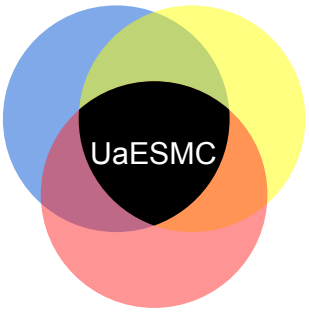

Project N°: **FP7-284731**

Project Acronym: **UaESMC**

Project Title: **Usable and Efficient Secure Multiparty Computation**

Instrument: **Specific Targeted Research Project**

Scheme: **Information & Communication Technologies**

Future and Emerging Technologies (FET-Open)

Deliverable D3.1
Potential Uses of SMC in Game Playing and Mechanism Design

Due date of deliverable: 31st January 2013

Actual submission date: 31st January 2013

Start date of the project: **1st February 2012** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **UoA**

Specific Targeted Research Project supported by the 7th Framework Programme of the EC		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Potential Uses of SMC in Game Playing and Mechanism Design

This document summarizes deliverable D3.1 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

This report constitutes a review on the interplay between secure multi-party computation (SMPC) protocols and Game Theory. Its main purpose is to provide a high-level presentation on existing techniques which incorporate SMPC protocols within game-theoretic notions and *vice versa*.

Also, this report presents potentially useful directions in which to extend the existing results on fair computation with rational players, such that a general theory on distributed Mechanism Design would result. The extensions would foremost consider the number of players, as well as different equilibria, eventually leading to a way to replace any to-compute functionality f with allocation and payment functions.

List of Authors

Yiannis Tselekounis (UoA)
Yiannis Giannakopoulos (UoA)

Contents

Chapter 1

Introduction

The application of secure multi-party computation protocols (SMPC) in real world problems is a major topic that has been extensively studied by the cryptographic community during the last decade, not only from an applied but also from a theoretical perspective. The primary objective of the **UaESMC** project is to provide tools that make such applications efficiently realizable. Towards that direction, the **UaESMC** project uses as a basis *cryptographic* and *game-theoretic* notions and builds theoretical foundations that lead to efficient applications of SMPC protocols.

The purpose of this text is to provide an overview of the world which lies on the intersection between Game Theory and Cryptography, and more specifically, we focus on the interplay between SMPC and Game Theory. The traditional cryptographic context considers two types of parties. The honest parties, who follow the prescribed protocol, and the malicious, who may deviate from the protocol arbitrarily. Now, an interesting question rises. What if we incorporate rationality on cryptographic protocols such as SMPC protocols? On the other hand, can we apply SMPC protocols in order to achieve *game-theoretic* notions? This deliverable aims to answer these questions by reviewing the state of the art in the intersection between SMPC and Game Theory.

Besides the introductory part the rest of the deliverable is structured as follows. Chapter 2 is a preliminary chapter and aims to familiarize the reader with prerequisite notions of *Cryptography* and *Game Theory*. Specifically, we define *secure multi-party computation* (SMPC) and the considered adversarial model, we present a variety of assumptions on the communication channels and we discuss the cryptographic primitives that are being employed throughout this text. Symmetrically, we define game-theoretic notions such as *games*, the notion of *Nash equilibrium* and some of its variations. Readers who are familiar with such notions may skip Chapter 2 and revisit it only when they meet notions unknown to them. In this chapter we try to avoid formal mathematical definitions and we provide brief and intuitive presentations for tools employed on subsequent chapters. Chapter 3 reviews the interconnection between Game Theory and secure multi-party computation and presents techniques that incorporate *game-theoretic* notions within SMPC and *vice versa*. Finally, Chapter 4 considers possible future directions towards meaningful and elegant constructions that incorporate both *cryptographic* and *game-theoretic* notions.

In order to provide neat and conceptual presentations we avoid counter-intuitive notational conventions in the entire text. For each protocol presented, we discuss the underlying model, the considered assumptions, we give a high-level overview of the protocol steps and in some cases we provide further intuition on the protocol properties. Therefore, in many cases technical details and proofs are intentionally omitted and we refer the reader to the original texts.

Chapter 2

Foundations

2.1 Basic notions from Cryptography and SMPC

Modern cryptography is a field which lies on the intersection between mathematics, computer science and electrical engineering, and studies techniques that prevent untrusted parties to intervene in any strictly defined interaction between honest parties so as to inflict private information disclosure or modification. Following the standard terms, cryptography incorporates well established algorithms, called *cryptographic primitives*, in order to enforce *confidentiality*, *data integrity*, *authentication* and *non-repudiation*. For the needs of the current text, we mainly focus on *confidentiality*, and specifically, *confidentiality* in the context of *secure multi-party computation* protocols (SMPC).

In multi-party computation (MPC) we consider n parties, where each party P_i, $1 \leq i \leq n$, possesses an input x_i and they all wish to jointly compute the vector $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. Then, each P_i receives only y_i. Throughout this text we mainly deal with the special case in which f outputs a single value y and all parties wish to learn that value. Now, according to the classical definition of *secure multi-party computation* (SMPC), the value of x_i is private and the parties may be of two types: the *honest-but-curious* (or *honest*), who follow the prescribed protocol but they also try to learn as much as possible about the private inputs of other parties, and the *malicious* (or *adversarial*) who may deviate from the protocol arbitrarily. Collusions between the adversarial parties are also permitted, or in other words, the model considers an adversary who may control up to k parties, for some $k \in \mathbb{N}$. A *k-secure multi-party computation* protocol (k-SMPC) ensures that any adversary who controls up to k parties cannot leak any information related to the private inputs of the honest parties, besides the leakage she gets from $f(x_1, \ldots, x_n)$. Moreover, the honest parties learn the correct output of the computation. All parties, honest or not, that participate on an SMPC protocol may be considered as *probabilistic polynomial-time* (PPT) algorithms.

Besides the considerations related to parties, SMPC incorporates assumptions related to the communication model. The most common communication assumptions are the following. The existence of a *broadcast* channel, through which someone may send messages to multiple recipients, simultaneously, and the existence of a *secure* (*private*) channel, which assumes that every communication between two end points is authenticated. More uncommon communication assumptions are the existence of *envelopes* and the *ballot box*. The former assumes that a party P_i may seal a message m of his choice into an empty *envelope* for a specific time period t, and send the *envelope* to another party P_j. Then, the construction guarantees m's integrity and that P_j may learn the value of m only after the time period exceeds. The later, provides a mechanism which randomizes the order of the envelopes. For more information on properties of *envelopes* and *ballot-boxes* we refer the reader to [25], [24] and [32].

Regarding the delivery of the output to the parties, the following cases are being considered:

1. **Output delivery**: This model guarantees that the honest parties will receive the output of the computation.

2. **Fairness**: In case fairness is enforced and at least on party learn their output, then, all the parties learn the their outputs.

3. **Correctness and privacy**: Here, the model ensures that honest parties receive only correct output values. Furthermore, the secrecy of the inputs and outputs is being preserved.

For a detailed analysis on these concepts we refer the reader to [11] and [28]. Now we describe the cryptographic tools that will be used throughout this text, together with their properties. Since those tools are employed in a black box way, we do not provide technical details and in each case we refer the reader to the corresponding texts.

2.1.1 Secret sharing

Consider a secret value s and group of parties $P_1, \ldots, P_n$. A t-out-of-n *secret sharing* is a method that: (i) splits the secret into n pieces, (ii) distributes one piece to each party and (iii) guarantees that the secret can be reconstructed only if at least t parties combine their shares. The combination of any $t-1$ or fewer shares preserves the secrecy of s.

Secret sharing can be interpreted as a special case of secure multi-party computation in which the underlying functionality is the reconstruction of the share. The role of secret sharing in constructing SMPC protocols will be made clear later on. For more details on *secret sharing* we refer the reader to [42].

2.1.2 Public key encryption

Suppose that we have two parties P_1 and P_2, such that P_1 (*sender*) wants to send a message m (*plaintext*) to P_2 (*receiver*). A *public-key encryption* scheme allows the two parties to communicate securely without the need to meet in advance and agree on any information. The receiver generates a pair of keys (pk, sk), called the public key and the private key, respectively. Then, the sender uses the public key in order to encrypt m and the receiver uses the private key to decrypt the encrypted version of m which is called *ciphertext*. The encryption of m, c, is an encoded version of it, and one can decrypt c only if she knows sk. Concretely, a *public-key encryption* scheme is a tuple of *probabilistic polynomial-time* (PPT) algorithms (Gen,Enc,Dec) such that

1. Gen receives as input a security parameter and generates the pair of keys (pk, sk).
2. Enc is the encryption algorithm. It receives the public key pk and a message m and produces the ciphertext $c \leftarrow \text{Enc}_{pk}(m)$.
3. Dec is the decryption algorithm. It receives a private key pk and a ciphertext c, and outputs a message m.

For more details see [27].

2.1.3 Oblivious transfer

Consider the following scenario. There are two parties P_1, P_2. P_1 possesses two pieces of information and P_2 wishes to receive one of the two pieces. A 1-out-of-2 oblivious transfer (OT) between the two parties guarantees that P_2 receives only the desired information, not both, while P_1 remains oblivious as to what information has delivered to P_2. The protocol ensures that any deviation will not help P_2 in learning which message was delivered while P_1 won't be able to get both pieces. There are also interesting generalizations of that protocol (1-out-of-n and k-out-of-n OT). For a detailed description of a 1-out-of-2 OT protocol we refer the reader to [13].

2.1.4 Zero-knowledge protocols

Consider again two parties, P_1 (prover) and P_2 (verifier), such that P_1 wants to convince P_2 about the validity of a statement without revealing any information besides what can be inferred from the validity of the statement. For example, suppose that P_1 wants to prove that she knows the prime factorization of n,

where $n = pq$, without revealing any information about the primes p and q. A *zero-knowledge* (ZK) protocol provides such a guarantee. In detail, its properties are the following.

1. **Completeness**: If both parties follow the protocol, and the statement is true, then the verifier will be convinced with probability equal to 1.
2. **Soundness**: A deceitful prover cannot convince an honest verifier for the validity of a false statement, except with some small (negligible) probability.
3. **Zero-knowledge**: In case the statement is true, a dishonest verifier can only learn that the statement is valid, plus all the information she can infer given the validity.

ZK protocols are interactive, i.e., the parties need to exchange information. There is also a variant of ZK, named *non-interactive zero-knowledge*. As the name suggests, these protocols do not consider interaction between the prover and the verifier. The prover sends the proof to the verifier in one step, and the verifier accepts or rejects. These constructions assume that parties share a common value, the *common reference string*, before the execution of the protocol.

Another variant of ZK is *unique zero-knowledge* (uniZK) which works in a public-key setting. The prover uses the public key in order to prove the validity of the statement and the verifier uses the secret key in order to check the correctness of the proof. Moreover, if there is only one witness that justifies the validity of the statement, then there is only one valid uniZK proof. For more details on ZK protocols and its variations we refer the reader to [18], [6], [33].

2.1.5 Message authentication codes

A message authentication code (MAC) is used to provide authenticity and integrity assurances for a specific message. Concretely, a MAC algorithm receives as input a message and a key, and outputs a MAC value. This value allows anyone who possesses the secret key to verify the authenticity and integrity of the message. So, given a message-MAC pair (m, c), in which the code c depends on key sk, the verifier checks if $\text{MAC}_{sk}(m) = c$. A secure MAC guarantees that any polynomially bounded algorithm who is unaware of sk cannot come up with a valid message-MAC pair, except with small probability. For more details see [27].

2.1.6 Commitment schemes

A commitment scheme allows a party to commit to a value v of her choice without revealing it to others. Then, she may "open" the commitment and reveal the hidden value. The scheme ensures that the party who committed to c won't persuade the others that she committed to $c' \neq c$, except with small probability. A commitment scheme executes in two phases. The commit phase, in which the party who commits chooses the value v and reveals the commitment $c(v)$. The reveal phase, in which v is revealed and checked by other parties. A secure commitment scheme guarantees (i) that anyone who observes $c(v)$ cannot infer any information about v (*hiding*), and (ii) any polynomially bounded algorithm cannot construct $c(v') = c(v)$, where $v \neq v'$, except with small probability (*binding*). For details see [5], [12] and [18].

2.1.7 Trapdoor permutations

A *trapdoor permutation* family is a tuple of *probabilistic polynomial-time* algorithms (Gen, Sample, f, Invert) such that

1. $\text{Gen}(1^n)$ is probabilistic and outputs (i, td), i.e., it produces an index i for a particular permutation f_i with trapdoor td. For any x that belongs to the domain of f_i, and for any PPT algorithm A, A cannot invert $f_i(x)$ unless she is given td. In other words, td is essential in order to efficiently invert f_i.
2. If i is the partial output of Gen, $\text{Sample}(1^n, i)$ is probabilistic and outputs an element x which is uniformly distributed over the domain D_i of f_i.

3. f is deterministic and $f(1^n, i, x)$ is just the evaluation of f_i over x. Here we assume that i and x are outputs of Gen and Sample, respectively.

4. Invert$(1^n, td, y)$ is a deterministic algorithm which inverts f_i given the trapdoor td, i.e., it outputs an x such that $f_i(x) = y$.

5. For all $n \in \mathbb{N}$, all (i, td) produced by Gen and all x that belong to the domain of f_i we have that Invert$(1^n, td, f(1^n, i, x)) = x$. Clearly, this implies correctness.

For further details on *trapdoor permutations* see [27].

2.2 Basic Notions from Game Theory

2.2.1 Normal-form games

The traditional *game-theoretic* concept considers a set of n parties, $P_1, \ldots, P_n$, a set of actions S_i and a function $u_i : S \to \mathbb{R}$, for each party P_i, where $S = S_1 \times \ldots \times S_n$. Each party P_i chooses an action $s_i \in S_i$, all players play their actions simultaneously and each one receives *utility* (*payoff*) $u_i(s_1, \ldots, s_n)$. The utility function of a party expresses the party's preference over an outcome $\mathbf{s} = (s_1, \ldots, s_n)$ (*strategy-profile*). The key idea here is that parties are *rational* and choose their moves so as to maximize their utility. Such *games* are called *normal-form games with complete information*. There are also *games* in which prior to choosing their actions, a vector of private values $\mathbf{t} = (t_1, \ldots, t_n)$ is sampled according to some distribution D, and each P_i receives t_i (normal-form games with incomplete information). Then, the parties play the game as we discussed above. The difference is that P_i's utility depends on both $\mathbf{s}$ and $\mathbf{t}$. Therefore, in games with incomplete information parties do not know the utilities of other parties, since the utility depends partially on a private value. Now, a *pure-strategy* x_i for P_i defines completely the action that will be taken by P_i in every possible situation of the game. On the other hand, a *mixed-strategy* for P_i is a distribution over the set of possible actions S_i. Hence, in the former the parties make deterministic choices in every possible game situation, while in the later, the parties act probabilistically. For details see [44] and [38].

2.2.2 Nash equilibrium

Informally, the idea of a *Nash equilibrium* is the following. Consider a set of n parties with strategy profile $\mathbf{s} = (s_1, \ldots, s_n)$, pure or mixed. Then, $\mathbf{s}$ is a *Nash equilibrium* if no party P_i has reason to deviate from the strategy s_i, given the strategies of others. In other words, if P_i chooses $s_i' \neq s_i$ and all other parties retain their initial strategies, then P_i receives utility no better than the one she would receive if she chose s_i (*weak Nash equilibrium*). If P_i's deviation leads to strictly lower *payoff*, then the equilibrium is classified as *strict Nash equilibrium*. One of the most significant results in Game Theory states that every game with a finite number of players and perfect information, in which each party can choose from finitely many pure strategies has a (mixed-strategy) Nash equilibrium ([35]).

2.2.3 Correlated equilibrium

The notion of equilibrium presented above considers strategy profiles in which parties' strategies are independent. Considering players who choose their strategies independently of each other seems to be a quite natural approach. Nevertheless, in many cases *correlated strategies* achieve higher expected payoffs. In order to incorporate correlated profiles in games, a trusted party M, called *mediator*, has to be introduced. The mediated game goes as follows. The mediator samples a correlated strategy profile $\mathbf{s} = (s_1, \ldots, s_n)$ according to some known distribution $\mathcal{M}$, and privately recommends the action s_i to party P_i. Then, the parties play as before by choosing an action s_i' from S_i. The recommended correlated strategy profile $\mathbf{s}$ is a *correlated equilibrium* if each party P_i chooses $s_i' = s_i$, i.e., if all parties have an incentive to follow the actions proposed by the mediator. For a detailed presentation we refer the reader to [38].

2.2.4 Coalition resilient equilibria

Up to now, we have considered parties who act individually in order to maximize their personal profit. A more general approach is to allow parties to cooperate with each other by forming coalitions. According to this setting, the members of a coalition coordinate their actions so as to maximize their total utility. A strategy profile $\mathbf{s}$ is a *k-resilient Nash equilibrium* if for any coalition of size at most k, no member of the coalition improves her utility by choosing a different strategy from the one that is being indicated by $\mathbf{s}$. For more details see [11].

2.2.5 Computational Nash equilibrium

In order to study the interconnection between Game Theory and Cryptography one has to adapt *game-theoretic* notions, such as *games* and *Nash equilibria*, to the cryptographic setting. According to that setting a *computational game* is a game as described above which also satisfies the following assumptions: (i) the parties are *probabilistic polynomial-time* algorithms, and (ii) the utility functions can be computed in polynomial time. Now, let $\mathbf{s} = (s_1, \ldots, s_n)$ be a strategy profile, where s_i denotes P_i's choice. Then $\mathbf{s}$ is a *computational Nash equilibrium* if for every P_i, every possible action $s_i' \neq s_i$ and regardless of what any other players do, deviating from s_i by choosing s_i' results in at most a negligible increase on the utility of P_i. For a formal definition see [11], and [10].

2.2.6 Extensive form games & cheap talk

Normal form games are special case of *extensive form* games. The main difference between those types is that in the former the parties act in one step, while in the latter each party may be active for more than one round, and consequently, her utility depends on the actions taken during the entire execution of the game. A special case of *extensive form* games are those that incorporate the so-called *cheap-talk phase*, prior to the execution of the original game. During the *cheap-talk phase* the parties execute some protocol by exchanging messages through channels that achieve certain properties. After the completion of the cheap-talk phase, the original game begins. The parties' interaction during the cheap-talk phase does not affect their utility, that's why this phase is called "cheap talk". The parties' utilities depend only on the actions taken during the execution of the original game.

2.2.7 Dominated strategies & iterated deletion

A strategy s is *weakly dominated* by s' if there exists an opponent's strategy profile for which s gives lower payoff than s', while in all other cases s' yields payoff no worse than the one that results when choosing s. In the same way, s is *strictly dominated* by s' if, regardless of what any other players do, choosing s yields a worse payoff than choosing s'. Now, as its name suggests, *iterated deletion of dominated strategies* is a procedure which iteratively removes dominated strategies. Clearly, this procedure eliminates strategies that no rational player would ever follow. Regarding its "strict" version, if after the iterative deletion of *strictly dominated* strategies there remains only one strategy for each party, then the corresponding strategy profile is the only *Nash equilibrium*. On the other hand, the elimination of *weakly dominated* strategies may eliminate some *Nash equilibria*, and clearly, the resulting equilibrium may not be unique. In the context of this text, we are primarily interested in *Nash equilibria* that survive iterated deletion of weakly dominated strategies. For more details see [11].

2.2.8 Non-cooperatively computable functions

We informally present the notion of *non-cooperatively computable* functions by considering the following game. A set of parties, where each one of them possess a private input, needs to evaluate a public function f on those inputs. Each party sends the input to a trusted entity through a private channel, the trusted entity evaluates f on the given inputs and returns the output to the parties. Here we assume that parties are *rational*, and primarily prefer getting the output of the function than not getting, and secondly, they

prefer that other parties receive an incorrect output. In such a setting, if parties have clear incentive to submit their true inputs, then f is a *non-cooperatively computable* (NCC) function. For example, the XOR function which receives n bits and outputs the exclusive or of those bits, is not an NCC function. In order to see this, consider the following scenario. Assuming that all other parties consign their true inputs and party P_i lies about her input, the trusted entity computes the XOR on the given inputs and returns the output y. Then, P_i simply computes $\neg y$ and is the only party who learns the correct output. Clearly, P_i has an incentive to lie since she can reconstruct the correct function output. On the other hand, the majority function is an NCC function since a party who consigns a false input will not be able to reconstruct the correct value. For more details on NCC functions we refer the reader to [43].

Chapter 3

Overview of Existing Work

3.1 Using Game Theory within SMPC

As we have already discussed in previous chapters, SMPC considers a set of n parties $P_1, \ldots, P_n$, where each party P_i possesses a private input x_i, $1 \leq i \leq n$, and they all wish to jointly compute and learn $y = f(x_1, \ldots, x_n)$, under the condition that each P_i will not learn any information related to the private inputs of the other parties. Moreover, each party may be honest or malicious. On the other hand, the *game-theoretic* approach of SMPC considers the case where some, or all of the parties are *rational* and follow the protocol only when they have an incentive to do so. Therefore, for each rational party P_i, we introduce a utility function u_i, which depends on the security parameter k, the transcript of the SMPC protocol and apparently on the choices made by P_i during the execution. In each protocol round, each rational party P_i acts so as to maximize u_i, with respect to the actions of the other parties, and the initial protocol has been reduced to a multi-round game for which we try to achieve different kinds of equilibria. The major point here is how to choose the utility function, or the class of utility functions so that *rationality* with respect to an SMPC protocol is being obtained. In the general case and out of the game-theoretic context, the following cryptographic properties seem to be a good starting point so that a utility function is being defined based on these properties ([11]):

1. **Correctness**: Correctness for a specific party P_i, indicates that P_i wishes to compute the correct output of the function f.

2. **Privacy**: Privacy for a party P_i indicates that the party wishes to keep her input private during and after the completion of the protocol execution.

3. **Exclusivity**: This property refers to the party's desire to be the only one who learns the correct output of the function f.

4. **Voyeurism**: Here, the party wishes to learn as much as possible about the other parties' private inputs.

Regarding the properties listed above, most of the works that will be presented in subsequent sections build utility functions based on *correctness* and *exclusivity*, and furthermore, it is assumed that the parties prefer the former over the later. Concretely, each party P_i prefers to get the correct output even if her private input becomes public after the execution, and secondly, each P_i wishes to be the only one who learns the output of f correctly. Regardless of the choice of the utility functions, the works presented in the rest of the chapter consider a variety of assumptions with respect to the communication model, such as *secure channels*, *broadcast channels*, *envelopes* and *ballot boxes* which have been mentioned in the preceding chapter. In the following sections we give a high-level overview on some major results in *rational* SMPC (RSMPC).

3.1.1 RSMPC based on rational secret sharing

A quite general technique is to construct RSMPC protocols by using rational secret sharing (RSS) as the main building block. In other words, the game-theoretic ideas presented previously are being applied on the underlying RSS protocol. In this section we give a high-level overview of such protocols, but before doing so, we briefly present two impossibility results.

On the impossibility of deterministic RSS and RSMPC. In [22] the authors present impossibility results regarding RSS and RSMPC in which the underlying functionality corresponds to fair exchange of secrets. The utility function considered is the one we described in the preceding paragraph, i.e., the parties primarily prefer *correctness*, and secondarily, they prefer *exclusivity*. Regarding the communication assumptions, the authors assume the existence of private-broadcast channels. Concretely, each party P_i possesses an additive secret share x_i with respect to a secret $x = \sum_1^n x_i$ and the impossibility result states that there is no deterministic protocol in each round of which the parties behave rationally and after its completion the protocol outputs the reconstructed x. Informally, the impossibility result goes as follows. Suppose the existence of a protocol which reconstructs x and assume that we are in the final protocol round in which the parties have to broadcast the shares computed in the preceding round. The core of the proof lies in the fact that sending out the output share cannot increase the utility of the party, and furthermore, the party's utility might be decreased if another party learns x (recall that *exclusivity* matters). In other words, sending out the output share in the final round is weakly dominated by the choice of not sending out the share. In order to see this, consider a party P_i with share x_i. Then, the protocol meets the following intuitive properties.

1. Since there is a broadcast channel, all the final shares are being sent simultaneously, and P_i's decision on sending out the share does not affect the decisions of the other parties.

2. The correctness of the reconstructed secret share is not being affected by P_i's decision to send out the share or not.

3. If P_i decides not to send out his share, then he may prevent other parties from reconstructing x. Furthermore, if a sufficient number of parties decide to send out their shares, then P_i might be the only one who reconstructs x. Therefore, P_i might obtain *exclusivety*.

The above properties constitute an informal justification of the fact that there is no incentive for the rational parties to follow the protocol, and the impossibility follows. [22] also gives impossibility results for any protocol, deterministic or not, and for the case of two parties, but as we will discuss later, a slight modification in the model leads to positive results.

Leaving behind the impossibility results, [22] presents randomized constructions for RSS and RSMPC. The key idea of the RSS protocol is the following. Each protocol round may be a test round in which the output returned to the parties is useless or it may be the final round in which the protocol outputs the reconstructed secret share. Furthermore, the parties do not know if the current round is a test round or not, and if a party deviates from the protocol on a specific round, the other parties abort. Informally, and for the 3-out-of-3 case, in each protocol round

1. A dealer sends to each party a new share of the secret input.

2. Each party flips a coin c_i which is equal to 1 with probability α. For more information regarding the value α we refer the reader to [22].

3. The parties execute a normal SMPC protocol so as to compute $c^* = \bigoplus c_i$.

4. Party P_i broadcasts its share if $c^* = c_i = 1$. If all parties send out their shares the secret is being reconstructed and the protocol terminates. If $c^* = 1$ and we have 0 or 2 shares, or if the parties do not follow the SMPC protocol of the previous step, the protocol aborts.

The above protocol gives a Nash equilibrium that survives iterated deletion of weakly dominated strategies and the case for $n > 3$ is similar. For further details we refer the reader to [22].

The RSMPC protocol proposed in [22] is based on the aforementioned RSS and the construction presented in [17]. Informally, the protocol goes as follows.

1. Each party P_i computes an n-out-of-n secret share for her private value x_i and sends out a share to each one of the $n-1$ parties.

2. The parties communicate so as to simulate the execution of a circuit which computes the value of f given the secret shares of the private inputs. In other words, the protocol simulates a circuit with gates that take as input secret shares of the real values and output the corresponding secret shares of the outcome. After the normal termination of this step, each P_i holds a share of the value $f(x_1, \ldots, x_n)$.

3. In the final step, the parties execute the RSMPC protocol presented above so as to reconstruct the value of $f(x_1, \ldots, x_n)$.

Here, the authors assume that f is a *non-cooperatively computable* (NCC) function, and therefore, the parties have clear incentive to provide their real private inputs.

An improvement of [22]. The work presented in [20] considers the same utility functions as in [22]. Moreover, regarding the communication assumptions they assume the existence of *simultaneous broadcast* channels. The RSS protocols presented in [22] and [20] are quite similar. Their main difference is that in each iteration the former allows the parties to probabilistically choose between two actions, i.e., to send their share or not, while in the later the parties are forced to send the shares that were given to them by the dealer at the beginning of the round. In [20] the dealer is the one who makes the probabilistic choice and sends out shares of the private input $s \in S$ with probability β, where S is a strict subset of a finite field $\mathbb{F}$, or sends the shares of an arbitrary value $\hat{s} \in \mathbb{F} \backslash S$ with probability $1-\delta$. Initially, each party holds the variable F with value equal to 1 which indicates if malicious behaviour has been detected, and each protocol round considers the following steps:

The RSS protocol round of [20]:

1. With probability δ the dealer computes a t-out-of-n share of the private value $s \in S$ and with probability $1-\delta$ she computes a t-out-of-n random share of an arbitrary element $\hat{s} \in \mathbb{F} \backslash S$. Then, she sends out the shares.

2. If $F = 1$, each party broadcasts the value received by the dealer. Otherwise, do nothing.

3. If the parties broadcast at least t shares, a value s' is being reconstructed and if $s' \in S$, then $s' = s$ and the protocol terminates.

4. If all parties broadcast their shares and the reconstructed value is $s' \in \mathbb{F} \backslash s$, then the secret is invalid and the protocol proceeds to the next iteration.

5. Otherwise, set $F = 0$ and proceed to the next iteration.

The modification on the dealer's behavior circumvents the impossibility presented in [22] for the case where $n = 2$. Note, that even after the detection of unwanted behaviour the protocol does not terminate. Nevertheless, the "honest" parties do nothing for the rest of the execution. [20] also provides an inefficient, as it is being characterized by the authors, RSS protocol which does not depend on the existence of the dealer. The above protocol is a Nash equilibrium for t-out-of-n secret sharing that survives iterated deletion of weakly dominated strategies. For further details we refer the reader to [20].

Now, as in [22], the above protocol constitutes the main building block for the following RSMPC protocol.

The RSMPC protocol of [20]:

1. The parties execute an SMPC protocol so as to compute the following function f': on input $x_1, \ldots, x_n$ generate a t-out-of-n sharing, $(s_1, \ldots, s_n)$, for $y = f(x_1, \ldots, x_n)$ and give s_i to P_i. If a party aborts, then the entire protocol aborts and the secret value cannot be reconstructed.

2. The parties compute the following functionality f'': it takes as input $(s_1, \ldots, s_n)$ and with probability β computes a random sharing, $(s'_1, \ldots, s'_n)$, of the value y, and with probability $1 - \beta$ computes a random share for some predetermined value that does not belong in the range of f. Each P_i receives s'_i and if a party aborts, then the entire protocol aborts and the secret value cannot be reconstructed.

3. The parties simultaneously broadcast the s'_i and reconstruct the corresponding value.

   (a) If some party did not broadcast her share, then all other parties terminate the protocol.

   (b) If the reconstructed value is in the range of f then it is equal to y and the parties learn the output of the function.

   (c) In any other case the protocol proceeds to the next iteration.

As in [22], [20] assumes that f is an NCC function. For further details regarding the proofs and the protocol construction we refer the reader to [20].

***Coalition-resilient* secret sharing and SMPC.** In [29] the authors consider utility functions according to which the parties prefer getting the protocol output to not getting it (*correctness*). Regarding the communication assumptions, they assume the existence of *simultaneous broadcast* channels. The new twist in this work is that (*i*) the model allows parties to form coalitions of size k, (*ii*) they assume that parties may perform polynomial time computation during each iteration in order to gain knowledge and maximize their utility according to it, and (*iii*) the protocol retains its properties even if it executes for exponentially many rounds on the size of the security parameter. The proposed protocol is a k-resilient computational Nash equilibrium.

As it is stated in [29], the previous works do not satisfy (*iii*), and to see this consider the following example. Suppose that in each iteration the parties perform polynomial time computations so as to break by exhaustive search the key of the underlying cryptographic primitive. After exponentially many rounds, say r, some party will finally find the key, and therefore, has no reason to participate in round $r + 1$. Using the same argument as in [22], the authors conclude that the parties have no incentive to participate even in the first round of the protocol.

In order to achieve property (*iii*), the authors introduce a new primitive called *meaningful/meaningless encryption*, which is a *public-key* encryption scheme with the property that some public keys produce ciphertexts that cannot be decrypted even by computationally unbounded adversaries. They call those keys *meaningless* while the others are called *meaningful*. For more details regarding *meaningful/meaningless encryption* we refer the reader to [29]. Now we give a high-level overview of the proposed RSS protocol.

The RSS protocol of [29]:

1. The dealer distributes an m-out-of-n share to the parties. She also provides the parties with two functions. Using the first function, the parties authenticate the shares broadcast by the others, and the second, allows a party to prove the authenticity of the share that she possesses.

2. As in the protocols presented previously, each protocol round is the final with probability δ. In each iteration of the protocol the following steps are being executed:

   (a) (**Key generation**) A traditional SMPC protocol is being executed, which receives no input and uses a random seed so as to generate private and public keys for a *β-meaningful-meaningless* encryption protocol. Each party is given a share of the seed, a *binding commitment* to that seed

and her corresponding public key. The public key generated is *meaningful* with probability β and the protocol terminates on that round.

(b) (**Encryption**) Each party encrypts the share of the secret and authentication information using the public key generated above. Then, she broadcasts the ciphertexts which are being validated using another traditional SMPC protocol.

(c) (**Verification**) Each party authenticates the seed's shares using the received commitments and regenerates the private key so as to determine if the iteration is meaningful. In such a case, she decrypts the ciphertexts and uses the retrieved information to authenticate the retrieved secret shares using the authentication functions that she was given. The verification step is valid if (i) each share is valid with respect to its corresponding commitment, and (ii) in a *meaningful* round the ciphertexts are valid with respect to the secret shares and the authentication information.

(d) (**Reconstruction**) If the verification step succeeds, the parties simultaneously broadcast the shares of the seed that they possess. The seed is being reconstructed and each player determines if the iteration is meaningful, and in case that it is, she decrypts the ciphertexts and reconstructs the secret value. Otherwise, the protocol proceeds to the next round.

The RSMPC protocol of [29] constitutes a combination between the above protocol and Yao's *Garbled Circuit*. Informally, a *Garbled Circuit* constitutes the encrypted version of the original circuit in which the entire computation is being executed on encrypted (*garbled*) values. The circuit receives *garbled* inputs, each circuit wire is being assigned two *garbled* strings, the first for the value 0 and the other for the value 1 and its gates contain information (gate tables) so as to produce valid *garbled* outputs. The *garbled circuit* computation reveals no information except the result of the evaluation. For more details on the construction we refer the reader to [46]. The proposed RSMPC constitutes a modification of the *rational* secret sharing protocol described above.

From RSS to RSMPC in [29]:

1. (***Garbled Circuit* creation**) In each protocol round a new *Garbled Circuit* is being created with respect to f. The gate tables and commitments for the two garbled strings which correspond to each input wire are being published in a way such that the correspondence remains secret. Moreover, each party is given an n-out-of-n share for each garbled string assigned to an input wire and commitments to all shares are published.

2. (**Obtaining *garbled* inputs**) Two garbled strings have been assigned to each input wire, one for the value 0 and one for the value 1. For all input wires, each player obtains all the shares of the garbled string of the input wire that corresponds to the value of that wire, using a 1-out-of-2 oblivious transfer protocol. Therefore, the parties receive the garbled values that correspond to their input bits without revealing any information. Besides the 1-out-of-2 OT protocol, this step incorporates a zero-knowledge proof which ensures the validity of the exchanged information. For more details see [29].

3. (**Encryption and verification**) Using the *β-meaningful-meaningless* encryption scheme the parties encrypt the *garbled versions* of their inputs. During the verification step each party decrypts the ciphertexts and obtains the garbled string for each input bit. Then, she verifies that the garbled value is valid with respect to one of the commitments that were published during the circuit creation step.

The above protocol is a *coalition-resilient computational Nash equilibrium* which retains its properties even if it executes for exponentially many rounds. Overall, it incorporates a traditional SMPC protocol, a 1-out-of-2 oblivious transfer protocol and a commitment scheme. In the last section the authors consider non-simultaneous broadcast channels in which there is only a single sender per protocol round and show how to run the above protocols under such a communication model. For proofs and more technical details we refer the reader to [29].

3.1.2 The mixed-behaviour model

In contrast to the models considered so far, [34] considers a mixed behaviour model, where some parties are rational and follow a specific strategy only if they maximize their utility function, and some others may be adversarial and act arbitrarily. The goal is the same as in the traditional SMPC with respect to the mixed-behaviour model (MMPC). Moreover, the authors assume the existence of a *synchronous broadcast channel*, and that all parties, rational and adversarial, are computationally bounded. They also allow covert channels and steganography, i.e., they permit message exchange which is undetectable by honest parties. Regarding the utility functions, and in contrast to [22], the main assumption is that the parties prefer to follow the protocol and consign the true private input.

The protocol construction incorporates the SMPC protocol presented in [17], and each party provides a *zero-knowledge* proof that all the information sent by her side are valid with respect to the prescribed protocol steps (*verifiable* SMPC). Now, let $P_1, \ldots, P_n$ be the parties that want to jointly compute $y = f(x_1, \ldots, x_n)$, where x_i is the private input of P_i. Recall that each P_i may be *rational* or *adversarial*, and the rational parties have clear incentive to follow the protocol and contribute the true inputs. A high-level description of the protocol goes as follows.

The RSMPC protocol of [34]:

1. (**Randomness initialization**) During the initialization phase the common random string which is being incorporated in the SMPC and *non-interactive zero-knowledge*, is being defined.
2. (**Input commitment**) After the completion of the input commitment phase, each party is committed to its private input x_i.
3. (**Secret sharing**) Using a verifiable SMPC protocol the parties construct with probability $\frac{1}{2}$ an m-out-of-n secret share of $y = 0$ or of $y = f(x_1, \ldots, x_n)$. Clearly, the protocol works only if $f(\mathbf{x}) \neq 0$, for all $\mathbf{x}$.
4. (**Broadcasting**) Each party broadcasts the share of y computed in the previous step, together with a *non-interactive zero-knowledge* proof of correctness.
5. (**Verification**)
   (a) If there are fewer than n correct shares, then abort.
   (b) If there are *correct* shares and $f(x_1, \ldots, x_n) \neq 0$, then the parties have obtained y.
   (c) If $f(x_1, \ldots, x_n) = 0$, go to step 3.

Overall, in every protocol round each party computes a zero-knowledge proof which guarantees that the parties' output in round q is consistent with the protocol steps, the data received in round $q-1$, the private input value and the initial randomness. Therefore, each party is able to verify the actions of the other parties (*verifiable* SMPC) and deviation from the prescribed protocol for a party leads the other parties to abort with high probability. Furthermore, the security properties of the SMPC protocol guarantee that the value of $f(x_1, \ldots, x_n)$ remains private.

The defined protocols is a Nash equilibrium that survives iterated deletion of weakly dominated strategies only if $m+1$ or more participants are rational. On the contrary, suppose that there are m rational parties. Furthermore, suppose that one of them, P_i, deviates from the prescribed protocol. If P_i chooses not to send out his share, then with probability $\frac{1}{2}$ all the other parties abort or not from the protocol. In the second case, and if $y \neq 0$, P_i learns the output of the function and her utility function is being maximized. Apparently, P_i has an incentive to deviate from the protocol. Now, if there are $m+1$ rational parties and P_i is the only one who deviates from the protocol, then if $y = 0$, the other parties will abort, otherwise, the m parties will learn the correct output of the protocol, regardless of the strategy of P_i. Clearly, there is no incentive for P_i to deviate from the protocol. For more details regarding the protocol and its security we refer the reader to [34].

3.2 Using SMPC within Game Theory

As we have already mentioned in Chapter 2, there are many games in which the existence of a trusted party (mediator) who assists the parties in choosing their strategies (*correlated equilibria*) leads to expected payoffs higher than the case where the mediator is absent (*Nash equilibria*). In this section we revise some interesting applications of SMPC within Game Theory, and more precisely, we focus on techniques that substitute the mediator with protocols that realize the mediator's functionality and they are provably secure. In the unmediated game the parties have a clear incentive to follow the prescribed protocol and they receive payoffs equal to those of the mediated game. In other words, the security guarantee is that the parties cannot distinguish between the two games, i.e., in the eyes of a player, the unmediated game is exactly the same with the mediated. Finally, for games that consider a *cheap-talk phase*, the same techniques are employed in order to securely implement the interaction between parties during that phase.

Eliminating the mediator using the *minmax* punishment. In [10] the authors consider *two player strategic games* in which there are two players and each one of them has a set of possible strategies. They manage to eliminate the mediator under the following assumptions: (*i*) they assume that parties are probabilistic polynomial time algorithms, and (*ii*) they consider games in which a *cheap talk* phase precedes the original game. As we have already discussed in previous chapters, during the *cheap talk* phase the parties may exchange messages for *free*, i.e., the interaction between parties during the *cheap talk* phase does not affect their utilities.

Now we give the basic idea of [10]. Consider a game G and a correlated equilibrium for that game. For the two party case, the role of the mediator is to sample a pair of strategies (s_1, s_1) according to a predefined distribution and give s_i to party P_i. As we have already stated, the mediator is like a trusted party who computes the pair (s_1, s_1) and sends the results to the parties. The key idea is to incorporate a cryptographic protocol that securely computes the mediator's functionality. The proposed strategy for each player would be to follow the protocol, and then, play the strategies that were indicated by that. But what if one of the parties deviates from the prescribed protocol steps? The authors consider a punishment method for those who deviate. Specifically, if a party cheats at some point, the other party plays a strategy which achieves the *minmax* payoff for the cheating player, which is the smallest payoff the one player can impose to the other. The authors prove that the proposed strategy is a *computational Nash equilibrium*. The main result of [10] is the following.

Let G be any two player strategic game and let G' be the extended game of G. If secure two-party protocols exist for non-trivial functions, then for any correlated equilibrium s of G there exists a computational Nash equilibrium σ of G, such that the payoffs for both players are the same in σ and s.

In an effort to provide efficient implementations for the underlying cryptographic protocols, the authors make the following observation. In *two player strategic games* the strategies of the parties define a pair (s_1, s_2), where s_i denotes party's i strategy, and the problem reduces in choosing randomly a pair from the set of all different strategy combinations. Therefore, a secure protocol for the *Correlated Element Selection problem* induces a solution for the games considered in [10]. Informally, the *Correlated Element Selection problem* is the following: we have two parties P_1, P_2, and a list of pairs $(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)$, and the parties need to jointly choose a random index i in a way such that P_1 learns only the value of a_i and P_2 learns only the value of b_i. The authors propose two protocols that solve the aforementioned problem. The first deals with the case where the parties are honest-but-curious and the second considers malicious parties.

The proposed protocols rely on a cryptographic primitive called *blindable encryption*, which is a *public-key encryption* scheme that satisfies the following property. If c is the encryption of the message m, then one is able to produce the encryption of $m + m'$ for some m' of her choice by only using c and the public key (*blinding*). Moreover, one may combine many blindings into one using the schemes' corresponding functionality (*combine*). As the authors suggest, one may use the ElGamal or the Goldwasser-Micali encryption schemes as building blocks for blindable encryption schemes. Also, any homomorphic encryption scheme

provides the necessary blinding functionality. We now present the proposed protocols for two parties, P and C.

Protocol for honest-but-curious parties:
Input: list of pairs $(a_i, b_i)_{i=1}^n$, public key pk.
Only P knows the secret key sk.

1. P creates a random permutation of the list, encrypts randomly each element of the permuted list and sends the resulting list to C. By the properties of the encryption scheme the permutation remains secret in the eyes of C.

2. C chooses randomly a pair of ciphertexts (c_l, d_l) from the permuted list. The decrypted elements of (c_l, d_l), denoted by (α, β), constitute the output of the algorithm. C blinds c_l with 0 and d_l with a random element r. The resulting pair is denoted by (e, f). C sends (e, f) to P.

3. Since the blinding of α with 0 produces e, P decrypts e and gets α. The decryption of f gives $\hat{\beta} = \beta + r$, and therefore, β remains hidden in the eyes of P. Then, P sends $\hat{\beta}$ to C.

4. C computes $\hat{\beta} - r$ and receives β.

The protocol outputs a random pair (α_i, β_i). Moreover, P learns no information about β other than it is implied by her own output, and symmetrically, C learns no information about α other than what is implied by her own output. In the former, the argument is information-theoretic since β is blinded by a random value r, while in the later the argument is computational.

Protocol for malicious parties:
Input: list of pairs $(a_i, b_i)_{i=1}^n$, public key pk.
Only P knows the secret key sk.

1. As in the first step of the "honest" case, P creates a random permutation of the list, encrypts randomly each element of the permuted list and sends the resulting list to C. Additionally, P uses zero-knowledge to prove that she knows the permutation and the randomness that she used in order to construct the encrypted pairs.

2. Again, C executes the instructions defined in the second step of the "honest" case, and then, she proves that she knows the randomness and the index used in order to produce (e, f).

3. P decrypts (e, f) and gets $(\alpha, \hat{\beta})$, where $\hat{\beta} = \beta + r$. Then, using zero-knowledge, P proves that $\hat{\beta}$ is indeed the decryption of f. Finally, P sends to C the permuted list of pairs (b_i, s_i), where s_i denotes the randomness used to encrypt b_i.

4. Let (b, s) be the l-th entry of the permuted list sent by P. C encrypts b using the randomness s, and checks if the result equals to d_l. If so, she outputs b.

As in the honest-but-curious case, the only information revealed to each party regarding the output of the other party, is implied by their own output. The authors prove that the above protocols securely compute the function that corresponds to the *Correlated Element Selection problem*. For more details regarding the proofs and some analysis related to the efficiency of the protocol we refer the reader to [10].

Improvements for [10]'s protocol. In [45] and [4] the authors propose some refinements for the protocol presented above, both for the honest-but-curious and the malicious settings. They are doing so by improving [10]'s protocol for the *Correlated Element Selection* problem. Specifically, the protocol of [45] is efficient with respect to both the security parameter and the required probability distribution. As in [10], the proposed protocol of [45] relies on blindable encryption schemes. [4] improves the work of [45] on the

correlated element selection problem and presents a protocol whose worst-case complexity is exponentially better. The protocol for the honest-but-curious setting relies on the homomorphic Pailier encryption scheme ([39],[8]). For the malicious setting the authors employ the Homomorphic ElGamal encryption scheme and construct a (t, n)-threshold cryptosystem, $0 < t \leq n$, in which the key is generated jointly by n parties and the encryption is performed as in every public-key encryption scheme. The main difference here is that the decryption succeeds only if at least t parties participate. For more details regarding those protocols we refer the reader to [45] and [4].

Coalition-safe cheap-talk. In the first part of [32] the authors present a completely fair SMPC protocol which is secure for any number of malicious players, based on standard assumptions on the communication model and computational assumptions. Specifically, they assume the existence of *envelopes* and they employ zero-knowledge proofs together with the construction of [17]. Although the first part of [32] is quite interesting from a cryptographic perspective, it does not incorporate any game-theoretic notions. Since we are primarily interested in the interplay between Game Theory and Cryptography, we skip the first part and by taking the SMPC protocol for granted we discuss applications of such a protocol in game-theoretic notions. Therefore, we proceed to the second part of the paper.

In the standard game-theoretic setting the steps of a mediated game with complete information are the following: (i) the mediator samples according to some distribution a vector of strategies $(s_1, \ldots, s_n)$, one for each party, and sends s_i to P_i, then, (ii) the game begins, the parties choose a strategy and they receive the corresponding payoffs. [32] allows the parties to form coalitions which, as we have already discussed, is like having an external party (algorithm $\mathcal{A}$) who controls any number of parties and imposes a strategy profile on them. Therefore, they consider the following steps.

A mediated game with coalitions:

1. The mediator samples a vector of strategies $(s_1, \ldots, s_n)$ and sends s_i to P_i.

2. $\mathcal{A}$ chooses a number of parties, and for each such party P_i, $\mathcal{A}$ requests s_i.

3. $\mathcal{A}$ imposes a strategy s_i' on each of the parties that she controls.

4. The parties choose their strategy. If a party belongs to the coalition she chooses s_i', otherwise, she chooses s_i. Finally, the parties receive their payoffs.

[32] uses the proposed fair SMPC protocol in order to implement a *cheap-talk* protocol that eliminates the mediator. Informally, let G be a game, E a correlated equilibrium for G and $CT = (s_1, \ldots, s_n)$ a protocol. Then, CT is a *cheap-talk* protocol that implements E, if a random execution with an adversary produces output distributed according to E. A *coalition-safe cheap talk* protocol guarantees that for any coalition of parties, the advantage received by the coalition during the extended game in which the parties execute the *cheap talk* protocol, can also be obtained by the same coalition when they participate in the original mediated game. We now give the main result as it is stated in Theorem 2 of [32]:

Given completely fair SMPC protocols for any efficient function f, then, for any game G and any correlated equilibrium E for G, there exists an efficient, coalition-safe, cheap-talk protocol $CT = (s_1, \ldots, s_n)$ such that the strategies in the extended game consisting of "following CT and then playing in G the finally computed recommendation" form a computational Nash equilibrium with payoffs indistinguishable from those of E.

For details concerning the SMPC construction and for an informal proof of the above statement, see [32].

3.3 Bidirectional approaches

In this section we review works that touch both direction, i.e., they incorporate *game-theoretic* notions on SMPC and *vice versa*.

Another improvement of [22]. In [1] the authors give an RSS protocol which is a k-resilient Nash equilibrium that survives iterated deletion of weakly dominated strategies. In addition, they show that the proposed protocol can be used to simulate games with mediators by games without mediators. The considered model assumes that parties prefer getting the output to not getting it (*correctness*), and regarding the communication assumptions, the authors assume the existence of private channels. Nevertheless, under the existence of specific cryptographic primitives that imply oblivious transfer and polynomially bounded adversaries, public channels are sufficient. Again, the proposed RSS protocol is used as a basis for the final RSMPC protocol and the main idea of the construction is similar to those presented in previous sections, both for the RSMPC protocol and the transition from RSMPC to RSS. Therefore, we omit the protocol details and we refer the reader to [1]. We now highlight the improvements of [1] over the work presented in [22] as they are listed in [1].

The improvements of [1] over [22]:

1. In [22] the proposed protocol is a 1-resilient Nash equilibrium, while in [1] the RSS protocol is $(n-1)$-resilient, where n is the number of parties.

2. In contrast to [22], the protocol of [1] works for $n = 2$. Recall that we also have the same improvement in [20].

3. Both [22] and [1] need to know the parties' utility functions in order to define the probability according to which a protocol round is the final round. However, in [1] they prove that if $k < n/3$, then their construction works for all choices of numerical utility, as long as the parties do not strictly prefer not learning the secret to learning the secret.

4. Finally, [1] considers a number of parties whose utilities may be unknown or nonstandard. As the authors indicate, if the number of parties is large, then there might be some "altruists" who maximize their utilities if more parties learn the secret.

5. The protocol of [1] can be used to simulate any mediated game by a game without a trusted mediator.

Concerning the substitution of the trusted mediator with an RSMPC protocol, the authors consider two different cases: (i) a punishment strategy, (ii) they replace the use of the punishment strategy by using Reed-Solomon decoding, verifiable secret sharing and the construction presented in [17]. For further technical details we refer the reader to [1].

Collusion-free protocols. In [31] the authors provide formal definitions for *collusion-free* protocols and prove the existence of such protocols under standard physical and computational assumptions. Specifically, they assume the existence of *broadcast channels* and *envelopes*, and regarding the computational assumptions, they assume the existence of *trapdoor permutations*.

As it is discussed in [31], in order to prevent a coalition from arising, there should be employed mechanisms that detect extra communication, i.e., communication which is not specified by the protocol and allows the "bad" players to coordinate their actions. Clearly, private channels do not provide such mechanisms since they would allow bad players to privately exchange any message of their choice while being undetected by honest parties. The same argument holds if the communication is based on envelopes. So, how do the authors provide collusion-free protocols under the existence of envelopes? The answer is that [31] employs envelopes only during a pre-processing stage, i.e., before the game begins. During that stage the malicious parties have no reason to coordinate their actions since they have not been provided any useful information

related to the game. The main result of [31] is the following.

"If trapdoor permutations exist, any finite, partial-information game with publicly observable actions has a collusion-free protocol whose communication channels consist of broadcast and plain envelopes."

A game has publicly observable actions if any action taken by a specific player becomes instantly known to all players. Now, in order to define collusion-free protocols [31] treats protocols as games and then considers the *ideal* and *real* implementations of those games. The ideal implementation incorporates a trusted party that is being eliminated on the real implementation. Then, collusion-freeness ensures that any collusion of parties cannot distinguish between the real and the ideal setting.

The finite games considered in [31] execute in rounds and in each round there is only one active player P, who may be honest or may collude with other parties. Moreover, the active player for round j is determined by some function $\mathsf{TURN}(j)$, which is part of the game definition. Then, a probabilistic function that depends on the game's global state and the active player's strategy is being evaluated, and outputs the new global state and a partial output for each party. Prior to the ideal execution of the game the trusted party generates the initial global state together with the randomness used to evaluate the probabilistic functions in every round. Informally, each round of the ideal game of [31] considers the following steps.

The j-th round of [31]'s ideal game:

1. Suppose that we have n parties and on round j party P_i is active, $i \in \{1, \ldots, n\}$. If P_i is honest, then she evaluates a predefined function H_i^j which depends on her local state so as to compute a strategy x_i^j for the current round. If P_i belongs to the set C of colluding parties, then the model considers an efficient adversary $\mathcal{A}_C$ who computes the strategy x_i^j on behalf of P_i. In each case, x_i^j is being sent to the trusted party.

2. The trusted party computes $(\sigma^{j+1}, y_1^j, \ldots, y_n^j) = g^j(\sigma^j, x_i^j)$, where g^j is the probabilistic function of round j, σ^j is the game's global state at round j, σ^{j+1} is the new global state and y_k^j, $1 \leq k \leq n$, is the function's partial output for player P_k.

3. The parties receive the output one-by-one in lexicographic order. Suppose that party P_i is about to receive the value y_i^j. First, the trusted party asks all the other parties if P_i should receive the partial output. If there is no objection, P_i receives y_i^j, otherwise, the trusted party informs all players that the game has been aborted.

In the real world scenario there is no trusted party and the above game is executed by a protocol Π between the parties. Informally, the protocol Π is a collusion-free realization of game G if the *real* and *ideal* worlds are indistinguishable in the eyes of any coalition of up to $n-1$ parties, where n denotes the total number of parties.

The proposed protocol, Π, consists of two subprotocols, Π_1 and Π_2. Π_1 is executed first, is probabilistic and takes no private inputs, uses broadcast channels and envelopes, and does not incorporate any part of the actual game (*game independence*). On the other hand, Π_2 is deterministic and receives private input which is the player's history from the execution of Π_1. Moreover, it ensures that honest parties can verify that the only non-deterministic choices for a player in Π_2 correspond to actions of the original game (*verifiable uniqueness*). A high-level and compact presentation of the protocol steps follows: (i) Π_1 is executed so as to generate randomness that will be used by Π_2, and (ii) the players execute Π_2, which is a traditional SMPC protocol that implements the original game and uses randomness provided by Π_1. The protocol is based on the construction of [17].

During the execution of Π_2 the honest parties need to be sure that all parties utilize randomness provided by Π_1. Therefore, each player needs to provide a zero-knowledge proof that she properly executes P_2. However, and as the authors suggest, since ZK is being implemented by a secure protocol would itself require randomness. So, the authors employ unique zero-knowledge proofs.

In the last section of [31] the authors prove that all assumptions are essential for the existence of collusion-free protocols. For more details we refer the reader to [31].

Rational secure computation in the ballot box model. In [25] the authors consider two games, Γ and G, where the former corresponds to an ideal mediated game and the later is the real game. Both in Γ and G the parties act so as to maximize their own interest. Specifically, Γ is normal-form, mediated, finite game of incomplete information, in which each party P_i may send her private input x_i to the mediator, and then, the mediator computes $y = f(x_1, \ldots, x_n)$. Each of the parties, may choose not to send her private input or she may send an incorrect input value.

Conversely, G is an extensive-form game of incomplete information, unmediated, with the same parties, private inputs and utility functions as Γ. The main difference here is that the computation takes place in many rounds during which the parties exchange messages over a predefined communication channel, so as to jointly compute $f(x_1, \ldots, x_n)$. As in Γ, a party P_i may choose not to send her input, and in both games a predefined penalty F is being imposed on P_i. Informally, the security definition states that all rational parties would not be able to distinguish between the two games. Since in Γ the computation is being performed by a trusted mediator, the security property follows.

Regarding the communication model, the main assumption of [25] is the existence of a *ballot box*, which requires the existence of envelopes. As we have already discussed in Chapter 2, under such an assumption, one may put a message m into an envelope with the following guarantees: the envelope will not leak any information related to m for a specific amount of time, and when someone unseals the envelope, the value of m is unmodified. The *ballot box* just randomizes the order of the envelopes. The resulting *ballot box* game is an *extensive form* game of incomplete information, in which the players exchange messages through envelopes. Apart from the envelopes, the model considers also super-envelopes, in which the players may put up to 5 envelopes. As in the case of the envelopes, super-envelopes preserve the secrecy and integrity of their content. Besides the *ballot box* communication assumption, a broadcasting channel is also being considered.

Informally, the main result in [25] is the following

Any mediated game of incomplete information Γ *can be rationally and securely simulated by a ballot-box game* G.

As it is stated in [25] G rationally and securely simulates Γ if the following hold:

1. For any equilibrium in Γ, of any strength, there exists an equilibrium in G that has the same strength and induces the same payoffs for every player (and *vice versa* for any equilibrium of G).

2. The privacy of the players in Γ is exactly preserved in G: no group of players (whether collaborating or not) may acquire more information about other players' inputs or outputs in G than they may in Γ.

In order to construct the proposed protocol, [25] modifies the protocol presented in [17] so as to be compatible with the *ballot-box* model. For technical details we refer the reader to [25].

Impossibility/possibility results for fair computation with rational players. In [21] the authors consider rational parties and propose a fair two-party computation protocol which is a *computational Nash equilibrium*. They assume that parties primarily prefer getting the correct output to not getting it (*correctness*), and secondly, each party prefers that the other party outputs an incorrect answer (*exclusivity*). Regarding the communication model, the authors consider standard channels. Based on these assumptions, they construct a protocol for fair two-party computation. Informally, the main result of [21] is the following: consider an *ideal world* in which there exists a trusty entity that receives inputs from the rational parties, computes the function f on the given inputs and supplies the parties with the function's output. Then, if

computing the function in the ideal world is a *strict Nash equilibrium*, i.e., if the parties have clear incentive to compute the function in the ideal world, then the proposed protocol is a *computational Nash equilibrium* for the real world in which the protocol simulates the trusted entity. The property according to which it is a *strict Nash equilibrium* for two parties to compute the value of the function f, on inputs sampled according to distribution D and utility functions as defined above, is called *incentive-compatibility* with respect to f, D and the utility functions. Clearly, [21] incorporates rationality on SMPC, and since the ideal world situation represents a mediated game with rational parties and the purpose is to eliminate the mediator, we may say that this work touches both directions.

By considering rational parties, [21], circumvents, not only the impossibility result of [7] in fair two-party computation, in which the parties may be either honest or malicious, but also the result of [3], which presents an impossibility result in fair two-party computation with rational parties. The ideas presented in [3] are close to that of [21], but in contrast to it they lead to an impossibility result. Nevertheless, and as it is stated in [21], the setting considered in [3] is not *incentive compatible*, i.e., the parties do not have incentive to compute the function even in the ideal world in which fairness is guaranteed. Briefly, in [3] the authors consider the following utility function: (i) getting the correct output while the other party does not, gives utility 1, (ii) getting incorrect output while the other party gets the correct output gives utility -1, (iii) any other case gives utility 0. Now, consider the case where the target function is the boolean XOR function. If both parties cooperate, then each one gets utility 0. On the other hand, if a party chooses to abort the protocol and makes a random guess about the output, then regardless of the other party's strategy in guessing the correct output, the two parties are correct with independent probability $1/2$ and the expected utility of the first party remains zero. Therefore, even under the existence of a trusted entity, the parties do not have clear incentive to compute the function using the trusted entity and they prefer to guess the output of the function. For further details regarding the impossibility result in fair two-party computation we refer the reader to [3].

In [21] the authors consider two parties P_0, P_1, with private inputs x_0 and x_1, respectively, and each P_i wishes to compute $f_i(x_0, x_1)$, for $i \in \{0, 1\}$. Moreover, they analyze both the *fail-stop* and the *byzantine* settings. In the former the parties may send the real input value or abort by sending the value $\perp$, while in the later, they are allowed to send $\perp$ and abort, or send an arbitrary value of their choice. In detail, the ideal world scenario of [21] considers the following game.

The ideal world game of [21]:

1. The input values x_0, x_1 are sampled according to some distribution D and x_i is given to party P_i, $i \in \{0, 1\}$.

2. Each party P_i sends an input x'_i to the trusted party, $i \in \{0, 1\}$. In the fail-stop setting x'_i may be the real input value or $\perp$, while in the byzantine setting x'_i may be $\perp$ or an arbitrary value. If a party sends the value $\perp$ indicates that she aborts the protocol.

3. If $x'_0 = \perp$ or $x'_1 = \perp$, the trusted party sends $\perp$ to both parties. Otherwise, the trusted party sends $f_0(x'_0, x'_1)$ to P_0 and $f_1(x'_0, x'_1)$ to P_1.

4. Each party outputs a value and receives utility which depends on the correctness of that output, and the correctness of the output of the other party.

The authors provide a protocol which eliminates the trusted party in the game defined above. The key idea of the protocol is similar to those presented so far, i.e., the parties do not know the protocol round at which the output of the function comes to their hands, and therefore, they choose to follow the protocol up to its termination. The parties interact with each other by exchanging two messages in each around. The maximum number of rounds is n and there exists an unknown (for the parties) round i^* with the following properties: (i) up to round $i^* - 1$ the parties exchange information irrelevant to the output of the function, (ii) for all rounds i, $i^* \leq i \leq n$, the parties exchange information that assist them to compute the output of the function. The main point here is that i^* is unknown. Therefore, if a party deviates from the protocol at

round $i < i^*$, with high probability she won't learn the correct output of the function. As we have already stated, [21] provides two protocols, one for the fail-stop setting and one for the byzantine setting. Here we give a high-level presentation for the fail-stop setting. The byzantine setting is similar and for further details see [21].

The protocol consists of two parts. The first part incorporates a traditional SMPC protocol which implements the following functionality.

The ShareGen pre-processing stage:
Input: x_0, x_1.
Main computation:

1. Chooses i^* according to a geometric distribution with some parameter $p > 0$.
2. In n iterations construct the values r_i^0, r_i^1, $i \in \{1, \ldots, n\}$, by doing the following:
   (a) If $i < i^*$, then sample r_i^0 according to distribution $W_0(x_0)$, which is the distribution from which P_0 samples a guess for the function output when the other party aborts. Symmetrically, sample r_i^1 according to $W_1(x_1)$. Therefore, if $i < i^*$, then r_i^b is P_b's guess on the function output which depends solely on her private input, $b \in \{0, 1\}$.
   (b) If $i \geq i^*$ the protocol sets r_i^b equal to the real function output of P_b, $b \in \{0, 1\}$, i.e., $r_i^0 = f_0(x_0, x_1)$ and $r_i^1 = f_1(x_0, x_1)$. Hence, if $i \geq i^*$, then r_i^b is the function output which corresponds to P_b.
3. Now the protocol constructs random secret shares for each r_i^b, $b \in \{0, 1\}$, $i \in \{1, \ldots, n\}$. Specifically, for each r_i^b the protocol generates random shares s_i^b and t_i^b such that $r_i^b = s_i^b \oplus t_i^b$.
4. Send $s_1^0, \ldots, s_n^0$, $s_1^1, \ldots, s_n^1$ to P_0 and $t_1^0, \ldots, t_n^0$, $t_1^1, \ldots, t_n^1$ to P_1.

The functionality presented above is being computed using a traditional SMPC protocol. After the first protocol stage the parties own shares of values, where some of them are useless, while others constitute shares of the function output. In each round of the main part of the protocol the parties exchange shares and reconstruct those values. Clearly, the values reconstructed in all rounds before the i^*-th round are junk, while in the i^*-th round and all the successive rounds the parties repeatedly reconstruct the desired outputs. Since the parties are unaware of the value of i^* they choose to follow the prescribed protocol. Now we present the main part of the protocol.

The protocol of [21]:

1. The parties securely compute ShareGen. Then, P_0 receives $s_1^0, \ldots, s_n^0$, $s_1^1, \ldots, s_n^1$ and P_1 receives $t_1^0, \ldots, t_n^0$, $t_1^1, \ldots, t_n^1$.
2. The protocol runs in n rounds. During the i-th round P_j reconstructs r_i^j as follows:
   (a) P_1 sends t_i^0 to P_0. Then, P_0 computes $r_i^0 = t_i^0 \oplus s_i^0$.
   (b) P_0 sends s_i^1 to P_1. Then, P_1 computes $r_i^1 = t_i^1 \oplus s_i^1$.
3. If a party aborts before the other party has computed any r_i value, then the second party outputs a value according to the distribution which is used in order to guess the output of the function. In any other case, i.e., if a party aborts and all parties have received at least one r_i value, or if the protocol terminates normally, then each party outputs the last r_i value that she received.

The authors prove that under the *incentive-compatibility* assumption the above protocol constitutes a *computational Nash equilibrium*. Then, they slightly modify the above protocols by incorporating message-authentication codes to each s_i^b and t_i^b. The resulting protocol is a *computational Nash equilibrium* for the *byzantine* setting. For more details see [21].

Verifiably secure devices. In [24] the authors present the notion of a *verifiably secure device* as a model of secure computation. Using those devices they (i) build secure multi-party computation protocols, and (ii), they show how to achieve *correlated* equilibria. The proposed solutions rely on the existence of *envelopes, super-envelopes* and *ballot-boxes*.

In order to analyze the security of the proposed construction [24] provides an analogue of the *ideal/real world* scenario, which has been employed on works who aim to eliminate the trusted mediator using cryptographic techniques. In the ideal world scenario the trusted mediator implements a desired functionality, and all the parties communicate with the mediator in order to have access to it. As always, the goal is to implement the mediator using an SMPC protocol so as to realize the *real world* scenario. In [24] the trusted entity is substituted by another entity which is able to perform *ballot* operations in a way such that these operations are verifiable by the parties that participate in the protocol. We now briefly describe the *ideal* and *real* world scenarios, and the security definition considered in [24].

The ideal world scenario of [24]:

1. Each party chooses a private input x_i of her choice and seals x_i into an envelope. Then, she gives the envelope to the trusted party T.

2. The trusted party privately opens the envelopes and privately computes $(y, y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. The authors assume that function f produces $n+1$ outputs. Each y_i will be privately delivered to P_i and y is public.

3. T publicizes y, seals each y_i to an envelope and sends the envelope to P_i.

The authors also consider a weaker ideal scenario in which the parties may abort at the first step. For further details see [24]. The purpose of [24] is to substitute the trusted party T with a verifiable party T', which performs ballot operations. The word verifiable is used to indicate that T' has the following property: each party P_i may verify that T' followed the prescribed sequence of operations. Before presenting the security definition of [24], we briefly mention some basic operations upon envelopes. Each entity that participates in a protocol that incorporates envelopes may publicly or privately open an envelope. In the former case a public value is being generated, while in the later, both a public and a private value are being generated. Therefore, the authors consider the notion of *public* and *private history*, and in order to achieve security in the proposed model, the following properties need to be satisfied.

Security in [24]:

1. The public history constitutes the concatenation of publicly generated values and the following must be true: if the public history generated is a fixed string R, then f has been evaluated correctly on the private inputs and *vice versa*.

2. For each party P_i, her private history contains the corresponding private input x_i.

3. Finally, the private history which corresponds to the verifiable device contains only a random string.

[24] expands the set of allowed operations with respect to *ballot-boxes* and gives detailed descriptions for these operations. Moreover, they define *global memory* which is the part of the device that stores the *ballots*, the public and private history of the execution. They also characterize the *global memory gm* as *feasible* if there exists a sequence of *global memories* that lead to *gm*. Now, a computer C for a function f is a ballot-box device, which receives a number of envelopes that correspond to the input x and outputs another set of envelopes that correspond to $f(x)$. The execution depends on the initial global memory of the device which constitutes a secure computer for f if the following hold.

The secure computer of [24]:

1. The device outputs envelopes that contain the value of the function on the input which was given using the input envelopes (*correctness*).

2. During the execution of the computer C on f, the public history of the execution is the only additional information that the parties gain. In other words, the parties do not learn anything about the other parties' input or output. After the termination of the execution the device contains only a random string, and therefore, the device does not learn the parties' inputs and outputs (*privacy*).

3. The device processes only information taken from the input envelopes. Moreover, all the input envelopes are being replaced by the output envelopes, i.e., any intermediate information that is being generated by the computer is temporary and cannot be a part of the final output.

The authors implement verifiable secure devices for three basic functions: (i) the *inverse* of a permutation, (ii) the composition of two permutations, and (iii) the *permutation clone function* which maps a permutation p to the pair of permutations (p, p). For more details on these constructions we refer the reader to [24].

Chapter 4

Rational Fair Computation for Mechanism Design

It is a well-known result (see the seminal paper of Cleve [7]) that fair computation is generally impossible to achieve under standard cryptographic assumptions of malicious behavior. Based on this, Asharov et al. [3] proposed a game-theoretic based model of *rationally*-behaving cryptographic parties, under which these inherent limitations could be overcome. Very recently, Groce and Katz [21] generalized this model and proved very interesting results regarding two-party fair computation with such rational players. In this section we shortly revisit the model and the main results in [21] and propose some closely related open problems that can bridge further Cryptography and Game Theory, and in particular Mechanism Design.

4.1 The Model

Let's assume two parties P_0 and P_1, each one having an *input* $x_0 \in X_0$ and $x_1 \in X_0$, respectively, drawn from some joint probability distribution D over $X_0 \times X_1$. These two parties want to compute the value of a certain deterministic function f on their inputs, where $f = (f_0, f_1) : X_0 \times X_1 \longrightarrow \{0,1\}^* \times \{0,1\}^*$. Party P_0 needs to receive the $f_0(x_0, x_1)$ component and P_1 the $f_1(x_0, x_1)$ one. There are two settings in which that could be done: an *ideal world* where there is a completely trusted third-party entity that implements the computation and the reporting of f to the players and a *real world* where this is done by the players themselves using some distributed cryptographic protocol Π to achieve this.

We model the players' (i.e. the cryptographic parties' P_0 and P_1) behavior in a game-theoretic way, by assuming that their incentives are driven by their *utilities* on every possible outcome, defining thus essentially a game[1]. So, we assume that the players are *rational* and they would either participate honestly (by reporting their *true* inputs x_0,x_1) in the execution of protocol Π or they would misreport some value $x_0' \neq x_0$ or even abort the execution $x_0 = \perp$ ($x_1' \neq x_1$ or $x_1 = \perp$ for player P_1 in an analogous way), if this is to give them higher resulting utilities.

In this model, the major question one asks is whether a certain function f, given some distributional prior D on the players' inputs, can be *fairly* computed this distributed way in the real world by a protocol Π such that it is a *Nash equilibrium* for the players to honestly participate in its execution. Groce and Katz [21] prove a surprising result: this is always possible, given the (rather necessary[2]) assumption that the players would have a *strong* incentive to participate in such a computation at the ideal world where everything is done in a centralized way by a trusted mediator.

We give here some subtle additional features of the model, and we refer the reader to the original article [21] for all technical details and precise formulations. In addition to their values x_0, x_1, the players' strategies also include a second component, W_0, W_1 respectively. These are probability distributions and

[1] For a very quick introduction in Game Theory and Mechanism design see e.g. section 2.2 of the current deliverable D3.1 or Chapter 1 in D3.2.

[2] If that's not the case, the players have no reason to want to give any kind of input anyways.

are used in case the trusted third part declines[3] to return a value for f (i.e. he reports $\perp$ to some players). In such case, party P_0 just generates an output $W_0(x_0)$ and P_1 outputs $W_1(x_1)$. In some sense thus, these probability distributions, which are *common knowledge* among the parties and the mediator, represent the "prior knowledge" that each player has about the setting and it is essentially used as an "empty threat" in case the other party does not cooperate.

4.2 Future Directions

The above discussion and results seem to provide a suitable ground fore some interesting questions and possible future directions at the intersection of Cryptography and Mechanism Design:

- First, an immediate extension of the results of Groce and Katz [21] would be to generalize them for the case of $n \geq 3$ parties. This is not trivial, since one needs to take care of how the third party mediator acts in case of only partial denials of strict subsets of the players to participate in the computation. The mechanics of the interactions in such a multi-party setting are not straightforward and we need to clearly decide on a "natural" behavior of the mediator. Also, the protocol Π proposed by the authors in order to achieve this distributed implementation includes a construction that needs to be extended in a clever way for more than two players, due to "loss of symmetry". We think that this problem of $n \geq 3$ players is the most obvious and immediate one that asks for our future attention.

- Another possible direction, proposed also by the authors themselves, is that of exploring *more strict* solution concepts for the underlying game, e.g. dominant strategies equilibria. This could be of significant importance for Mechanism Design in particular, since incentive-compatibility in dominant strategies is a predominant requirement. Furthermore, it is well-known that equilibria in dominant strategies are much more "reliable" and straightforward to explain and implement, both semantically but also computationally.

- Finally, the previous two points paired with our exposition in section 4.1 provide a very attractive setting for trying to develop some general theory for *distributed* Mechanism Design: a mechanism essentially consists of two functions, an allocation one and a payment one, computed over the input types of the players. So, can we apply the result of Groce and Katz [21] in a meaningful and "natural" way in order to provide models of distributed, secure and fair implementation of our mechanisms by the various parties themselves i.e., loosely speaking, replacing f by the allocation and payment functions?

[3] This happens if some party reports $\perp$ to the mediator.

Bibliography

[1] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 53–62, New York, NY, USA, 2006. ACM.

[2] G. Asharov and Y. Lindell. Utility dependence in correct and fair rational secret sharing. *Journal of cryptology*, 24(1):157–202, 2011.

[3] Gilad Asharov, Ran Canetti, and Carmit Hazay. Towards a game theoretic view of secure computation. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 426–445, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] Mikhail J. Atallah, Marina Blanton, Keith B. Frikken, and Jiangtao Li. Efficient correlated action selection. In *Proceedings of the 10th international conference on Financial Cryptography and Data Security*, FC'06, pages 296–310, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[6] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[7] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. ACM.

[8] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 119–136, London, UK, UK, 2001. Springer-Verlag.

[9] K. Das. *Privacy preserving distributed data mining based on multi-objective optimization and algorithmic game theory.* PhD thesis, UNIVERSITY OF MARYLAND, BALTIMORE COUNTY, 2010.

[10] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology—CRYPTO 2000*, pages 112–130. Springer, 2000.

[11] Yevgeniy Dodis and Tal Rabin. Cryptography and game theory. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 8, pages 181–206. Cambridge University Press, 2007.

[12] Shimon Even. A protocol for signing contracts. *SIGACT News*, 15(1):34–39, January 1983.

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

[14] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '02, pages 1–13, New York, NY, USA, 2002. ACM.

[15] G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. *Theory of Cryptography*, pages 419–436, 2010.

[16] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[17] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[18] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, New York, NY, USA, 2000.

[19] Oded Goldreich. *Foundations of cryptography: a primer*, volume 1. Now Publishers Inc., Hanover, MA, USA, April 2005.

[20] S. Gordon and J. Katz. Rational secret sharing, revisited. *Security and Cryptography for Networks*, pages 229–241, 2006.

[21] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 81–98, Berlin, Heidelberg, 2012. Springer-Verlag.

[22] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 623–632, New York, NY, USA, 2004. ACM.

[23] J.Y. Halpern and R. Pass. Game theory with costly computation. *Arxiv preprint arXiv:0809.0024*, 2008.

[24] S. Izmalkov, M. Lepinski, and S. Micali. Verifiably secure devices. *Theory of Cryptography*, pages 273–301, 2008.

[25] S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 585–594. IEEE, 2005.

[26] Hillol Kargupta, Kamalika Das, and Kun Liu. Multi-party, privacy-preserving distributed data mining using a game theoretic framework. In *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD 2007, pages 523–531, Berlin, Heidelberg, 2007. Springer-Verlag.

[27] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman & Hall/CRC, 2007.

[28] Jonathan Katz. Bridging game theory and cryptography: recent results and future directions. In *Proceedings of the 5th conference on Theory of cryptography*, TCC'08, pages 251–272, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] Gillat Kol and Moni Naor. Cryptography and game theory: designing protocols for exchanging information. In *Proceedings of the 5th conference on Theory of cryptography*, TCC'08, pages 320–339, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] Gillat Kol and Moni Naor. Games for exchanging information. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 423–432, New York, NY, USA, 2008. ACM.

[31] Matt Lepinksi, Silvio Micali, and abhi shelat. Collusion-free protocols. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 543–552, New York, NY, USA, 2005. ACM.

[32] Matt Lepinski, Silvio Micali, Chris Peikert, and Abhi Shelat. Completely fair sfe and coalition-safe cheap talk. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 1–10, New York, NY, USA, 2004. ACM.

[33] Matt Lepinski, Silvio Micali, and Abhi Shelat. Fair-zero knowledge. In *Proceedings of the Second international conference on Theory of Cryptography*, TCC'05, pages 245–263, Berlin, Heidelberg, 2005. Springer-Verlag.

[34] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *Proceedings of the 26th annual international conference on Advances in Cryptology*, CRYPTO'06, pages 180–197, Berlin, Heidelberg, 2006. Springer-Verlag.

[35] J. Nash. Non-cooperative games. *Annals of mathematics*, 54(2):286–295, 1951.

[36] Noam Nisan. Introduction to mechanism desing (for computer scientists). In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 9. Cambridge University Press, 2007.

[37] S. Ong, D. Parkes, A. Rosen, and S. Vadhan. Fairness with an honest minority and a rational majority. *Theory of Cryptography*, pages 36–53, 2009.

[38] M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[39] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[40] Christos H. Papadimitriou. The complexity of finding nash equilibria. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 2. Cambridge University Press, 2007.

[41] A. Shamir, R.L. Rivest, and L.M. Adleman. *Mental poker*. Laboratory for Computer Science, Massachusetts Institute of Technology, 1979.

[42] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[43] Yoav Shoham and Moshe Tennenholtz. Non-cooperative computation: boolean functions with correctness and exclusivity. *Theor. Comput. Sci.*, 343(1-2):97–113, October 2005.

[44] Éva Tardos and Vijay V. Vazirani. Basic solution concepts and computational issues. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*, chapter 1. Cambridge University Press, 2007.

[45] V. Teague. Selecting correlated random actions. In *Financial Cryptography*, pages 181–195. Springer, 2004.

[46] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.