

# Distributing Trusted Third Parties\*

Alptekin Küpçü  
Koç University, İstanbul, TURKEY  
akupcu@ku.edu.tr

April 17, 2013

## Abstract

Trusted Third Parties (TTPs) are widely employed in various scenarios for providing fairness guarantees (e.g., in fair exchange or e-commerce protocols, including secure two-party computation), for distributing secrets (e.g., in authentication or secret-sharing protocols, as well as group signatures), and for creating trust (e.g., as certificate authorities). Such wide use of TTPs, as well as the trust requirement that is put on them make them a prime target for distributed systems and cryptography research.

There are some well-known and proven solutions to the problem of distributing the trust put on TTPs: byzantine agreement or secure multi-party computation techniques can be employed to distribute the job of any TTP to multiple parties, tolerating up to half or one-third of those parties being malicious. Such techniques are not widely-employed in practice possibly due to their quadratic complexity or inter-operation requirements. This brings up the question of distributing TTPs in a much more efficient way, possibly via using autonomous agents, who do not directly communicate with each other.

In this paper, we present various known techniques for distributing the trust put on TTPs. Then, we concentrate on using multiple autonomous parties, who do not communicate with each other at all, to realize a single TTP. We discuss the role of synchrony in such attempts, and conclude with some open questions.

**Keywords:** trusted third party, fair exchange, distributing trust.

## 1 Introduction

Many current systems rely on *trusted third parties* (TTPs) for performing their operations fairly, securely, or efficiently. TTPs are crucially important in many scenarios, some of whom we will look at in detail in this paper. For example, it is proven that efficient fair exchange protocols cannot be completely fair without the help of a TTP that is mutually trusted by both of the parties performing the exchange [71]. Therefore, TTPs are employed in all state-of-the-art optimistic fair exchange and e-commerce protocols [2, 56, 57, 3, 7, 10, 33, 67]. If the TTP acts dishonestly, then the result may be unfairness against one of the parties.

Many other systems rely on TTPs. In authentication and certification systems [69, 51] the dishonesty of the TTP may break the security of the whole system. In group signature schemes

---

\*Parts of Section 4 of this article previously appeared in [55].

[26] dishonesty may result in losing privacy. In secret-sharing protocols [80, 14] a dishonest dealer may prevent the system from working at all. In some secure multi-party computation and secure two-party computation protocols that employ a TTP for providing fairness [63, 19], indeed the protocol is unfair without a TTP anyway; thus one may argue that using a dishonest TTP is no worse than not using a TTP. In some reputation-based systems and e-auction protocols TTP incentivizes honest behavior, and absence or dishonesty of TTP would mean either no or faulty work getting done, or a high-price economy [50, 79, 77, 9].

Due to the multitude of systems that rely on TTPs for various guarantees on fairness, security, or efficiency, there is a pressing need to distribute the trust put on those central entities. The usual way of distributing the trust is employing multiple parties instead of a single central TTP. Obviously, other than distributing the trust, such systems generally increase fault tolerance of the system, though not always (e.g., some systems require that all parties must act as part of the TTP function, for example by using n-out-of-n secret sharing, which actually increases the number of points of failure, thus creating a negative effect on fault tolerance [41]).

The standard mechanisms of distributing trust include Byzantine agreement [12, 16, 39, 60, 53] and secure multi-party computation [86, 45, 11, 24, 22] techniques. Given such a general tool, any TTP functionality can be distributed to multiple parties, up to half or one-third of whom may be malicious. Unfortunately, this extraordinary benefit comes with a price: communication complexity that is quadratic in terms of the number of parties employed for distributing the TTP [34]. Indeed, this is one of the reasons why TTPs are not implemented in a distributed manner in reality. Threshold cryptosystems [32, 30, 31] may also be used to distribute the job of TTP in some systems, and we believe further research in this area should be done.

On the other hand, it is conceivable to employ multiple *autonomous* parties to perform the job of a TTP [55]. Autonomous parties never directly communicate with each other, and hence can be realized by independent entities with ease. Their decisions do not depend on each other's decisions. They do not even need to be aware that other parties realizing the TTP exist. In essence, they can consider themselves as the sole TTP in the system, and base their decisions accordingly. Effectively, you can create your own TTP agent and set it running independently of others, without explicitly having an identity as part of a group as in byzantine systems. This is a very desirable property especially in peer-to-peer settings [5].

As in previous scenarios, the idea is to rely on maybe at least half of these parties being honest, thereby distributing the trust. One benefit of such an approach is that since the parties forming the TTP are autonomous, the complexity of communication among these agents is zero. The parties participating in the underlying system using the TTP (e.g., parties trying to fairly exchange their items) may have increased communication costs, since now they may need to contact more than a single agent. Just as in many other solutions, the synchrony of those agents forming the TTP will play a crucial role in such autonomous distributed TTPs, as we will discuss later.

In this paper, we shall first discuss non-autonomous solutions to distributing trust. There has been extensive work in this area, yet we can only provide an overview. Then, we will transition to employing autonomous agents in distributing trust. The main idea will be based on representing the finite state machine of the TTP and the agents employed to form that TTP, and then relating the states of the participating parties, explaining how the state of one affects the state of the other. We will then look at a single scenario in more detail: autonomous distributed TTP in optimistic fair exchange. After discussing the role of synchrony, we shall conclude with some related topics and open problems.

## 2 Cryptographic Preliminaries

We informally define many cryptographic primitives that will be used in this paper. A crypto-savvy reader may freely skip this section. Parts of these will be repeated in a scattered manner throughout the text.

**Secure multi-party computation** is a technique where  $n$  multiple independent parties may jointly compute a function  $f(x_1, \dots, x_n)$  where  $x_i$  is the input provided by party  $i$ . The security property states that no party  $j$  can learn information about others' inputs, except what he can infer using the output of the function  $f$  and its own input  $x_j$ . Secure two-party computation is just the same, but with only two parties.

**Secret sharing** techniques let a single party owning a secret value  $s$  to distribute this secret to  $n$  parties. Each party  $i$  will obtain a share  $s_i$ . If this is a threshold secret sharing system where the threshold is  $t$ , then any  $t$  out of these  $n$  parties may combine their shares to obtain the original secret  $s$ . The security property states that any  $t - 1$  element subset of the parties has no way of figuring out the original secret.

**Threshold cryptography** is essentially the generalization of the secret sharing idea to other systems. For example, in encryption, if we distribute shares of the secret decryption key to  $n$  parties, then the moment  $t$  of these parties combine their shares to reconstruct the secret, they have obtained the decryption key, and thus there is almost no meaning left for distributing it. In threshold encryption, this idea is taken one step further: The decryption key is never reconstructed, but any  $t$  out of  $n$  parties may jointly decrypt some given ciphertext. Threshold cryptography is hence the name given to such cryptosystems where multiple parties jointly perform some operation without reconstructing and revealing the secret.

An **escrow** is simply a public key encryption under the TTP's key, together with a label. This label is public, but is tied to the ciphertext. The TTP must verify that the conditions on the label are satisfied before decrypting the escrow.

A **verifiable escrow** is an escrow with the additional property that the contents of the ciphertext can be checked to satisfy a particular relation, without being decrypted. For example, if electronic cash is verifiably escrowed, then the party who receives the verifiable escrow can verify that the encrypted cash has real value, without obtaining the cash. Just as for regular escrows, the TTP verifies the conditions on the label before decrypting.

**Fair exchange** involves two parties who wish to exchange two items. Fairness means at the end of the exchange, either both parties obtain each other's value, or no one obtains anything useful. A fair exchange requires a TTP to be involved [71]. **Optimistic fair exchange** systems still require a TTP to be present, but the TTP is involved only when there is a dispute between the participants.

## 3 Non-Autonomous Distribution of Trust

This section concentrates on distribution of trust via multiple communicating agents. We first overview some general techniques that are applicable to a wide-range of problems. We may mention specific applications to these techniques, but the techniques themselves are general. Then, we mention solutions to distributing trust for some specific applications.

### 3.1 General Techniques

**Byzantine Agreement and Quorum Systems:** The “Specification of Dependable Trusted Third Parties” [18] provides a nice overview of some general techniques for distributing TTPs in many applications, including certificate authorities, directory services, optimistic fair exchange, digital notary (timestamping), and Kerberos-like authentication services (see also [17]). Since all the techniques presented are based on byzantine agreement type of solutions [12, 16, 39, 60, 53] as well as threshold cryptography [32, 30, 31], they require broadcast messages and thus incur high communication complexity. The general framework can be summarized as follows: The client broadcasts messages to at least  $t+1$  agents of the distributed TTP. Depending on the requirements of the TTP job, the client and the agents need to employ reliable broadcast, atomic broadcast, or secure causal atomic broadcast to realize the distributed service. The agents use threshold cryptography techniques to respond. The client waits for at least  $2t+1$  responses, and then accepts the majority decision. If secure causal atomic broadcast is used, the client encrypts her request, and in many cases, the client may perform recovery of threshold cryptosystem shares.

The specification considers general adversary structures [49, 8, 66] rather than just threshold systems. An interesting observation follows: Such systems that are based on quorums or general adversary structures may, in cases, tolerate an adversary who actually corrupts more than half of the agents. Following an example in the specification document, the agents used to distribute the TTP may employ, let’s say, four different operating systems: Microsoft Windows, Mac OS, Linux, BSD. Furthermore, it is possible that different numbers of agents use each (e.g., let’s say Microsoft Windows is used by half of the agents, Linux is used by a quarter, and Mac OS and BSD are used by one-eighth of agents each). It is possible to design the system such that the system may tolerate corruption of any two operating system while still maintaining security (e.g., if Windows and Linux are corrupted, then three-quarters of the agents are malicious, but the system is still secure) [18]. The basic idea is to create virtual players in a quorum or secure multi-party computation system such that a single virtual player is potentially simulated via multiple real agents’ interaction [49, 8].

**Threshold Cryptography Systems:** Two other related techniques include threshold cryptography [32, 30, 31] and proactive security [23]. Threshold cryptography can be thought of as an extension of threshold secret sharing. In general, secret sharing is a technique that can be used to distribute a TTP functionality to multiple agents. If a  $k$ -out-of- $n$  secret-sharing scheme is used, an adversary must corrupt  $k$  servers to break security, or fault  $n-k+1$  servers to break availability [62]. However, in secret-sharing schemes [80, 14], there are two main problems: the trusted distributor problem and the trusted combiner problem [62]. The trusted distributor problem is addressed using (publicly) *verifiable secret-sharing* schemes [38, 74, 43], and the trusted combiner problem is addressed via threshold cryptography [32, 30, 31]. Using verifiable secret sharing, one may make sure that the initial distribution of shares to distributed TTP agents is done correctly. In some cases, this is not enough of a guarantee, since this still necessitates an *initial secret creator* who is trusted. Furthermore, secret-sharing schemes, in general, require secure (encrypted and authenticated) channels between the agents. If the distributed TTP job is an authentication service, this creates a chicken-and-egg issue. The better alternative is joint generation of secrets [75, 44, 48, 40].

Jointly generating the secret is the first step toward distributing some TTPs, but it is not enough from a security standpoint. When the secret needs to be reconstructed, normally, one of the agents will be responsible for combining the shares, and will thus obtain the initial secret. If that single agent is corrupted, then the adversary obtains the whole secret, and the system is completely broken. The solution is two-fold: either (1) the reconstruction must be done by the

client contacting the TTP [5], or (2) threshold cryptography must be employed. In threshold cryptosystems [32, 30, 31], the secret is never reconstructed; rather, each agent contributes to the final output (e.g., final ciphertext or signature) using her own secret, and only this final output will be revealed instead of the secret. It makes perfect sense, for example, when the final output is a signature on a message using a key that was secret-shared: The signature is revealed, but the key is not. This way, it is possible to distribute the job of a certificate authority easily: As long as some  $k$ -out-of- $n$  agents in the distributed certificate authority sign the certificate, then it is possible to obtain a valid certificate. To break such a system, an adversary must corrupt at least  $k$  agents. Note that, the reason we have listed such a solution under non-autonomous category is that, even though the agents need not communicate with each other (especially if the client contacting the TTP performs reconstruction), there is still the initial setup phase that involves all distributed TTP agents.

Proactive security takes the idea one step further, and combines such a distributed system with periodic key refresh [23]. The main idea is that, even when threshold cryptography is employed, it is conceivable that the adversary corrupts many parties, just not at the same time. With each corrupted party, the adversary learns a share of the secret, and with enough corrupted parties (not necessarily corrupted simultaneously; thus this scenario is not considered in the system's security definition) the adversary obtains enough shares to reconstruct the original secret, hence completely breaking the security of the system. With periodic refreshes, it is now necessary that the adversary must corrupt enough parties (again, not necessarily simultaneously) before the next time period (epoch) begins, thus increasing security and longevity of such distributed systems greatly in practice.

**Trusted Hardware/Software Systems:** In some systems, a central TTP is deemed trusted based on hardware or software assumptions, including secure execution environments (e.g., Java applets), smartcards, or trusted computing modules [29, 1]. If such hardware or software trust assumptions are realistic, then the load will definitely be distributed, but not necessarily the trust. Firstly, if all of the distributed TTP agents employ the same hardware or software trust assumption (e.g., same tamper-resistant smartcard technology or operating system is used), then the security of such a distributed TTP system still relies on a central assumption; if that trust assumption is broken, it does not matter much that the system is distributed. Second, synchronization still constitutes a huge problem: If the distributed TTP is a timestamping service, then all clocks must be synchronized, if a certificate authority is distributed this way, the certificate revocation lists must be synchronized [29]. Yet, some benefits of such approaches include potentially increased security in situations where each user essentially runs its own TTP based on such a hardware or software assumption. In such a case, if a single agent is compromised, then generally only a single user will be affected (although this does not address the first issue above). Furthermore, such a distribution may be useful in many scenarios, provided that the trust assumption holds, including type checking, verifying proof-carrying code, certificate checking, timestamping, virus confinement, censorship (e.g., DRM) [1].

**State-Machine Replication Systems:** Finally, as a general technique, some works have employed the finite state machines of the TTPs to be distributed in their protocols [76, 58, 78, 55, 81], referred to as *state-machine replication*. In particular, Shmatikov and Mitchell [81] provide finite-state machine representation of the participants in two contract signing (i.e., fair signature exchange) protocols, and use a finite-state verification tool to analyze their weaknesses.

Yet, state-machine replication technique is not limited to non-autonomous scenarios. Küpçü

and Lysyanskaya [55] employ a similar technique (without a tool) to find general impossibility results regarding distributing the TTP. We will discuss details of this technique in Section 4. We believe the technique is widely-applicable to other scenarios.

### 3.2 Specific Scenarios

**E-Commerce Systems:** TTP in e-commerce has been one of the concentration areas for distribution. Some systems employ a hierarchical TTP structure not for distributing trust but for scalability [70], some look at strategies of buyers and sellers and conclude that TTPs are required to incentivize honest trading behavior and to keep the risk and thus prices low [50], and some realize that there are at least two types of different TTPs required in such systems: The first is a TTP who certifies the quality of the good being sold. In reality, forums, review sites, or reputation systems play the role of such a (possibly distributed) TTP [56, 57, 79, 77, 28]. The second is a TTP who has legal authority on the parties, and able to fine them or revoke their reputation [54].

E-commerce systems such as Bitcoin [68] try to improve the efficiency and fault tolerance of the system while reducing the trust requirements by distributing the job of the trusted central *bank* to peers in a peer-to-peer system. The system uses the proof-of-work idea [36, 35, 52, 6, 61, 65], leading to a secure system as long as the total CPU time spent by honest parties is much more than that of the attacker. As with almost all other non-autonomous distribution systems, it employs broadcast messaging or public announcements.

**Timestamping Systems:** This requirement of broadcast messaging or public announcements is also tied to the use of digital timestamping services [46, 13, 64]. The ideas used in digital timestamping is almost the same as byzantine agreement techniques. Some systems, for efficiency reasons, allow for a mixed approach: If the participants agree on a single trusted TTP, such a central TTP may be employed, otherwise a costly distributed approach must be used [15].

**Fair Exchange Systems:** In secure two-party computation (2PC) and secure multi-party computation (SMPC) [86, 45, 11, 24, 22], a general problem is fairness: how can one guarantee that the output is fairly delivered to all participants. While trying to address this issue, the use of external or internal TTPs have been proposed. External TTP systems require an additional party to play the role of TTP, whereas internal TTP systems employ some or all of the participants in the system to create a TTP. Lindell [63] proposes a 2PC system with an external TTP who can enforce that if a malicious party breaks fairness, then he must pay the honest party for his loss. Cachin and Camenisch [19] similarly proposes a 2PC extension where basically the TTP ensures fair exchange of the computation results. The techniques used are very similar to those used in the optimistic fair exchange setting [2, 56, 57].

In addition to the use of such external TTPs, internal TTP usage has also been proposed. If a 2PC protocol using a TTP exists for some task, it may be possible to use other participants in an SMPC protocol as the internal TTP [83]. It is possible to imagine a tree hierarchy of SMPC participants. For each pair of parties in this hierarchy, their parent will act as the TTP in their 2PC protocol execution. Unfortunately, this technique cannot be employed directly for all tasks, and extreme care must be taken to ensure the security properties are preserved.

**Group Signature Systems:** Group signatures introduce yet another type of TTP: group managers. In a group signature, any member of the group can sign on behalf of the group, without revealing his identity [26]. In general, a group manager needs to be able to identify, *if necessary*, the member who signed a particular message. When a group manager misbehaves, anonymity is lost. Yet, an interesting property of group signature schemes require that, at least, the group

manager cannot frame a member by claiming that she signed a message that she did not sign in reality. Hierarchical group signatures [82] are proposed to distribute the load and also trust of group managers, such that a group manager in the hierarchy can identify only users that are one level below. As an example scenario, a group manager bank can identify which one of the credit card holders signed a message, and the credit card company can identify which bank's card was used to sign that purchase message. This way, even though there are multiple group managers, a customer is required to only trust her bank (in terms of her identity in the trade), rather than the credit card company or any other bank (who may be the manager of some other group, and who may issue the same brand credit cards).

**Outsourced Computation Systems:** Looking at the issue from a different angle, one may consider outsourced computation systems [9, 79, 77]. Such systems have multiple TTPs: a *boss* who assigns the jobs and decides if it has been done correctly, and a *bank* who rewards or fines the *contractors* using credits or reputation. Interestingly, though, one may think of such systems as distributing the trust in some other way. When a group of contractors are employed to perform some job and their results are compared, instead of employing just a single contractor, such systems are effectively distributing the trust put on any one contractor. Indeed, the Belenkiy et al. [9] scheme does this using autonomous contractors, but the setting is much different than the one we consider here: they assume these distributed agents can be fined and rewarded (by possibly yet another TTP). When we distribute a TTP, ideally we do not want to rely on yet another TTP. But in some situations, for example situations as above where this technique is employed to reduce the number of TTPs in the system (the *contractor* is no longer a TTP), this strategy may make perfect sense.

**Forms of TTPs:** Lastly, TTPs are sometimes referred to as *trusted neutral parties* (since the trust is on their neutrality: they do not collude with participants of the underlying system), *semi-trusted parties*, or even *untrusted third parties* (since they are not trusted in terms of the privacy of the messages in the underlying protocol) [83, 41]. Moreover, public bulletin boards or trusted hardware may also be considered as TTPs [72]. In some settings TTP is called the *judge* emphasizing the fact that the TTP has legal authority over the participants [54]. In essence, these are all *trusted parties* in one sense or the other.

## 4 Autonomous Distribution of Trust

There is no known generic system proposed for autonomous distribution of trust, but the technique is used for several applications. Indeed, autonomous techniques are applied in distributed storage and shared memory scenarios [4, 73, 59, 47]. In general, in those systems, a *writer* is supposed to contact multiple entities for writing a single value. Similarly, a *reader* retrieves the value from multiple entities as well. In the setting where the storage servers do not communicate, this constitutes an autonomous system (assuming a static set of servers, since otherwise the initialization will be non-autonomous). Some other solutions do employ non-autonomous techniques though [37]. See [27, 85] for more discussion about distributed storage techniques.

In this section, we will concentrate on a method for analyzing autonomous TTP distribution for optimistic fair exchange protocols [55]. The main goal and hope is that these techniques may be widely applicable, thus leading to interesting open problems, and possible solution techniques for other types of protocols.

We will first explain the general technique of using finite state machine abstraction, as presented by K upc u and Lysyanskaya [55]. Then, we will summarize the known results for distributing

the TTP in optimistic fair exchange protocols, based on this finite state machine abstraction. Following that, we will discuss the role of synchrony, and summarize a technique that is known to distribute the trust using autonomous agents, assuming all agents have synchronized clocks (**though communication is asynchronous**).

Informally, an optimistic fair exchange protocol involves three participants: Alice and Bob who would like to exchange items, and the *arbiter* (TTP) who is involved only if there is a dispute between Alice and Bob. Alice and Bob each have some item that they would like to exchange. They want the exchange to be fair: At the end, either both Alice and Bob obtain each other’s item, or neither party obtains anything useful. In the case that there is a dispute, they contact the arbiter.

Slightly more formally, a fair exchange protocol is composed of three interactive algorithms: Alice running algorithm  $A$ , Bob running algorithm  $B$ , and the arbiter running the trusted algorithm  $T$ . Alice has content  $f_A$ , and Bob has content  $f_B$ . Bob would like to obtain  $f_A$  from Alice, and Alice would like to obtain  $f_B$  from Bob.

*Completeness* for an optimistic fair exchange protocol states that the interactive run of  $A$  and  $B$  by *honest parties* results in  $A$  getting  $f_B$  and  $B$  getting  $f_A$ . The arbiter’s algorithm  $T$  is not involved, assuming an ideal network where messages are not delayed or lost.

*Fairness* states that at the end of the protocol, either  $A$  obtains  $f_B$  and  $B$  obtains  $f_A$ , or neither Alice nor Bob gets anything useful. If something goes wrong, Alice and/or Bob contact the arbiter (TTP) to either *resolve* the issue, or *abort* the protocol. For formal definitions, we refer the reader to the paper by K upc u and Lysyanskaya [56].

#### 4.1 Finite State Machine Abstraction

We start by defining a general optimistic fair exchange model that fits currently known state-of-the-art optimistic fair exchange schemes. Our model uses an arbiter, and has semantics for aborting and resolving, which we define below.

All the participants (Alice, Bob and the agents that form the distributed TTP) are interactive Turing Machines (ITMs). Those ITMs have the following 4 semantic states: *working*, *aborted*, *resolved*, *dispute* (see Figure 1). These semantic states can correspond to multiple states in the actual ITM definitions of the participants, but these abstractions will be used to prove the results. A *dispute* state is not absolutely necessary for our analysis, but is included for the sake of intuitive formulation.

The ITM of each participant starts in the *working* state. Semantically, *working* state denotes any state that the actual ITM of a participant is in when the protocol is still taking place. When a participant does not receive the expected correctly-formed message from the other participant, (s)he can possibly abort or decide to contact the TTP agents for resolving or aborting with them, in which case the ITM of that participant enters her/his *dispute* state. If everything goes well in the protocol execution (all messages received from the other party are correctly formed), then the ITM of a participant transitions to the *resolved* state directly from the *working* state. Otherwise, if the TTP agents must be contacted, the ITM first enters the *dispute* state, and then transitions to either *resolved* or *aborted* state. Arbiters’ *dispute* state is not needed in our analysis, and one may safely ignore it.

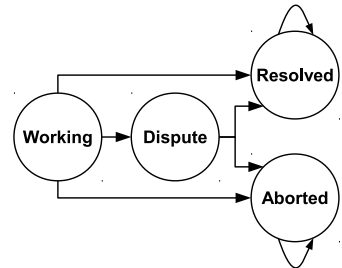


Figure 1: Semantic view of the state machines of the participants.



When the optimistic fair exchange protocol ends, Alice and Bob are allowed to end only in *aborted* or *resolved* states. If Alice ends at her *resolved* state, then, by definition, she must have obtained  $f_B$ . Similarly, if Bob ends at his *resolved* state, this means he obtained  $f_A$ . When the protocol ends, if the ITM of a participant is not in its *resolved* state, it is considered to be in its *aborted* state. Using these semantic definitions, even an adversarial ITM can be considered to have *resolved* and *aborted* states (since the adversary either obtains the honest party's item and hence ends at its *resolved* state, or not therefore ending at its *aborted* state). Thus, if both parties are honest, then both end at their *resolved* states. If both are malicious, then the protocol is not supposed to guarantee fairness. Therefore, these two cases are not interesting, and hence will not be covered in this paper. If one party is malicious and the other is honest, then we need to analyze possible scenarios.

**Definition 4.1** (End of the Protocol). *We say that the protocol has ended if both participants satisfy one of the following based on them being honest or adversarial:*

- *The honest party ended up being in her either resolved or aborted state.*
- *The adversarial party produced its final output at its either resolved or aborted state, after running at most a polynomial number of steps (polynomial in some security parameter).*

To be able to meaningfully conclude anything from the finite state machine representations of the participants, we need to rely on some observations, assumptions, specifications, possible generalizations, and most importantly, dependence between states of different parties. To do this, we adopt the DAFE (Distributed Arbiter Fair Exchange) framework [55]. We will assume that instead of using a single TTP, the optimistic fair exchange protocol now uses multiple agents (arbiters). DAFE protocols are optimistic fair exchange protocols that satisfy the following three properties, which are defined immediately afterward:

- Exclusive states assumption
- Dependence between arbiters' state and Alice's and Bob's
- Autonomous arbiters assumption

**EXCLUSIVE STATES ASSUMPTION:** This assumption states that the *resolved* and *aborted* states are mutually exclusive. For an arbiter, those states informally mean whether or not the arbiter helped one of the parties to resolve or abort. We assume that there is no combination of state transitions that can take an *honest arbiter* from the *aborted* state to the *resolved* state, or vice versa. In most existing protocols, this corresponds to the fact that the arbiter will not abort with a participant first and then decide to resolve with him or the other participant, or vice versa. An honest arbiter can keep executing abort (or resolve) protocols with other participants in the exchange while he is in the *aborted* (or *resolved*, respectively) state, but cannot switch between these states.

**Definition 4.2** (Aborting and Resolving with an arbiter). *If a participant interacts with an arbiter and aborts with him, the arbiter goes to his aborted state. Similarly, if a participant resolves with an arbiter, the arbiter goes to his resolved state.*

**Definition 4.3** (Aborted and Resolved Protocol Instance). *A protocol instance is called aborted if both Alice and Bob end at their aborted states, and called resolved if both Alice and Bob end at their resolved states.*

DEPENDENCE BETWEEN ARBITERS' STATE AND ALICE'S AND BOB'S: A resolution makes sense if at least one of the parties has not resolved yet (i.e., there is a dispute). In such a case, at least one or both of Alice and Bob is not already in their *resolved* state. The party who is not yet in its *resolved* state can end in its *resolved* state (unless (s)he already is in her/his *aborted* state) if and only if a set of arbiters end in their *resolved* states. This set of arbiters can be different for Alice or Bob. Actually, there can be more than one set of arbiters that is enough for this resolution. One may consider a threshold of arbiters to be necessary, or maybe some well-defined subset [49, 8].

AUTONOMOUS ARBITERS ASSUMPTION: We assume that the honest arbiters' decisions are made autonomously, without taking into account the decisions of the other arbiters. Arbiters can arrive at the same decision seeing the same input, but they will not consider each other's decision while making their own decisions. In particular, this means no communication takes place between honest arbiters (malicious arbiters can do anything they want).

Yet, a dependence between the arbiters' decisions can be generated by Alice or Bob, by contacting the arbiters with some specific order. Therefore, to model the autonomy, we require the protocol design to state that when the ITM of an *honest* participant decides to contact the arbiters for dispute resolution, the participant creates the message to send to all of the arbiters before receiving any response from any arbiter. One can model this with the *dispute* state in which the message to send to the arbiters are prepared all at once. Note that a malicious party can freely introduce dependence between messages to arbiters. Interestingly, Küpçü and Lysyanskaya [55] find that this indirect autonomy assumption is not necessary for most of their results, and only the direct autonomy (i.e., no communication between arbiters) suffices to prove meaningful results based on this finite state machine abstraction of optimistic fair exchange protocols.

To argue about fairness of protocols in this finite state machine abstraction model, we define semantic fairness property that must be satisfied by all optimistic fair exchange protocols.

SEMANTIC FAIRNESS: The semantic fairness property states that at the end of the protocol, Alice and Bob both end at the same state (they both end at their *aborted* states, or they both end at their *resolved* states). In other words, we need the protocol instance to be either resolved or aborted as in Definition 4.3, for every possible instance of the protocol. Note that, due to our *aborted* and *resolved* state definitions, semantic fairness implies fairness.

**Notation:** Let  $N$  denote the set of all arbiters, where there are a total of  $n$  of them ( $|N| = n$ ). An honest arbiter acts as specified by the protocol. Let  $F$  be the set of arbiters who are friends with a malicious participant. Those arbiters are adversarial (e.g., may appear as aborted to the honest party, but still resolve with the malicious party).

Define two sets  $\mathcal{H}_{\mathcal{R}}$  and  $\mathcal{M}_{\mathcal{R}}$ , which are sets of sets. Any set  $H_R \in \mathcal{H}_{\mathcal{R}}$  is a set of arbiters that is sufficient for the *honest* party to *resolve*. Similarly, any set  $M_R \in \mathcal{M}_{\mathcal{R}}$  is a set of arbiters that is sufficient for the *malicious* party to *resolve*. Therefore, by definition, in case of a dispute, the honest party will end at her *resolved* state **if and only if** she resolves with **all** the arbiters in **any one** of the sets in  $\mathcal{H}_{\mathcal{R}}$  (assuming she is not already in her *resolved* or *aborted* state). Similarly, the malicious party will end at his *resolved* state **if and only if** he resolves with **all** the arbiters in **any one** of the sets in  $\mathcal{M}_{\mathcal{R}}$  (assuming he is not already in his *resolved* or *aborted* state). For DAFE protocols, these sets are well-defined by the protocol description, and do not change once the honest party enters its *dispute* state.

A special case of these sets can be represented as thresholds. Let  $T_H$  be the number of arbiters the honest party needs to contact for resolution. Similarly,  $T_M$  denotes the number of arbiters the malicious party needs to contact for resolution. Thus, the set  $\mathcal{H}_{\mathcal{R}}$  is composed of all subsets of  $N$

with  $T_H$  or more arbiters. Similarly, the set  $\mathcal{M}_{\mathcal{R}}$  is composed of all subsets of  $N$  with  $T_M$  or more arbiters.

Define  $R_H$  as the set of arbiters the honest party  $H$  has already resolved with, and  $R_M$  as the set of arbiters the malicious party  $M$  has already resolved with. Also define  $R_A$  as the set of all arbiters that are available for  $H$  for resolution. Initially, when the dispute resolution begins, we assume that  $R_H = \emptyset$ ,  $R_M = F$ , and  $R_A = N - F$  (and all arbiters are available for resolution to the malicious party). We furthermore have the following actions and their effects on these sets:

**Action 1.**  $H$  resolves with an arbiter  $X$ : As a result,  $R_H$  becomes  $R_H \cup \{X\}$ .

**Action 2.**  $M$  resolves with an arbiter  $X$ : As a result,  $R_M$  becomes  $R_M \cup \{X\}$ .

**Action 3.**  $H$  aborts with an arbiter  $X \in R_A$ : As a result,  $R_A$  becomes  $R_A - \{X\}$ .

**Action 4.**  $M$  aborts with an arbiter  $X \in R_A$ : As a result,  $R_A$  becomes  $R_A - \{X\}$ .

As a final note, the DAFE protocols do not assume that the arbiters have access to synchronized clocks. We assume that the adversary can re-order messages, delay the honest party's messages to the arbiters, insert his own messages, etc. But he cannot delay honest party's messages *indefinitely*: the honest party eventually reaches the arbiters that he wants to contact initially.

## 4.2 Applying DAFE Framework

In this section, we provide a sample use of such a finite state machine abstraction framework. Full results have been presented by K upc u and Lysyanskaya [55]. The idea is to first prove smaller results, called *scenarios*, and then use these in proving results for full protocols. In these proofs, only the state machine semantics described in Section 4.1 are used, and no particular assumption about the underlying protocol are employed. This way, the results are as general as possible.

### 4.2.1 Scenario 1: $M$ can Abort

In this scenario, we consider a protocol instance where the malicious party has the ability to abort and resolve. The honest party can possibly abort and resolve too, but this does not affect the results. In this scenario, actions 1, 2, and 4 are possible. The results will remain valid regardless of action 3 being possible.

**Lemma 4.1.** *For every DAFE protocol instance where  $M$  can abort, there must exist a time  $t$  when  $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t) \exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$  s.t.  $H_R \subseteq M_R - F(t)$ .*

*Proof.* Assume otherwise: At any time in the protocol instance  $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$  s.t.  $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t)$   $H_R \not\subseteq M_R - F(t)$ . The malicious party can break fairness as follows: He aborts with the set of arbiters  $R_A - M_R$ , and resolves with the set of arbiters  $M_R$ . Since no  $H_R$  is now a subset of the available arbiters  $R_A = M_R - F(t)$ , the honest party cannot resolve, while the malicious party already resolved. Thus this protocol instance is unfair (does not satisfy semantic fairness).  $\square$

**Corollary 4.1.1.** *Using threshold-based mechanisms, there must exist a time  $t$  that satisfies  $T_H \leq T_M - |F(t)|$ .*

### 4.2.2 Scenario 2: Only $H$ can Abort

In this scenario, we assume that the malicious party has the ability to resolve only (since the case where he can abort is already covered in the previous scenario), whereas the honest party can abort and resolve. In this scenario, actions 1 to 3 are possible (action 4 is not possible).

**Lemma 4.2.** *For every DAFE protocol instance where only  $H$  can abort, there must exist a time  $t$  when  $\forall M_R \in \mathcal{M}_{\mathcal{R}}(t) \exists H_R \in \mathcal{H}_{\mathcal{R}}(t)$  s.t.  $H_R \subseteq M_R - F(t)$ .*

*Proof.* Assume otherwise: At any given time  $\exists M_R \in \mathcal{M}_{\mathcal{R}}(t)$  s.t.  $\forall H_R \in \mathcal{H}_{\mathcal{R}}(t) H_R \not\subseteq M_R - F(t)$ . The malicious party can break fairness as follows: When  $H$  wants to abort the protocol,  $M$  lets abort messages to all arbiters in  $R_A - M_R$  reach their destinations, but intercept the messages to  $M_R - F(t)$ . He then resolves with  $M_R$ . Even if  $H$  notices this, he cannot go and resolve since there is no set  $H_R \in \mathcal{H}_{\mathcal{R}}(t)$  that will allow him to. Therefore, this protocol instance also does not satisfy semantic fairness.  $\square$

Note that the result of Lemma 4.2 is the same as Lemma 4.1, and therefore all the corollaries apply to this scenario too.

### 4.2.3 Protocols where only one party can Abort

In this section, we consider protocols where only one party may initiate an abort. Without loss of generality, Alice is given the ability to abort and resolve, whereas Bob is given only the ability to resolve. Consider the following two cases when such a protocol is run:

**Case 1: Honest Alice and Malicious Bob:** This case falls under Scenario 2, which requires that  $\forall B_R \in \mathcal{B}_{\mathcal{R}} \exists A_R \in \mathcal{A}_{\mathcal{R}}$  s.t.  $A_R \subseteq B_R - F_B$ .

**Case 2: Malicious Alice and Honest Bob:** This case falls under Scenario 1, which requires that  $\forall A_R \in \mathcal{A}_{\mathcal{R}} \exists B_R \in \mathcal{B}_{\mathcal{R}}$  s.t.  $B_R \subseteq A_R - F_A$ .

These two cases lead to the conclusion that every protocol instance needs two sets  $A_R \in \mathcal{A}_{\mathcal{R}}$  and  $B_R \in \mathcal{B}_{\mathcal{R}}$  s.t.  $A_R = B_R \subseteq TR$ , where  $TR$  denotes the set composed of all trusted arbiters. These arbiters must be trusted, and so there is no point in distributing the TTP to multiple agents. It is even worse: If any of these arbiters are corrupted, the DAFE protocol fails to be fair. When considering threshold-based schemes, this corresponds to the requirement that  $T_B \leq T_B - F_A - F_B$ , which means no party should have any friends for such a protocol to be fair. If even one arbiter is corrupted, the protocol becomes unfair.

Therefore, we conclude that for DAFE protocols where only one party can abort, distributing the TTP is worse than not distributing (since having multiple arbiters means higher chance of corrupting one of them). Küpçü and Lysyanskaya [55] present even more impossibility results regarding other types of DAFE protocols.

## 4.3 Implications for Existing Optimistic Fair Exchange Protocols

We have observed, in the context of optimistic fair exchange protocols, how a finite state machine abstraction can be employed to prove general results about protocols having common semantics. Before commenting on the results about protocols where only one party can abort, as discussed above, let us investigate a generic mechanism of converting a regular optimistic fair exchange protocol to a DAFE protocol; equivalently, distributing the trust put on the TTP to multiple autonomous arbiters.

As a representative of optimistic fair exchange protocols, we consider a state-of-the-art protocol due to Asokan, Shoup and Waidner (ASW) [2]. Their protocol employs verifiable escrow [20, 21, 2, 56] under the (one and only) arbiter’s public key. The intuition behind using verifiable escrows is that the recipient can verify, without learning the actual content, that the encrypted content is the content that is promised and the arbiter can decrypt it. Verifiable secret sharing techniques can be employed to split the promised secret to multiple arbiters. Each of these secrets will be encrypted under a different arbiter’s public key. The recipient can still verify those encrypted shares can be decrypted and combined to obtain the promised secret, thereby effectively achieving the same goal as a verifiable escrow, but for multiple arbiters. The resolution procedure now requires contacting multiple arbiters. For example, if the threshold for the secret sharing method used is  $k$ , the resolution will involve contacting at least  $k$  arbiters. For a detailed explanation of how to use verifiable secret sharing in distributing the arbiters, we refer the reader to [5].

In terms of the state semantics of the participants, it is clear that the ending states of the participants can be parsed into *aborted* and *resolved* states which are mutually exclusive. There is no message exchange between the arbiters; they are autonomous. As for the dependence between arbiters’ state and Alice’s and Bob’s, since resolution needs  $k$  shares, and secure secret sharing and encryption methods are used, a participant can obtain the other participant’s exchange item if and only if (s)he resolves with at least  $k$  arbiters (in case of a dispute). This relationship makes perfect sense when multiple autonomous arbiters are used, since the main goal in distributing the arbiter is distributing the trust. Therefore, the goal is to find some number of honest arbiters each one of which will individually contribute to dispute resolution between participants by resolving or aborting with them. When arbitrary sets are used instead of thresholds, it is easy to see all these arguments will still apply.

Unfortunately, the ASW protocol allows Alice to abort and resolve whenever necessary, and Bob to resolve if necessary. Considering the extension presented above where the TTP job is distributed to multiple autonomous arbiters, this protocol fits the definition of protocols where only one party can abort. This means, using static resolution sets and autonomous arbiters, ASW-type protocols cannot be extended to use multiple arbiters to distribute trust in an asynchronous system.

## 4.4 Role of Synchrony

Remember that the DAFE framework explicitly stated that the arbiters do not have access to synchronized clocks. In this section, we extend the DAFE framework with timeouts of arbiters (accomplished via synchronized clocks), and present the DAFET framework (DAFE with Timeouts). We first present the changes to the original framework, and then present possibility and optimality results regarding the synchronous setting.

### 4.4.1 DAFET Framework

In DAFET protocols, we allow for timeouts by giving the arbiters access to loosely synchronized clocks. To reflect existing constructions, instead of actions 3 and 4 in DAFE protocols (honest or malicious party aborting), the following action is allowed:

**Action 5.** *An arbiter  $X \in R_A - R_H - R_M$  times out: As a result,  $R_A$  becomes  $R_A - \{X\}$ .*

Another difference between DAFE and DAFET protocols is the sets  $\mathcal{H}_{\mathcal{R}}$  and  $\mathcal{M}_{\mathcal{R}}$  being static and dynamic, respectively. DAFE protocols define such sets as static: the overall set of arbiters

that needs to be contacted for resolution does not change with time once the honest party enters its *dispute* state (hence the notation  $\mathcal{H}_{\mathcal{R}}$  and  $\mathcal{M}_{\mathcal{R}}$ ).

Consider a DAFE type protocol that employs dynamic sets like this: Bob can resolve only with arbiters that Alice has already resolved with. We can think of it as Bob’s set initially being empty, and then getting populated. Unfortunately, this protocol does not guarantee timely resolution (unless there is a timeout in the protocol) since Bob may need to wait indefinitely for Alice.

In contrast, we allow DAFET protocols to employ dynamic sets (hence the notation  $\mathcal{H}_{\mathcal{R}}(t)$  and  $\mathcal{M}_{\mathcal{R}}(t)$ ). These sets may depend on the timeout and possibly the parties’ actions in that particular instance of the protocol. Consider the following two cases as illustrative examples: Some type of protocols allow, let’s say, Alice to resolve only after a timeout. Some other type of protocols allow Alice to resolve only with an arbiter that Bob has already resolved with (or vice versa). In analyzing such types of protocols, we will consider  $\mathcal{H}_{\mathcal{R}}(t)$  and  $\mathcal{M}_{\mathcal{R}}(t)$  as dynamic, letting them change with those actions.

Lastly, the set of friends of a malicious party can also change with time, if the adversary is allowed to adaptively corrupt arbiters. In that case, we will use the notation  $F(t)$ . As for DAFE protocols, we assume that the adversary can re-order messages, delay the honest party’s messages to the arbiters, insert his own messages, etc. But we assume that the honest party reaches the arbiters that he wants to contact before the timeout, whenever necessary.

#### 4.4.2 Results in the Synchronous Setting

When one considers synchrony between distributed agents forming the TTP, the results may change. In particular, using the DAFET framework, we allow the arbiters to have access to a synchronized clock, and employ timeouts. Timeouts are tied to the use of dynamic sets in general. When only one party can resolve before the timeout, static resolution sets lose their meaning since the resolution set for the party who cannot resolve before the timeout is empty until the timeout. That set gets defined only after the timeout, which results in that set being dynamic in a very basic sense. Thus, the set of arbiters needed by a party for resolution changes during the course of the execution of the protocol instance. By adjusting resolution sets reactively, a protocol may provide semantic fairness. One such existing protocol is due to Avoine and Vaudenay (AV) [5].

They present a three-step protocol: (1) Alice starts by sending verifiable secret shares of her item encrypted under each arbiter’s public key. (2) Bob responds with his item. (3) Alice responds with her item. To resolve, Bob contacts  $k$  arbiters before the *timeout* to get the decrypted shares and reconstruct the secret of Alice (where  $k$  is the threshold for the secret sharing scheme). Before giving the decrypted share, each honest arbiter asks for the item of Bob. When Alice, after the timeout, contacts an arbiter who has resolved with Bob, then she is given Bob’s item. Alice may need to contact up to  $n - k + 1$  arbiters. Note that the moment Bob resolves with any honest arbiter, Alice is guaranteed to be able to resolve. It is relatively straightforward to see that this constitutes a DAFET protocol.

In this protocol, sets  $\mathcal{H}_{\mathcal{R}}(t)$  and  $\mathcal{M}_{\mathcal{R}}(t)$  are dynamic. The set  $\mathcal{B}_{\mathcal{R}}(t)$  contains all subsets of  $N$  with  $k$  or more arbiters and  $\mathcal{A}_{\mathcal{R}}(t)$  is initially empty<sup>1</sup>. We further have the following action (specific to this protocol):

---

<sup>1</sup>It does not contain the empty set, it is empty. This means initially no set of arbiters is sufficient for Alice to resolve.

**Action 6** (Bob resolves with an arbiter  $X \in R_A$ ). As a result, the set  $\{X\}$  is added to the set of sets  $\mathcal{A}_{\mathcal{R}}(t)$ .

Küpçü and Lysyanskaya [55] present a detailed analysis of this protocol using the finite state machine abstraction, and considering further scenarios and protocol types. We will only re-iterate their results to discuss the effectiveness of such a framework.

**Lemma 4.3.** *AV protocol cannot provide semantic fairness unless for all times  $t \forall B_R \in \mathcal{B}_{\mathcal{R}}(t) B_R \not\subseteq F_B$  AND for some time  $t \exists B_R \in \mathcal{B}_{\mathcal{R}}(t)$  s.t.  $B_R \cap F_A = \emptyset$ .*

**Corollary 4.3.1.** *AV protocol cannot provide semantic fairness unless  $|F_B| < T_B$  AND  $T_B \leq n - |F_A|$ .*

It is important to notice that the AV paper [5] proves essentially the same result: They prove that the same bound is also sufficient for their protocol. Importantly, the result above is applicable to all protocols of the same type; no DAFET protocol of the same type can achieve better bounds. In particular, the same technique of employing multiple autonomous arbiters can be used on the ASW protocol [2] to convert their timeout-based version to a DAFET protocol, and the same lemma will hold. This shows how such a framework can easily be applied to prove optimality of a protocol and extended to other protocols of the same type.

As the corollary immediately reveals, when using  $n$  arbiters, to obtain maximum tolerance, one should set the threshold for Bob  $T_B = n/2$  so that the protocol tolerates up to  $n/2 - 1$  friends of each participant. Of course, this greatly reduces the efficiency of the resolution of the optimistic fair exchange protocol, but still provides a much more efficient solution than using byzantine agreement or secure multi-party computation. Essentially, there is no communication between arbiters themselves; only the client communicates with  $T_B$  arbiters, and can do so in parallel.

## 5 Conclusion and Open Problems

Throughout this article, we presented various methods to distribute trusted third parties (TTPs). TTPs play a crucial role in many useful scenarios and applications, including fair exchange, electronic commerce, certificate authorities, authentication services, etc. Therefore, it is a very important research topic to consider distributing the trust put on TTPs to multiple entities.

It is possible to effectively distribute the trust put on any TTP using byzantine agreement or secure multi-party computation techniques. Unfortunately, such generic techniques are very costly; they incur quadratic communication complexity, and sometimes have high coordination costs. As an alternative TTP distribution technique, one may consider employing autonomous agents, and form a framework to analyze classes of protocols that have some common finite machine semantics. To this end, we have re-presented some of the results by Küpçü and Lysyanskaya [55] showing the effectiveness of such a finite state machine abstraction approach.

Yet, their results apply only to distributing the TTP of optimistic fair exchange protocols. It is interesting to apply similar frameworks on other topics. Thus, analyzing the distribution of trust put on the TTP employing autonomous agents in other scenarios, including digital notary (timestamping) services, online bidding applications, certification authorities, directory services, and authentication services remain as open problems. Analyzing the effect of timeouts, or in general synchrony, in those scenarios is also useful. Alternatively, it may always be beneficial to reduce

the number of TTPs in a system. One such approach is used in outsourced computation schemes (where reliance on a single TTP –the contractor who is trusted to perform the job correctly– is removed by employing multiple contractors) [9], and further applications should be sought. Lastly, finding fast non-autonomous ways of distributing TTPs for specific protocols (not general ones), possibly employing threshold cryptography, may constitute a valid research direction.

Finally, already there are many legal issues surrounding the use of TTPs in various applications. Just as an example, consider the electronic cash usage [25]. If offline e-cash is employed, once double-spending occurs, there is theft. But can the TTP (the bank) really charge the double-spender? The answer depends on the relevant jurisdiction, and the issue is especially complicated when buyer and seller are under different jurisdictions [42]. Ironically, some TTPs present disclaimers; they do not provide any legal warranty about the service they provide and reject responsibility [84]. If a TTP disclaims any warranty, how can it be a trusted party?

## Acknowledgements

We acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 111E019. We would like to thank Idit Keidar for her encouragement and invitation to prepare this article, as well as her detailed feedback. We also thank Anna Lysyanskaya, as Section 4 of this paper highly depends on our joint paper on the same topic.

## References

- [1] Martin Abadi. Trusted computing, trusted third parties, and verified communications. In *Security and Protection in Information Processing Systems*, volume 147, pages 290–308. 2004.
- [2] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Selected Areas in Communications*, 18:591–610, 2000.
- [3] Giuseppe Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *ACM CCS*, 1999.
- [4] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of ACM*, 42:124–142, 1995.
- [5] Gildas Avoine and Serge Vaudenay. Optimistic fair exchange based on publicly verifiable secret sharing. In *ACISP*, 2004.
- [6] Adam Back. Hashcash - a denial of service counter-measure. <http://www.hashcash.org>.
- [7] Feng Bao, Robert Deng, and Wenbo Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Security and Privacy*, 1998.
- [8] Donald Beaver and Avishai Wool. Quorum-based secure multi-party computation. In *EUROCRYPT*, 1998.
- [9] Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing outsourced computation. In *NetEcon*, 2008.



- [10] Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *ACM WPES*, 2007.
- [11] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *STOC*, 1993.
- [12] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, 1988.
- [13] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical report, Clarkson University, 1991.
- [14] George Robert Blakley. Safeguarding cryptographic keys. In *NCC*, 1979.
- [15] A. Bonneau, P. Liardet, A. Gabillon, and K. Blibech. A distributed time stamping scheme. In *SAR*, 2005.
- [16] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of ACM*, 32:824–840, 1985.
- [17] Christian Cachin. Distributing trust on the internet. In *DSN*, 2001.
- [18] Christian Cachin. Specification of dependable trusted third parties. Technical report, IBM MAFTIA deliverable D26, 2001.
- [19] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *CRYPTO*, 2000.
- [20] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In *ASIACRYPT*, 2000.
- [21] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.
- [22] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, 1996.
- [23] Ran Canetti, Rosario Gennaro, and Amir Herzberg. Proactive security: Long-term protection against break-ins. *CryptoBytes*, 3:1–8, 1997.
- [24] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
- [25] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.
- [26] David Chaum and Eugne van Heyst. Group signatures. In *EUROCRYPT*, 1991.
- [27] Gregory Chockler, Idit Keidar, Rachid Guerraoui, and Marko Vukolic. Reliable distributed storage. *IEEE Computer*, 42:60–67, 2009.
- [28] Bram Cohen. Incentives build robustness in bittorrent. In *WEPS*, 2003.

- [29] Alexander W. Dent and Geraint Price. Certificate management using distributed trusted third parties. *Trusted Computing*, 6:251, 2005.
- [30] Yvo Desmedt. Society and group oriented cryptography: a new concept. In *CRYPTO*, 1987.
- [31] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.
- [32] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.
- [33] Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. Optimistic fair exchange in a multi-user setting. In *PKC*, 2007.
- [34] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of ACM*, 32:191–204, 1985.
- [35] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *CRYPTO*, 2003.
- [36] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.
- [37] Mohammad Etemad and Alptekin Küpçü. Transparent, distributed, and replicated dynamic provable data possession. In *ACNS*, 2013.
- [38] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987.
- [39] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32:374–382, 1985.
- [40] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive rsa. In *CRYPTO*, 1997.
- [41] Matthew K. Franklin and Michael K. Reiter. Fair exchange with a semi-trusted third party. In *ACM CCS*, 1997.
- [42] A. Michael Froomkin. The essential role of trusted third parties in electronic commerce. *Oregon Law Review*, 75:49, 1996.
- [43] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *EUROCRYPT*, 1998.
- [44] Rosario Gennaro, Stanisaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *EUROCRYPT*, 1996.
- [45] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.
- [46] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3:99–111, 1991.

- [47] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM TOPLAS*, 12:463–492, 1990.
- [48] Amir Herzberg, Markus Jakobsson, Stanisaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *ACM CCS*, 1997.
- [49] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13:31–60, 2000.
- [50] Xiaorui Hu, Zhangxi Lin, Andrew B. Whinston, and Han Zhang. Hope or hype: On the viability of escrow services as trusted third parties in online auction environments. *Information Systems Research*, 15:236–249, 2004.
- [51] IETF. *RFC 4120, The Kerberos Network Authentication Service*.
- [52] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *CMS*, 1999.
- [53] Valerie King and Jared Saia. Scalable byzantine computation. *ACM SIGACT News Distributed Computing Column*, 41:89–104, 2010.
- [54] Alptekin Küpçü. Official arbitration with secure cloud storage application. Cryptology ePrint Archive, Report 2012/276.
- [55] Alptekin Küpçü and Anna Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS*, 2010.
- [56] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, 2010.
- [57] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56:50–63, 2012.
- [58] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.
- [59] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1:86–101, 1986.
- [60] Leslie Lamport. The part-time parliament. *ACM TOCS*, 16:133–169, 1998.
- [61] Ben Laurie and Richard Clayton. Proof-of-work proves not to work. In *WEIS*, 2004.
- [62] Albert Levi and M. Ufuk Çağlayan. The problem of trusted third party in authentication and digital signature protocols. In *ISCIS*, 1997.
- [63] Yehuda Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, 2008.
- [64] Helger Lipmaa. *Secure and Efficient Time-Stamping Systems*. PhD thesis, University of Tartu, 1999.
- [65] Debin Liu and L Jean Camp. Proof of work can work. In *WEIS*, 2006.
- [66] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11:203–213, 1998.

- [67] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, 2003.
- [68] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <http://www.bitcoin.org>.
- [69] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *ACM Communications*, 21:993–999, 1978.
- [70] Khoi Nguyen. A hierarchical trusted third-party system for secure peer-to-peer transactions. Master’s thesis, San Jose State University, 2007.
- [71] Henning Pagnia and Felix C. Gartner. On the impossibility of fair exchange without a trusted third party. Technical report, Darmstadt University of Technology TUD-BS-1999-02, 1999.
- [72] Henning Pagnia, Holger Vogt, and Felix C. Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–75, 2003.
- [73] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *ACM SIGMOD*, 1988.
- [74] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.
- [75] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, 1991.
- [76] Michael Reiter and Kenneth Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems*, 16:986 – 1009, 1994.
- [77] Rosetta@home. <http://boinc.bakerlab.org/rosetta/>.
- [78] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM CSUR*, 22:299–319, 1990.
- [79] Seti@home. <http://setiathome.berkeley.edu>.
- [80] Adi Shamir. How to share a secret. *ACM Communications*, 22:612–613, 1979.
- [81] Vitaly Shmatikov and John C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283:419–450, 2002.
- [82] Mrten Trolin and Douglas Wikstrm. Hierarchical group signatures. *Automata, Languages and Programming*, 3580:103–103, 2005.
- [83] Jaideep Vaidya and Chris Clifton. Leveraging the multi in secure multi-party computation. In *ACM WPES*, 2003.
- [84] Verisign. Disclaimer. <https://www.symantec.com/about/profile/policies/legal.jsp>.
- [85] Marko Vukolic. The byzantine empire in the intercloud. *ACM SIGACT News Distributed Computing Column*, 41:105–111, 2010.
- [86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*, 1986.