler, Jörn Müller-Quade, and Dominik Raub. Secure computability of functions in the IT setting with dishonest majority and applications to long-term security. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 238–255. Springer, 2009.

[Kre11] Gunnar Kreitz. A zero-one law for secure multi-party computation with ternary outputs. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 382–399. Springer, 2011.

[Kus89] Eyal Kushilevitz. Privacy and communication complexity. In *FOCS*, pages 416–421. IEEE, 1989.

[KY86] Anna Karlin and Andrew Chi-Chih Yao. Probabilistic lower bounds for the Byzantine generals problem. Manuscript, 1986.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[MMP12] Mohammad Mahmoody, Hemanta K. Maji, and Manoj Prabhakaran. Limits of random oracles in secure computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:65, 2012.

[MNS09] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *TCC 2009*, volume 5444 of *Lecture Notes in Computer Science*, pages 1–18. Springer, March 2009.

[MOPR11] Hemanta K. Maji, Pichayoot Ouppaphan, Manoj Prabhakaran, and Mike Rosulek. Exploring the limits of common coins using frontier analysis of protocols. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2011.

[MP11] Hemanta K. Maji and Manoj Prabhakaran. The limits of common coins: Further results. In *INDOCRYPT*, pages 344–358, 2011.

[MPR09] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Complexity of multi-party computation problems: The case of 2-party symmetric secure function evaluation. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 256–273. Springer, 2009.

[MPR10a] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Cryptographic complexity classes and computational intractability assumptions. In Andrew Chi-Chih Yao, editor, *ICS*, pages 266–289. Tsinghua University Press, 2010.

[MPR10b] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A zero-one law for cryptographic complexity with respect to computational UC security. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 595–612. Springer, 2010.

[MPR12] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A unified characterization of completeness and triviality for secure function evaluation. To Appear in the proceedings of Indocrypt, 2012.

[PR08] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2008.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In David S. Johnson, editor, *STOC*, pages 73–85. ACM, 1989.

[Rog91] Phillip Rogaway. *The Round Complexity of Secure Protocols*. PhD thesis, Massachusetts Institute of Technology, 1991.

[Ros12] Mike Rosulek. Universal composability from essentially any trusted setup. In Rei Safavi-Naini, editor, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 406–423. Springer, 2012.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.

[SRA79] Adi Shamir, R. L. Rivest, and Leonard M. Adleman. Mental poker. Technical Report LCS/TR-125, Massachusetts Institute of Technology, April 1979.

[TX03] Stephen R. Tate and Ke Xu. On garbled circuits and constant round secure function evaluation. CoPS Lab Technical Report 2003-02, University of North Texas, 2003.

[Wie83] Stephen Wiesner. Conjugate coding. *SIGACT News*, 15:78–88, January 1983.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167. IEEE Computer Society, 1986.