Partially-Fair Computation from Timed-Release Encryption and Oblivious Transfer

Geoffroy Couteau[1], Bill Roscoe[2], Peter Ryan[3]

[1] IRIF, Paris-Diderot University, CNRS, France
couteau@irif.fr
[2] Department of Computer Science, University of Oxford, Oxford, UK
awroscoe@gmail.com
[3] Interdisciplinary Centre for Security and Trust, University of Luxembourg, Luxembourg
peter.ryan@uni.lu

Abstract. We describe a new protocol to achieve two party ε-fair exchange: at any point in the unfolding of the protocol the difference in the probabilities of the parties having acquired the desired term is bounded by a value ε that can be made as small as necessary. Our construction uses oblivious transfer and sidesteps previous impossibility results by using a *timed-release* encryption, that releases its contents only after some lower bounded time. We show that our protocol can be easily generalized to an ε-fair two-party protocol for *all functionalities*. To our knowledge, this is the first protocol to truly achieve ε-fairness for all functionalities. All previous constructions achieving some form of fairness for all functionalities (without relying on a trusted third party) had a strong limitation: the fairness guarantee was only guaranteed to hold if the honest parties are at least as powerful as the corrupted parties and invest a similar amount of resources in the protocol, an assumption which is often not realistic. Our construction does *not* have this limitation: our protocol provides a clear upper bound on the running time of all parties, and partial fairness holds even if the corrupted parties have much more time or computational power than the honest parties. Interestingly, this shows that a minimal use of timed-release encryption suffices to circumvent an impossibility result of Katz and Gordon regarding ε-fair computation for all functionalities, without having to make the (unrealistic) assumption that the honest parties are as computationally powerful as the corrupted parties – this assumption was previously believed to be unavoidable in order to overcome this impossibility result. We present detailed security proofs of the new construction, which are non-trivial and form the core technical contribution of this work.

Keywords. Fair exchange, partial fairness, timed-release encryption

1 Introduction

Secure computation allows parties to perform a joint computation on their private data, without compromising their security. An important security property

of secure computation protocols is known as *fairness*: intuitively, it states that either all participants to the protocol should receive the output, or none should. In a wide variety of real-world situations, ensuring that no participant can get an unfair advantage by learning the output early is highly desirable. Unfortunately, a well-known result of Cleve [10] established that fairness is impossible to achieve in its full generality – in fact, it is already impossible to achieve for very simple functionality such as coin tossing, or exchange of values. As a consequence of this impossibility result, a large body of work has been devoted to developing mechanisms to achieve some relaxed notion of fairness. We overview the main existing approaches below, and outline their advantages and drawbacks.

1.1 Relaxed Notions of Fairness

Some lines of research overcome Cleve's impossibility result by relying to some extent on a trusted third party [2, 9, 12], non-standard communication models [25], or by punishing unfair behaviour through smart contracts [22–24, 26]. Another approach works by gradually increasing the parties' confidence in the output [3,16,27]; however, this approach is inherently limited to protocols with a single-bit output, and where the output is the same for all parties; furthermore, they allow the adversary to significantly bias the output of the honest parties by aborting early.

Fairness From Gradual Weakening of Encryption. Most closely related to our work is the following important line of research in fairness, which seeks to achieve a relaxed notion of fairness where if the adversary can recover the output in time T, then the honest parties can recover it as well within time $s \cdot T$, where s is some slackness parameter [7, 8, 11, 13, 14, 29]. Therefore, this approach guarantees fairness, as long as all honest parties are *at least as computationally powerful* as the corrupted parties. At an intuitive level, this approach will proceed by letting the parties jointly compute an encryption of the output, and gradually "weakening" its security in rounds, until its content can be recovered by brute-force. This approach, however, has several downsides: when an adversary aborts early, the protocol does not specify how a party should decide whether to invest the necessary computational effort to recover the output. More generally, the protocol does not provide any a priori (polynomial) upper-bound on the computational effort that honest parties might have to invest in the protocol: if any such precise bound is given, the adversary is guaranteed to break the fairness property by investing more resources than specified by this bound. In any real-world situation, this mean that the protocol will only satisfy fairness under the unrealistic assumption that the corrupted parties will never be able to spend more computational resources than the honest parties.

An alternative to all of the above is the notion of *partial fairness*. Since it will be the main focus of our work, we elaborate on it in the next section.

1.2 Partial Fairness

The notion of partial fairness was introduced by Katz and Gordon in [18], and was recently re-discovered by Roscoe and Ryan in a different context [31], where it was called *stochastic fairness*. Partial fairness relaxes the standard fairness notion to hold except with some non-negligible probability, which can be made smaller than any given (inverse polynomial) threshold. In an informal sense, partial fairness corresponds to a best-possible notion of fairness, in settings where one does not want to rely on trusted parties, or to assume that the computational power of honest parties will be as high as those of malicious parties. An important feature of the notion partial fairness is that it fits nicely in the standard simulation paradigm of secure computation, allowing to provide formal security proofs. Indeed, the simulation paradigm established the security of a protocol by exhibiting a simulator which is given access to an ideal functionality, and whose behavior cannot be distinguished from that of a honest user. Now, proving that a protocol satisfies $1/p$-partial fairness (for some polynomial p) is done by exhibiting a simulator which is given access to a *perfectly fair* functionality, and whose behavior cannot be distinguished from that of a honest user *except with* $1/p$ *probability*.

In [18], Katz and Gordon exhibit a generic partially-fair secure computation protocol, provided that either one of the inputs or one of the outputs comes from a polynomial-size domain. While this already considerably broadens the type of functionalities that can be implemented compared to the setting of full fairness, this remains a rather strong limitation. It prevents, for example, to evaluate functionalities as simple and useful as fair exchange of data, unless one of the data comes from a very small domain. Unfortunately, Katz and Gordon showed that this limitation is *inherent* [18, Section 4], by proving that it is already impossible to securely execute (with partial fairness) a form of authenticated fair exchange (where two parties wish to exchange values if and only if they have been correctly authenticated via some one-time MAC scheme) when the values come from a large domain. Katz and Gordon further mention that their setting requires a polynomial upper-bound on the running time of the parties, which is why alternative fairness notions (based on gradually weakening a commitment to the output until it can be opened by brute-force) escape their impossibility result.

1.3 Context and Motivation

The starting point of our work, and its initial motivation, stems from considerations regarding the security some existing password-authenticated key-exchange (PAKE, [5]) protocols against a form of online attacks which are not captured by the standard security model for PAKE. A PAKE is an interactive protocol between a server and a user, both holding a (low-entropy) password, who wish to securely generate a shared secret-key provided that their passwords are equal. The protocol should resist offline dictionary attacks: an adversary should not

be able, given only the transcript of a PAKE execution, to try out many passwords against this transcript. In the past decade, numerous PAKE protocols have been proposed, satisfying this natural security notion, under a variety of cryptographic assumptions [4–6, 15, 20].

However, the standard security model for PAKE does not preclude the following simple *online* attack: an adversary could potentially attempt to guess the password, learn from the protocol whether his guess was successful or not, and then abort the protocol before the server gets to know that the user tried to execute the PAKE with an incorrect password. Since network failures are relatively common, the server cannot distinguish in this scenario a malicious attempt at guessing the password from a network failure for an honest user. Because of this, the adversary could potentially repeat this attack several times before the repeated failures become suspicious. This form of online attack was first identified and studied by Ryan and Roscoe in [31]. This is not a purely theoretical concern: studies indicate that human-generated passwords have less that 7 bits of min-entropy on average [19]. Even if the servers enforce the use of strong passwords, with (say) 20 bits of entropy, and assuming a medium-scale deployement of a PAKE system with 2^{10} online services protected by the system, and an adversary allowed to make 2^{10} online guesses with the above attack (possibly spanning over a reasonable period of time) would break into one of the services with good probability. To mitigate this attack, [31] suggest to rely on a fair exchange protocol, and show that such a protocol can be used to ensure that the adversary cannot learn whether his guess was correct without the server learning it as well. Since fair exchange protocols are impossible in general, [31] suggests to rely on a protocol with partial fairness (or *stochastic fairness*, using their terminology). This way, any adversary attempting to mount an online guessing attack has a high (yet not overwhelming) probability of getting caught doing so.

1.4 Our Contribution

In this work, we develop a new method to construct protocols with partial fairness. Inspired by the above scenario, we introduce a new *partially-fair exchange protocol* (i.e., a $(p + O(1))$-round two-party protocol which allows for exchange of values, and satisfies $1/p$-fairness in the framework of [18]), and seek to obtain a protocol as concretely efficient as possible. Afterward, we observe that our protocol extends naturally to an *authenticated* partially fair exchange protocol, and show that this naturally gives rise to a secure computation protocol with partial fairness *for all functionalities*. Because of the impossibility result of Katz and Gordon, this can provably not be achieved directly within the standard model of computation, and all previous works aiming at fairness for all functionalities could only achieve a very relaxed notion of partial fairness (either using trusted parties, or assuming that the honest parties are computationally more powerful than corrupted parties). We stress that this is the case even for protocol that used tools such as gradual weakening of encryptions, or time-lock puzzles - even

though the use of these primitives does, in principle, escape the impossibility result of Katz and Gordon, since they do not fit directly into the standard model of computation.

Escaping the Impossibility Result. To escape the impossibility result of Katz and Gordon, we rely on a *timed-release encryption scheme*, a primitive introduced in [30] which allows to encrypt a message such that it can only be recovered after some time period has elapsed. A timed-release encryption scheme is simply a public-key version of the most well-known notion of time-lock puzzles; this is similar to the primitive employed in [7, 8, 11, 13, 14, 29], and can be constructed under the assumption that some tasks inherently require a long sequential computation – the most classical construction relying on the hardness of parallelizing squaring modulo an RSA modulus [30].

However, there is an important difference between our work and all previous works on fairness which relied on time-lock puzzle, timed-release encryption, or gradual weakening of secrets: in all these works, the use of these primitives only guarantees fairness when the honest parties are always able to invest more computational resources than the corrupted parties, a scenario which we already said is unrealistic. In contrast, our protocol *truly* achieves partial fairness, even if the corrupted parties are allowed a *much longer* running time than the honest parties; in fact, our protocol does even guarantee a strict polynomial upper-bound on the running time of all parties, hence does not suffer from the important downside of these works. We note that this is a surprising result, as the absence of a strict polynomial upper bound on the running time of the honest parties was pointed out in [18] as the reason why fair exchange protocols could escape their impossibility result; our result shows that this is not the case, and that a minimal use of a primitive in the spirit of time-lock puzzles already suffices to overcome this barrier.

In addition to timed-release encryption, we assume only standard generic cryptographic primitives, such as commitment schemes and oblivious transfers. To optimize for concrete efficiency, we do not employ zero-knowledge proofs and do not target security against malicious adversaries; rather, we prove that our protocol is directly secure against *covert* adversaries [1]: an adversary can deviate from the specifications of the protocol, but will be caught (with probability one) if he does so. Note that in the motivating scenario of preventing online guessing attacks on PAKEs, security against covert adversaries captures the desired security notion, since our aim is to distinguish guessing attacks from honest network failures. The security of our protocol can easily be enhanced to the malicious setting using zero-knowledge proofs.

To summarize, our main contributions are

- On the *practical* side, a new partially fair key exchange protocol from timed-release encryption and standard cryptographic primitives, which is formally proven secure against covert adversaries in the framework of [18]. Our protocol is concretely efficient: when instantiating the oblivious transfer with the

DDH-based OT of [28], the protocol communicates only $4p \log p + O(1)$ group elements in $p + O(1)$ rounds to reach $1/p$-fairness. This protocol can be used to mitigate the risk of online guessing attacks on password-authenticated key exchange protocols.
- On the *theoretical* side, a generic secure two-party computation protocol for all polynomial-size circuits (with input and output domain of arbitrary size) which achieves $1/p$-fairness in $p + O(1)$ rounds using $\tilde{O}(\lambda p) + O(c)$ bits of communication (where λ is a security parameter, and c is the communication of a protocol computing the circuit and satisfying security with aborts), which is (to our knowledge) the very first protocol to achieve partial fairness for all functionalities (or even for the fair exchange functionalities). Our protocol makes a minimal use of a timed-release encryption scheme (which seems unavoidable by the impossibility result of [18]), where each party sends a *single* timed-release encryption right before the output phase, and must complete the phase before the time bound elapses. In particular, our protocol is the first of its kind to guarantee a polynomial upper-bound on the running time of all parties, for arbitrary functionalities.

In particular, this means that the fairness guarantee that we obtain does also extend to the (very realistic) scenarios where the adversaries might be more powerful than the honest parties; to our knowledge, every previous paper achieving some form of fairness for all functionalities (without the help of a trusted third party, or a smart contracts) could not guarantee this highly desirable property. In addition, our protocol is not purely of theoretical interest: it is practical, and its building blocks can be instantiated efficiently from a variety of standard cryptographic assumptions.

1.5 Our Method

Our starting point is an idea sketched in [31], which achieves partial fairness by creating a randomly permuted size-p list of masked values, one of them being the target value to be exchanged, and the remaining ones being dummy values. Each list of masked values is permuted by both parties, so that the actual permutation remains unknown to each party. Before the exchange phase, the parties encrypt a string indicating their choices of permutations (for both lists), as well as the mask used to hide the values, using a timed-release encryption scheme which ensures that the encrypted values remain hidden for a time T. Afterward, the parties simply exchange the permuted values one-by-one, using p rounds of interaction, so that the last value of the list is exchanged before time T elapse. Intuitively, partial fairness stems from the fact that if the adversary aborts at any point in the computation, he does not know yet whether his opponent already sent him the right masked value (he will only discover this after time T has elapsed). Therefore, the best advantage he can obtain over his opponent is by aborting right after he received *any* given message, which gives him a probability roughly $1/p$ of discovering later on that he had already received the masked output, while his opponent had not.

The protocol developed in [31], however, relies on an ad-hoc construction using discrete-log-hard groups; more importantly, it entirely lacks any security analysis. We therefore first develop and elegant partially-fair exchange protocol, inspired by the approach of [31], relying on a (simulatable) oblivious transfer protocol, a timed-release encryption scheme, and a commitment scheme. Afterward, we provide a detailed security analysis of our protocol in the framework of [18], and show that it $1/p$-realizes a perfect fair exchange functionality in the presence of covert adversaries. While the security of the protocol is relatively intuitive, the proof turns out to be non-trivial, and is the main technical contribution of this work. To establish the existence of an efficient simulator, we must rely on cryptographic primitives with strong simulation guarantees; in particular, we need a simulatable oblivious transfer (as defined in [28]), together with an *equivocable trapdoor commitment*: in addition to the standard hiding and binding properties, we require that with an appropriate trapdoor, any commitment can be opened to an arbitrary value; yet, at the same time, the commitment must be *weakly extractable*, meaning that with another appropriate trapdoor, an extraction algorithm can recover a message m such that no PPT adversary (without the equivocation trapdoor) can open this commitment to a value $m' \neq m$. The proof requires carefully tracking the advantage of a polynomial time adversary in distinguishing the real protocol from the simulated one, where we must show that the (non-negligible) distinguishing advantage can be broken in two parts, one corresponding to a malicious behavior of a non-aborting adversary (which we show can be detected with overwhelming probability, hence is acceptable in the setting of security against covert adversaries), and another quantity which corresponds to the fairness error induced by an aborting adversary, which we must show to be bounded by $1/p$. To prove the latter, we introduce a useful proof technique, whih is new to our knowledge and might have other applications for building and analyzing protocols using timed-release encryption: we analyze the advantage adversary conditioned on aborting at any given round, and crucially rely on the fact that if we *know* that the adversary is going to abort at round i (since we condition on this event), then the adversary can be thought of as having running time bounded above by the time bound of the timed-release encryption scheme, hence we can use this adversary to derive a contradiction with respect to the semantic security of the timed-release encryption scheme. This is a rather unusual method, in the sense that this step allows us to replace values from a distribution in the security game by values from another distribution which is *not* indistinguishable from the first one from the viewpoint of the environment; however, it cannot be distinguished from the first one by the environment *before the adversary aborts the protocol*, due to the timed-release property and the conditionning on an early abort of the adversary.

1.6 Informal Overview of the Protocol

We described here a simplified version of the protocol, to ease the presentation - the actual protocol handles additional technicalities required to achieve provable security. We assume familiarity with standard cryptographic primitives such as

oblivious transfer here – necessary details about standard primitives can be found in the preliminaries. At a high level, our protocol proceeds as follows: it is parametrized by a polynomial $p = p(\lambda)$, which will correspond to the number of rounds of the protocol. Intuitively, the parties will exchange values V_A and V_B, hidden among dummy values and appropriately masked, such that the parties will only learn *after the protocol* the round number at which they actually got their output value - guaranteeing that any attempt to abort before the protocol is completed will not allow them to break fairness, except with probability roughly $1/p$. More precisely:

- First, the two parties (Alice and Bob) generate random masks (k_A, k_B), and pick random indices (i_A, i_B) between 1 and p, as well as random permutations $(\pi_A, \pi_B, k_A \oplus V_A)$ of $[1, p]$.
- A will then commit to her index (let r_A be the opening information), and encrypt $(r_A, i_A, \pi_A, k_A \oplus V_A)$ with a timed-release encryption scheme, that can only be bruteforced after some time T has elapsed (unless the secret key is known). Alice sends the commitment and the encryption to Bob; Bob executes a similar procedure in the other direction.
- Both parties exchange the first flow of a 1-out-of-p oblivious transfer protocol, each playing the role of the receiver in the parallel instances, using their random choice of index as their selection value.
- Alice and Bob each compute their p messages $(m_A^i)_{i \leq p}$ and $(m_B^i)_{i \leq p}$, playing the role of the sender in the two parallel OT instances, each using their random masks (k_A for Alice, k_B for Bob) as input for each of the p messages (that is, all p input messages of the player P are equal to k_P for $P \in \{A, B\}$).
- In each of the next rounds, for $i = 1$ to p, Alice sends $m_A^{\pi_A(i)}$ and Bob sends $m_B^{\pi_B(i)}$. Note that the same message is 'encrypted' in all OT messages: however, the receiver security guarantees that the sender $P \in \{A, B\}$ does not know which of the OT messages the receiver can decrypt to k_P, while the permutation chosen by the sender ensures that the receiver himself cannot yet know which message he can decrypt (it is important here that the key k_P is random, so that the receiver cannot notice a successful decryption attempt).
- Upon completion of all p rounds, Alice and Bob open the commitment and the encryption, revealing their secret index as well as $(k_A \oplus V_A, k_B \oplus V_B)$, from which each party can recover the output.

In the above protocol, all p rounds of interaction must be completed before the time T within which the timed-release encryption can be bruteforced has ellapsed. If any party aborts early, by the security of the commitment scheme and that of the timed-release encryption guarantees that this adversary cannot know whether he had or not already received the OT message that he can decrypt, nor whether or no his opponent did (unless, of course, the party aborts before his opponent received any message at all). Since aborting right before sending a message can only give, informally, 'one round of advance' to a cheating party, this party has only probability $1/p$ of having already received his outpout while his

opponent has not - which both parties will find out within time T, by bruteforcing the timed-release encryption.

Unlike all previous protocols using similar primitives, it is not important here that T is higher than the time the corrupted party could possibly invest; rather, it suffices that T is higher than the time it takes to complete the p rounds of the protocol, and failing to complete the protocol within time T simply amounts to aborting before the end of the protocol – hence guaranteeing a strict upper bound on the running time of all parties. Of course, an adversary can cheat by putting wrong values inside the commitment and/or the timed-release encryption, hence the protocol is not secure against malicious adversaries; however, as we will formally show in the rest of this paper, the protocol outlined above does satisfy *1-deterrent covert security* (meaning that if the adversary attempts to cheat, other than by aborting early, he will be detected with probability negligibly close to 1) and $1/p$-fairness.

2 Preliminaries

A positive function f is *negligible* if for any polynomial p there exists a bound $B > 0$ such that, for any $\lambda \geq B$, $f(\lambda) \leq 1/|p(\lambda)|$. An event depending on λ occurs with *overwhelming probability* when its probability is at least $1 - \mathsf{negl}(\lambda)$ for a negligible function negl. Given a finite set S, the notation $x \xleftarrow{\$} S$ means a uniformly random assignment of an element of S to the variable x. A *distribution ensemble* X is an infinite sequence of random variables $X = \{X(a, \lambda)\}_{a \in D_\lambda, \lambda \in \mathbb{N}}$, where D_λ is a set that can depend on λ. Following [17], we define for any poynomial p the notion of *computational* $1/p$*-indistinguishability*:

Definition 1 (Computational $1/p$-Indistinguishability [17]). *Two distribution ensembles $X = \{X(a, \lambda)\}_{a \in D_\lambda, \lambda \in \mathbb{N}}$ and $Y = \{Y(a, \lambda)\}_{a \in D_\lambda, \lambda \in \mathbb{N}}$ are computationally $1/p$-indistinguishable, and write $X \overset{1/p}{\approx} Y$, if for any non-uniform PPT adversary Adv, there exists a function $\mu(\cdot) = \mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}, a \in D_\lambda$,*

$$|\Pr[\mathsf{Adv}(X(a, \lambda)) = 1] - \Pr[\mathsf{Adv}(Y(a, \lambda)) = 1]| \leq \frac{1}{p(\lambda)} + \mu(\lambda).$$

Two distribution ensembles are *computationally indistinguishable* if they are computationally $1/p$-indistinguishable for every polynomial p.

Two Party Computation. A two-party protocol between parties A and B is said to *compute* a functionality $f : (x, y) \mapsto (f_A(x, y), f_B(x, y))$ if it runs in polynomial time and satisfies the following natural correctness requirement: at the end of the protocol, if A begins with input x and B begins with input y, then A outputs $f_A(x, y)$ and B outputs $f_B(x, y)$ (for simplicity, we consider only deterministic functionalities; the definition easily extends to randomized functionalities).

The security of a two-party computation protocol is usually defined in the *real/ideal paradigm*, by showing that every attack a real adversary can mount on the real protocol can be translated to an attack performed by an ideal adversary on an ideal functionality computing the desired function, which is perfectly secure by definition. Given an ideal functionality $\mathcal{F}$, we define the random variable $\text{IDEAL}_{\mathcal{F},\mathsf{Adv}}(x, y, \lambda)$ as the output of an ideal adversary Adv together with the output of parties with respective inputs (x, y) following the execution of $\mathcal{F}$ on (x, y), with security parameter λ. Given a real protocol Π, we define the random variable $\text{REAL}_{\Pi,\mathsf{Adv}}(x, y, \lambda)$ as the output of a real adversary Adv together with the output of parties with respective inputs (x, y) following the execution of Π on (x, y), with security parameter λ. Then a protocol Π is said to $1/p$-*securely compute* a functionality $\mathcal{F}$ if Π emulates the ideal functionality $\mathcal{F}$ to within a difference of $1/p$. More precisely, let p be an arbitrary polynomial.

Definition 2 ($1/p$-Secure Computation [17]). *Let $\mathcal{F}$ be an ideal functionality, and Π be a two-party protocol which computes $\mathcal{F}$. Then Π is said to* $1/p$-securely compute *$\mathcal{F}$ if for every non-uniform PPT* Adv *against Π, there exists a non-uniform PPT ideal adversary* Sim *(called the* simulator*) such that*

$$\{\text{IDEAL}_{\mathcal{F},\mathsf{Sim}}(x, y, \lambda)\}_{x,y,\lambda} \overset{1/p}{\approx}_\lambda \{\text{REAL}_{\Pi,\mathsf{Adv}}(x, y, \lambda)\}_{x,y,\lambda}.$$

Two Party Computation against Covert Adversaries. The above definition of two-party computation captures security against *malicious adversaries*, who can mount arbitrary attacks on a protocol. A weaker security model, which remains very relevant in practice, is the *covert security model*: in this model, the parties might still arbitrarily deviate from the specification of the protocol, but they do not want to be caught cheating, hence they will not adopt a malicious behavior which would be detected with too high probability. This security model has been formalized in [1], who gave several variants. In this work, we will focus on the *failed simulation* formulation, as this formulation can be integrated in a very natural way to the notion of $1/p$-secure computation. Intuitively, the failed simulation formulation states that a malicious adversary can cause the simulation to fail by cheating, but if he can cause the simulation to fail with probability x, then he will be caught cheating with probability $\varepsilon \cdot x$, where ε is called the *deterrence factor* of the protocol. Extending this definition to $1/p$-secure computation, we will say that a protocol $1/p$-securely compute a functionality against covert adversaries with deterrence factor ε if every time an adversary causes the simulation to be distinguishable from a real run of the protocol with probability $1/p + x$, then he is caught cheating with probability $x \cdot \varepsilon$. More formally, let us first define the notion of *detection accuracy* for protocols with static corruption from [1]:

Definition 3 (Detection Accuracy [1]). *A party P_b in a two-party protocol Π (with $b \in \{0, 1\}$) is said to* detect cheating *the party P_{1-b} if it outputs $\mathsf{corrupted}_{1-b}$ in Π. A two-party protocol Π is* detection accurate *if the probability that a party outputs $\mathsf{corrupted}_b$ when party P_b is not corrupted is negligible.*

We can now formally define covert $1/p$-security:

Definition 4 (Covert $1/p$-Security). *Let $\mathcal{F}$ be an ideal functionality, and Π be a two-party protocol between parties P_0 and P_1 which computes $\mathcal{F}$. Then Π is said to* $1/p$-securely compute $\mathcal{F}$ in the presence of covert adversaries with ε-deterrent *if it is* detection accurate *and for every non-uniform PPT* Adv *against Π which corrupts P_b, there exists a non-uniform PPT ideal adversary* Sim *(called the* simulator*) such that for every inputs (x, y) and every non-uniform PPT distinguisher D,*

$$\Pr[P_{1-b} \textit{ outputs } \mathsf{corrupted}_b] \geq \varepsilon(\lambda) \cdot (|\Pr[D(\text{IDEAL}_{\mathcal{F},\mathsf{Sim}}(x, y, \lambda)) = 1] - \Pr[D(\text{REAL}_{\Pi,\mathsf{Adv}}(x, y, \lambda)) = 1]| - 1/p(\lambda)) - \mathsf{negl}(\lambda).$$

3 A Partially-Fair Exchange Protocol

3.1 Definition

Informally, a partially-fair exchange protocol allows two parties to exchange their inputs, with the guarantee that either the two parties will learn their output, or none will, except with a $1/\mathsf{poly}$ probability which can be made arbitrarily small. In other words, the protocol realizes the fair exchange functionality, except with probability $1/\mathsf{poly}$. More precisely:

Definition 5 (Partially-Fair Exchange). *A partially-fair exchange protocol is a family $\{\Pi_p\}_{p\in\mathsf{poly}}$ of two-party protocol such that for any polynomial p, the protocol Π_p $1/p$-securely compute the ideal functionality $\mathcal{F}_{\mathsf{fe}}$ represented on Figure 1.*

Ideal Functionality $\mathcal{F}_{\mathsf{fe}}$

The functionality interacts with parties through perfectly secure authenticated channels. It maintains a memory M. It ignores all incorrectly formatted messages.

1. On input a message $(\mathsf{input}, \mathsf{sid}, x_A, A, B)$ from A and $(\mathsf{input}, \mathsf{sid}, x_B, B, A)$ from B, where sid is a unique session identifier, if M does not contain an item starting with (sid, A, B), store $(\mathsf{sid}, A, B, x_A, x_B)$ in M, and send $(\mathsf{ready}, \mathsf{sid}, A, B)$ to Adv. Otherwise, ignore the input.
2. On input a message $(\mathsf{proceed}, \mathsf{sid}, A, B)$ from A and B, search M for an item $(\mathsf{sid}, A, B, x_A, x_B)$ for some (x_A, x_B) (ignore if there is none). Send x_B to A and x_A to B, and erase $(\mathsf{sid}, A, B, x_A, x_B)$ from M.
3. On input a message $(\mathsf{abort}, \mathsf{sid}, A, B)$ from either A or B, search M for an item $(\mathsf{sid}, A, B, x_A, x_B)$ for some (x_A, x_B) (ignore if there is none). Send $\perp$ to A and B, and erase $(\mathsf{sid}, A, B, x_A, x_B)$ from M.

Fig. 1. Ideal Functionality $\mathcal{F}_{\mathsf{fe}}$ for fair exchange between two parties A and B.

3.2 Building Blocks

Equivocable Trapdoor Commitment. An equivocable trapdoor commitment scheme is computationally hiding, computationally binding commitment scheme which satisfies two properties:

- it is *equivocable*, meaning that with some appropriate trapdoor, any commitment can be opened to an arbitrary value;
- it is *weakly extractable*, meaning that given an appropriate trapdoor, there is an extraction algorithm that recovers a message m from a commitment such that no PPT adversary can open this commitment to a value $m' \neq m$.

We provide a formal definition below.

Definition 6. *(Equivocable Trapdoor Commitment) An equivocable trapdoor commitment C with message space $\mathcal{M}$, commitment space $\mathcal{C}$, opening space $\mathcal{D}$, and random source $\mathcal{R}$, is a 5-tuple of PPT algorithms $(C.\mathsf{Setup}, C.\mathsf{Commit}, C.\mathsf{Verify}, C.\mathsf{Extract}, C.\mathsf{Equivocate})$, such that*

- $C.\mathsf{Setup}(1^\lambda)$*, on input the security parameter, generates the public parameters* pp *of the scheme and a trapdoor* τ*,*
- $C.\mathsf{Commit}(\mathsf{pp}, m; r)$*, given the message* $m \in \mathcal{M}$ *and some random coins* $r \in \mathcal{R}$*, outputs a pair commitment-opening* (c, d)*,*
- $C.\mathsf{Verify}(\mathsf{pp}, c, m, d)$*, given a commitment* $c \in \mathcal{C}$*, a message* $m \in \mathcal{M}$*, and an opening* $d \in \mathcal{D}$*, outputs a bit* $b \in \{0, 1\}$*,*
- $C.\mathsf{Extract}(\tau, c)$*, given a trapdoor* τ *and a commitment* c*, outputs a message* $m \in \mathcal{M}$*,*
- $C.\mathsf{Equivocate}(\tau, c, m)$*, given a trapdoor* τ*, a commitment* c*, and a message* m*, output an opening* $d \in \mathcal{D}$*,*

which satisfies the following properties:

Correctness. For any $(\mathsf{pp}, \tau) \leftarrow C.\mathsf{Setup}(1^\lambda)$*, any* $(m, r) \in \mathcal{M} \times \mathcal{R}$*, if* $(c, d) = C.\mathsf{Commit}(\mathsf{pp}, m; r)$*, then* $C.\mathsf{Verify}(\mathsf{pp}, c, m, d) = 1$*.*

Equivocable. A commitment scheme C is equivocable *if for any and $m \in \mathcal{M}$, the following distributions are indistinguishable:*

$$\{(\mathsf{pp}, \tau) \xleftarrow{\$} C.\mathsf{Setup}(1^\lambda), (c, d) \xleftarrow{\$} C.\mathsf{Commit}(\mathsf{pp}, m) : (\mathsf{pp}, c, d)\}$$

and

$$\{(\mathsf{pp}, \tau) \xleftarrow{\$} C.\mathsf{Setup}(1^\lambda), c \xleftarrow{\$} \mathcal{C}, d \leftarrow C.\mathsf{Equivocate}(\tau, c, m) : (\mathsf{pp}, c, d)\}.$$

Weakly Extractable. A commitment scheme C is weakly extractable *if for any PPT adversary* Adv*, it holds that*

$$\Pr\begin{bmatrix}(\mathsf{pp}, \tau) \xleftarrow{\$} C.\mathsf{Setup}(1^\lambda),\\ (c, m, d) \leftarrow \mathsf{Adv}(\mathsf{pp}),\\ m' \leftarrow C.\mathsf{Extract}(\tau, c)\end{bmatrix} : (m \neq m') \wedge (C.\mathsf{Verify}(\mathsf{pp}, c, m, d) = 1)\Bigg] \approx 0.$$

Instantiating Equivocable Trapdoor Commitments. There are several possible approaches to constructing equivocable trapdoor commitments. The most natural one, however, is simply to start from a standard extractable commitment (e.g. ElGamal encryption), and to replace the usual opening (which consists in revealing the message m and the random coin r) by a zero-knowledge proof that the ciphertext encrypts m. Equivocability follows immediatly from the simulatability of the zero-knowledge proof, while weak-extractability follows directly from the extractability of the commitment scheme, together with the soundness of the proof system. Instantiating the extractable scheme with ElGamal, the zero-knowledge proof can either be any four-move interactive zero-knowledge proof for the DDH relation (very efficient and standard protocols exist for this relation), or a non-interactive proof (which can be secure in the random oracle, or alternatively rely on pairing-based cryptography if we use ElGamal over a pairing-friendly elliptic curve. We note that pairing-based non-interactive proofs for linear languages such as DDH relations can be as short as a *single* group element, using the scheme of Kiltz and Wee [21]).

Public-Key Encryption. We first recall the standard definition of a semantically-secure public-key encryption scheme:

Definition 7 (Encryption Scheme). *An encryption scheme E is a triple of efficient algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs a public key* pk *and a secret key* sk*,* pk *specifies the message space $\mathcal{M}$, the ciphertext space $\mathcal{C}$, and the random source $\mathcal{R}$;*
- $\mathsf{Enc}(\mathsf{pk}, m; r)$*, given the message m, outputs a ciphertext c, under the encryption key* pk *with the randomness r;*
- $\mathsf{Dec}(\mathsf{sk}, c)$*, outputs a plaintext m, encrypted in the ciphertext c using the decryption key* sk*.*

Encryption schemes are assumed to satisfy the following properties:

Correctness. An encryption scheme Π is *correct* if for any pair of keys $(\mathsf{pk}, \mathsf{sk})$ generated by KeyGen and any message $m \in \mathcal{M}$, it holds that $\mathsf{Dec}(\mathsf{Enc}(m)) = m$.

Semantic Security ($\mathsf{IND\text{-}CPA}$*).* The classical security notion for encryption is the indistinguishability of ciphertexts: no adversary can distinguish the encryptions of the plaintexts m_0 and m_1 of its choice, given just access to public key.

Timed-Release Encryption. To define timed-release encryption, we first introduce the notion of a T-bounded algorithm: we say that an algorithm is *T-bounded* if it runs in sequential time strictly upper-bounded by T. A *(T, T')-timed release encryption scheme*, with $T' \geq T$, is a public-key encryption scheme where semantic security is relaxed to hold only against T-bounded PPT adversaries, with an additional T'-bounded PPT algorithm $\mathsf{ForceDec}$ which, on input (pk, c), outputs $m = \mathsf{Dec}(\mathsf{sk}, c)$. Timed-release encryption was introduced in [30]; it can be constructed under the assumption that squaring modulo an RSA modulus cannot be parallelized efficiently [30].

Oblivious Transfer. An oblivious transfer protocol allows a receiver to obliviously select one of n strings held by a sender, with the guarantee that the sender will not learn which string was selected, while the receiver will not learn anything about the strings he did not selected. We will rely in this work on a simulatable two-round 1-out-of-n oblivious transfer protocol (in the common reference string model); we provide a formal definition below.

Definition 8 (Simulatable Oblivious Transfer). *A two-round simulatable* 1*-out-of-n oblivious transfer protocol in the common reference string model is a quadruple of PPT algorithms* $(\mathsf{Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{Decode})$ *such that*

- $\mathsf{Setup}(1^\lambda, b)$*, on input the security parameter an a bit* b *(called the* mode*), outputs a pair* (pp, τ) *where* pp *a set of public parameters, and* τ *is a trapdoor (used only in the simulation);*
- $\mathsf{OT}_1(\mathsf{pp}, i)$*, on input a selection value* $i \in [n]$*, outputs a pair* (c_1, o) *where* o *is called the* opening information*;*
- $\mathsf{OT}_2(\mathsf{pp}, c_1, i', m)$*, on input a value* c_1*, a value* $i' \in [n]$*, and a message* $m \in \mathcal{M}$ *(*$\mathcal{M}$ *is the message space), outputs a value* c_2*;*
- $\mathsf{Decode}(\mathsf{pp}, c_2, o)$*, on input a value* c_2 *and an opening information* o*, outputs a message* $m' \in \mathcal{M} \cup \{\perp\}$*.*

The scheme is assumed to satisfy the following properties:

Correctness. For any $i \in [n]$ *and any message* $m \in \mathcal{M}$*,*

$$\Pr\left[\begin{array}{ll}(\mathsf{pp}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 0), & \\ (c_1, o) \xleftarrow{\$} \mathsf{OT}_1(\mathsf{pp}, i), & : \mathsf{Decode}(\mathsf{pp}, c_2, o) = m \\ c_2 \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}, c_1, i, m) & \end{array}\right] = 1.$$

Indistinguishability of Modes. The distributions D_0 *and* D_1*, where* $D_b = \{(\mathsf{pp}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, b) : \mathsf{pp}\}$*, are computationally indistinguishable.*

Sender Simulatability. There exists a simulator $\mathsf{SenderSim}$ *which satisfies the following: for any* $(\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 0)$ *and every (possibly malformed)* c_1*,* $\mathsf{SenderSim}(\tau, c_1)$ *outputs a value* i *such that for every pair of messages* (m_0, m_1)*, and every* $i' \neq i$*,* $\{\mathsf{OT}_2(\mathsf{pp}, c_1, i', m_0)\}$ *and* $\{\mathsf{OT}_2(\mathsf{pp}, c_1, i', m_1)\}$ *are statistically indistinguishable.*

Receiver Simulatability. There exists a simulator $\mathsf{ReceiverSim}$ *which satisfies the following: for every* $i \in [n]$*, the distributions*

$$\{(\mathsf{pp}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1), (c_1, o_1, \cdots, o_n) \xleftarrow{\$} \mathsf{ReceiverSim}(\tau) : (c_1, o_i)\}$$

and $\{(\mathsf{pp}, \tau) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1), (c_1, o_i) \xleftarrow{\$} \mathsf{OT}_1(\mathsf{pp}, i) : (c_1, o_i)\}$ *are statistically indistinguishable.*

Instantiating Simulatable Oblivious Transfer. The most natural approach to instantiate the above primitive is to rely on the construction of [28], which can be based on either DDH, quadratic residuosity, or LWE. The second message of a 1-out-of-p simulatable OT under the DDH assumption requires $2 \log p$ group elements under the scheme of [28], leading to the claimed communication in the introduction of this paper.

3.3 Protocol

We now proceed with the description of a partially-fair exchange protocol in the common reference string model. The protocol is parametrized with a polynomial p such that it $1/p$-securely compute the fair exchange functionality $\mathcal{F}_{\mathsf{fe}}$, with security against covert adversaries (and a negligible deterrence factor). The protocol relies on timed-release encryption; it assumes that both parties have access to a synchronized clock. Furthermore, for correctness, we assume that there is a known upper-bound Δ on the network delay for message transmission. We represent the protocol on Figure 2.

Theorem 9. *The protocol Π_{sfe} $1/p$-securely compute $\mathcal{F}_{\mathsf{fe}}$ in the presence of covert adversaries with 1-deterrent.*

After providing an informal overview of the proof, we formally prove Theorem 9 in Section 3.5.

3.4 Informal Overview

To prove Theorem 9, we exhibit a simulator who "almost-correctly" simulates Π_{sfe} given access to $\mathcal{F}_{\mathsf{sfe}}$. The simulator, who does not know A's input x_A, will play as follows: the initialization phase is executed honestly, except that the value $(d_A||x_A) \oplus K_A$ in the ciphertext E_A is replaced by a random value x'_A of the appropriate size (Sim will derive later on the appropriate mask K_A to transmit to B so that x'_A is unmasked to $(d_A||x_A)$). From the initial commitment com_B and ciphertext E_B of B, Sim extracts his input x_B and sends it to the functionality $\mathcal{F}_{\mathsf{fe}}$. He also extracts the round number i^* at which B should obtain the key K_A, if he did not abort before. Then, during the fair exchange phase, Sim computes his OT_2 messages on dummy inputs, except for round i^* (the simulatability of the OT guarantees that all OT_2 messages are perfectly lossy, except for the one corresponding to the round i^* identified by Sim, hence this simulation is indistinguishable from a real run of the protocol). At round i^*, if B has not yet aborted, Sim will send $\mathsf{proceed}$ to the functionality $\mathcal{F}_{\mathsf{fe}}$, and get x_A. He will then equivocate the commitment com_A to derive an appropriate opening d_A which "explains" com_A as a commitment to x_A, and sends the key $K_A \leftarrow x'_A \oplus (d_A||x_A)$ to B. If B aborts before round i^*, Sim simply sends abort to $\mathcal{F}_{\mathsf{fe}}$.

The above simulation fails in two situations:

- When B cheats by not transmitting the opening of com_B to A, or by not including the random coins of his OT_1 message $c_{1,B}$ in E_B.

Protocol Π_{sfe}

The protocol is parametrized with a polynomial p and a network delay upper-bound Δ.

- **Primitives.** The protocol relies on an equivocable trapdoor commitment $C = (C.\mathsf{Setup}, C.\mathsf{Commit}, C.\mathsf{Verify}, C.\mathsf{Extract}, C.\mathsf{Equivocate})$, a (T, T')-timed release encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ForceDec})$ (where the value T is chosen so that executing steps 2-3 of the protocol takes time at most T given the upper-bound Δ on the network delay), and a two-round simulatable 1-out-of-$p(\lambda)$ oblivious transfer $(\mathsf{Setup}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{Decode})$.
- **Inputs.** The parties A and B holds respective inputs x_A and x_B (which are assumed to be λ-bit long for simplicity).
- **Setup.** On input the security parameter 1^λ, the setup algorithm computes $(\mathsf{pk}_A, \mathsf{sk}_A) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $(\mathsf{pk}_B, \mathsf{sk}_B) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $(\mathsf{pp}_A, \tau_A) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 0)$, $(\mathsf{pp}_B, \tau_B) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 0)$, and $(\mathsf{pp}, \tau) \leftarrow C.\mathsf{Setup}(1^\lambda)$. It outputs $\mathsf{crs} \leftarrow (\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pp}_A, \mathsf{pp}_B, \mathsf{pp})$.

We now proceed with the actual description of the protocol.

1. **Initialization.** A picks a random index $i_A \xleftarrow{\$} [p(\lambda)]$, a random key $K_A \xleftarrow{\$} \{0,1\}^{2\lambda}$, a random coin r_A for the oblivious transfer, and a random permutation π_A of $[p(\lambda)]$. She computes $(\mathsf{com}_A, d_A) \xleftarrow{\$} C.\mathsf{Commit}(\mathsf{pp}, x_A)$, $E_A \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_A, (i_A, r_A, \pi_A, (d_A || x_A) \oplus K_A))$, $(c_{1,A}, o_{1,A}) \leftarrow \mathsf{OT}_1(\mathsf{pp}_A, i_A; r_A)$, and sends $(E_A, c_{1,A}, \mathsf{com}_A)$ to B. B executes the corresponding symmetrical operations (with public parameters $\mathsf{pk}_B, \mathsf{pp}_B$), and sends $(E_B, c_{1,b}, \mathsf{com}_B)$ to A. Upon reception of the other party's message, both parties start a timer.
2. **Fair Exchange.** This phase proceeds in $p(\lambda)$ rounds. For $j = 1$ to $p(\lambda)$, the parties execute the following simultaneously: A sends $c_{2,A,j} \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, \pi_A(j), K_A)$, and B sends $c_{2,B,j} \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}_A, c_{1,A}, \pi_B(j), K_B)$.
3. **Reveal.** A sends (x_A, π_A) to B. Simultaneously, B sends (x_B, π_B) to A.
4. **Output.** In the output phase, A does the following:
   - If B aborted before the start of the fair exchange phase, she outputs $y_A \leftarrow \perp$.
   - Otherwise, she executes the $\mathsf{ForceDec}$ procedure to recover (i_B, r_B, π_B, x'_B). She checks whether $c_{1,B} = \mathsf{OT}_1(\mathsf{pp}_B, i_B; r_B)$; if this checks fails, she outputs $\mathsf{corrupted}_B$.
   - Otherwise, let j denote the index of the round at which B aborted (or failed to send a message before time T elapsed); if B did not abort in the fair exchange phase, set $j \leftarrow p(\lambda) + 1$. A checks that *both* $\pi_A(i_B) \leq j$ and $\pi_B(i_A) < j$. If any of these checks fails, she outputs $y_A \leftarrow \perp$.
   - Otherwise, she computes $K_B \leftarrow \mathsf{Decode}(\mathsf{pp}_A, c_{2,B,\pi_B(i_A)}, o_{1,A})$, $x_B || d_B \leftarrow x'_B \oplus K_B$, and she checks that $C.\mathsf{Verify}(\mathsf{pp}, \mathsf{com}_B, x_B, d_B) = 1$. If this check fails, she outputs $\mathsf{corrupted}_B$; otherwise, she outputs $y_A \leftarrow x_B$.

   B performs the corresponding symmetrical operations and outputs y_B (or $\mathsf{corrupted}_A$).

Note that the values exchanged during the Reveal phase are not used in the output phase. The purpose of the Reveal phase is to let the parties obtain their output before the time T' elapsed in practice, while being able to check later on (after executing the $\mathsf{ForceDec}$ procedure) whether their adversary was honest or malicious. For the security analysis, however, we only consider the outputs obtained by the parties after the $\mathsf{ForceDec}$ procedure.

Fig. 2. Partially-Fair Exchange Protocol Π_{sfe} between two parties A and B.

– When B aborts at a round j such that he already got enough information to obtain his output (after executing the $\mathsf{ForceDec}$ procedure), but A did not yet receive the information on her output. Indeed, in Sim's simulation, perfect fairness is always guaranteed by $\mathcal{F}_{\mathsf{fe}}$.

The first issue corresponds to B following an active cheating strategy; however, this strategy is always successfully detected by A, who outputs $\mathsf{corrupted}_B$ when she does not receive the appropriate opening and coins. This captures the fact that the protocol is secure against covert adversaries with 1-deterrent: a malicious adversary can deviate from the protocol (and break partial fairness), but will always be caught when doing so.

The second issue corresponds exactly to the $1/p$ gap in partial fairness. Most of the analysis will be devoted to proving that this situation happens with probability at most $1/p$ (conditioned on the first situation not happening, *i.e.*, A not outputting $\mathsf{corrupted}_B$). The analysis proceeds by considering the probability of this even conditioned on B aborting at round j, for every possible round j. Then, the crucial observation is that when he aborts early, B can be thought of as a T-bounded adversary. This will allow us to invoke the semantic security of the timed-release encryption scheme to show that its content is hidden from B. From this, we conclude that the rounds at which A and B must receive their outputs (respectively $\pi_B(i_A)$ and $\pi_A(i_B)$) are uniformly random (π_A, i_A are honestly picked at random by Sim, and they remain hidden from B, who chooses π_B, i_B). Therefore, in the event of the round $\pi_B(i_A)$ being after some given round j, while $\pi_A(i_B) \leq j$, can be shown to be (almost) independent of B's behavior (including his choice to abort or not). This allows us to bound the probability of this event by $\frac{j}{p} \cdot \frac{p+1-j}{p} + \mathsf{negl}(\lambda)$ for every j. Each term being upper bounded by $1/p + \mathsf{negl}(\lambda)$, the security argument follows.

3.5 Security Analysis

First, it immediately follows by inspection that if all parties play honestly (except for potential early abortion), no party will output $\mathsf{corrupted}$. Therefore, the protocol is detection accurate. We now proceed with the security analysis of the protocol. We do so by exhibiting a simulator Sim which simulates the protocol against a maliciously corrupted B, given access to the ideal functionality $\mathcal{F}_{\mathsf{fe}}$; the security against a corrupted A follows symmetrically. Let (x_A, x_B) be two arbitrary inputs, and let D be a non-uniform PPT distinguisher. We proceed through a sequence of hybrids, denoted H_i; for each hybrid H_i, we denote by HYBRID_i the random variable associated to the transcript and outputs of the parties in an execution of H_i with a corrupted B and inputs (x_A, x_B).

– **Hybrid H_0.** The first hybrid, H_0, corresponds to the real game, where Sim honestly emulates A (using her input x_A) and B is corrupted. By definition, HYBRID_0 is the random variable $\text{REAL}_{\Pi_{\mathsf{sfe}},B}(x_A, x_B, \lambda)$ corresponding to the transcript and outputs of an execution of Π_{sfe} with an honest A and a corrupted B.

Simulator Sim

– **Setup Simulation.** On input the security parameter 1^λ, the simulated setup algorithm computes $(\mathsf{pk}_A, \mathsf{sk}_A) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $(\mathsf{pk}_B, \mathsf{sk}_B) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, $(\mathsf{pp}_A, \tau_A) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1)$, $(\mathsf{pp}_B, \tau_B) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 0)$, and $(\mathsf{pp}, \tau) \leftarrow C.\mathsf{Setup}(1^\lambda)$. It outputs $\mathsf{crs} \leftarrow (\mathsf{pk}_A, \mathsf{pk}_B, \mathsf{pp}_A, \mathsf{pp}_B, \mathsf{pp})$.

Sim stores the trapdoor $(\mathsf{sk}_B, \tau_A, \tau_B, \tau)$. Let sid be the session id for this execution of $\mathcal{F}_{\mathsf{fe}}$.

1. **Initialization.** Sim simulates A as follows: he picks a random index $i_A \xleftarrow{\$} [p(\lambda)]$, a random $x'_A \xleftarrow{\$} \{0,1\}^{2\lambda}$, a random coin r_A for the oblivious transfer, and a random permutation π_A of $[p(\lambda)]$. He computes $E_A \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_A, (i_A, r_A, \pi_A, x'_A))$ and $(c_{1,A}, o_{1,A}) \leftarrow \mathsf{OT}_1(\mathsf{pp}_A, i_A; r_A)$, picks a random $\mathsf{com}_A \in \mathcal{C}$, and sends $(E_A, c_{1,A}, \mathsf{com}_A)$ to B. Upon reception of $(E_B, c_{1,B}, \mathsf{com}_B)$ Sim starts a timer and does the following:
   – he computes $(i_B, r_B, \pi_B, x'_B) \leftarrow \mathsf{Dec}(\mathsf{sk}_B, E_B)$;
   – he computes $i'_B \leftarrow \mathsf{SenderSim}(\tau_B, c_{1,B})$, and set $i^* \leftarrow \pi_A(i'_B)$;
   – he computes $x_B \leftarrow C.\mathsf{Extract}(\tau, \mathsf{com}_B)$, sends $(\mathsf{input}, \mathsf{sid}, x_B, A, B)$ to $\mathcal{F}_{\mathsf{fe}}$ on behalf of the ideal B, and waits for the answer $(\mathsf{ready}, \mathsf{sid}, A, B)$ from $\mathcal{F}_{\mathsf{fe}}$.
   – If B aborts in this phase, Sim sends $(\mathsf{abort}, \mathsf{sid}, A, B)$ to $\mathcal{F}_{\mathsf{fe}}$ on behalf of the ideal B.
2. **Fair Exchange.** For $j = 1$ to $p(\lambda)$, Sim plays as follows:
   – If $j \neq i^*$, and B did not abort, Sim sends $c_{2,A,j} \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, \pi_A(j), 0^\lambda)$.
   – If $j < i^*$ and B aborts, Sim sends $(\mathsf{abort}, \mathsf{sid}, A, B)$ to $\mathcal{F}_{\mathsf{fe}}$ on behalf of the ideal B.
   – if $j = i^*$ and B did not abort in the previous rounds, Sim sends $(\mathsf{proceed}, \mathsf{sid}, A, B)$ on behalf of the ideal B, and wait until he receives x_A from $\mathcal{F}_{\mathsf{fe}}$. He computes $d_A \leftarrow C.\mathsf{Equivocate}(\tau, \mathsf{com}_A)$ and sends $c_{2,A,i^*} \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, \pi_A(j), x'_A \oplus (d_A || x_A))$.
3. **Reveal.** If B did not abort during the fair exchange phase, then Sim received x_A from $\mathcal{F}_{\mathsf{fe}}$ at round i^*. Sim sends (x_A, π_A) to B.

Fig. 3. Simulator Sim for the protocol Π_{sfe} against a maliciously corrupted B. Sim is given access to the ideal functionality $\mathcal{F}_{\mathsf{fe}}$.

- **Hybrid** H_1**.** In hybrid H_1, instead of generating $(\mathsf{pp}_A, \tau_A) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 0)$ as in H_0, (pp_A, τ_A) is generated as $(\mathsf{pp}_A, \tau_A) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1)$ in H_1; the rest of the protocol remains unchanged. By the indistinguishability of the modes of the simulatable OT scheme, we have $|\Pr[D(\text{HYBRID}_0) = 1] - \Pr[D(\text{HYBRID}_1) = 1]| = \mathsf{negl}(\lambda)$.
- **Hybrid** H_2**.** In hybrid H_2, Sim extract $i'_B \leftarrow \mathsf{SenderSim}(\tau_B, c_{1,B})$, using the simulator for the sender simulatability of the oblivious transfer, and computes $i^* \leftarrow \pi_A(i'_B)$ (which is the index of the round at which the OT message that B can decode will be sent). At every round $j \neq i^*$, Sim sends $c_{2,A,j} \xleftarrow{\$} \mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, \pi_A(j), 0^\lambda)$. By the sender simulatability property, the simulated $c_{2,A,j}$ for $j \neq i^*$ are statistically indistinguishable from honest $c_{2,A,j}$, hence we have $|\Pr[D(\text{HYBRID}_1) = 1] - \Pr[D(\text{HYBRID}_2) = 1]| = \mathsf{negl}(\lambda)$.
- **Hybrid** H_3**.** In hybrid H_3, Sim does not compute (com_A, d_A) as $(\mathsf{com}_A, d_A) \xleftarrow{\$} C.\mathsf{Commit}(\mathsf{pp}, x_A)$ anymore; instead, she picks a random $\mathsf{com}_A \xleftarrow{\$} \mathcal{C}$ from the commitment space, and set $d_A \leftarrow C.\mathsf{Equivocate}(\tau, c, m)$. By the equivocability of the commitment scheme, we have

$$|\Pr[D(\text{HYBRID}_2) = 1] - \Pr[D(\text{HYBRID}_3) = 1]| = \mathsf{negl}(\lambda).$$

- **Hybrid** H_4**.** In hybrid H_4, Sim now computes E_A as $\mathsf{Enc}(\mathsf{pk}_A, (i_A, r_A, \pi_A, x'_A))$ for a uniformly random $x'_A \xleftarrow{\$} \{0,1\}^{2\lambda}$. At round $j = i^*$, he computes $c_{2,A,j} \leftarrow \mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, \pi_A(j), (d_A || x_A) \oplus x'_A)$. This hybrid is perfectly indistinguishable from the previous one: $\Pr[D(\text{HYBRID}_3) = 1] = \Pr[D(\text{HYBRID}_4) = 1]$.
- **Hybrid** H_5**.** Hybrid H_5 corresponds to the simulation described on Figure 3; therefore, $\text{HYBRID}_5 = \text{IDEAL}_{\mathcal{F}_{\mathsf{fe}}, \mathsf{Sim}}(x_A, x_B, \lambda)$. Below, we analyze the advantage of D in distinguishing HYBRID_4 from HYBRID_5.

We now analyze the advantage of D in distinguishing HYBRID_4 from HYBRID_5. First, note that the only difference between the transcripts produced by Sim in H_4 and H_5 (setting the outputs aside) is the way c_{2,A,i^*} is computed; Sim computes all other messages identically in H_4 and H_5. Let us denote $c^* \leftarrow c_{2,A,i^*}$ for notational simplicity. In both hybrids, c^* is constructed as $\mathsf{OT}_2(\mathsf{pp}_B, c_{1,B}, i^*, (d_A || x_A) \oplus x'_A)$; the only difference lies in the way x_A is determined by Sim: in H_4, Sim knows A's input x_A from the start, while in H_5, Sim obtain x_A by extracting x_B from com_B (using the trapdoor τ) and sending it to $\mathcal{F}_{\mathsf{fe}}$ on behalf of the ideal B, until he receives x_A from $\mathcal{F}_{\mathsf{fe}}$. By definition of $\mathcal{F}_{\mathsf{fe}}$, independently of whether Sim extracted the correct input x_B from B, it always hold that the value x_A sent by $\mathcal{F}_{\mathsf{fe}}$ is exactly A's input. Therefore, the transcript produced by Sim is always exactly the same in both H_4 and H_5, so we can upper bound the probability that D distinguishes HYBRID_4 from HYBRID_5 by the probability that the (simulated) output of A in H_4 differs from the output of the ideal A in H_5. We analyze this probability below.

For $j = 0$ to $j = p + 1$, let Abort_j denote the event "B aborts at round j" (Abort_0 is the event of B aborting in the initialization; Abort_{p+1} is the event of B not aborting). Let $\mathsf{NotCorrupt}$ denote the event of Sim not outputting $\mathsf{corrupted}_B$

in H_4. Let y_A^4 denote Sim's output in H_4, and let y_A^5 denote the output of the ideal A in H_5.

Claim. It holds that

$$\Pr[y_A^4 \neq y_A^5 \mid \mathsf{Abort}_{p+1} \wedge \mathsf{NotCorrupt}] = \mathsf{negl}(\lambda).$$

This follows immediately from the previous analysis. Indeed, if B does not abort during the protocol and Sim does not output $\mathsf{corrupted}_B$ in H_4, then it holds that $y_A^4 = x_B$, where x_B comes from the pair $x_B||d_B \leftarrow x'_B \oplus K_B$, which Sim obtained by computing $K_B \leftarrow \mathsf{Decode}(\mathsf{pp}_A, c_{2,B,\pi_B(i_A),o_{1,A}})$ and $x'_B \leftarrow \mathsf{ForceDec}(\mathsf{pk}_B, E_B)$. As Sim does not output $\mathsf{corrupted}_B$, it also necessarily holds that $C.\mathsf{Verify}(\mathsf{pp}, \mathsf{com}_B, x_B, d_B) = 1$. By the weak extractability property of C, this implies that the value $C.\mathsf{Extract}(\tau, \mathsf{com}_B)$ computed by Sim in H_5 is equal to x_B, except with probability $\mathsf{negl}(\lambda)$. But this extracted value x_B is the one sent by Sim to $\mathcal{F}_{\mathsf{fe}}$ on behalf of B, hence by the functionality of $\mathcal{F}_{\mathsf{fe}}$, it is also the output y_A^5 of the ideal A, hence the claim.

Claim. It holds that

$$\Pr[y_A^4 \neq y_A^5 \mid \mathsf{Abort}_0 \wedge \mathsf{NotCorrupt}] = \mathsf{negl}(\lambda).$$

This claim also follows immediately, as the output in both H_4 and H_5 is equal to $\perp$ when B aborts during the initialization.

Claim. For any $j \in \{1, \cdots, p\}$, it holds that

$$\Pr[y_A^4 \neq y_A^5 \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] \leq \frac{j \cdot (p+1-j)}{p^2} + \mathsf{negl}(\lambda).$$

Observe that in H_5, the functionality guarantees perfect fairness: either both A and B get their output, or none does. If B aborts at round j, and Sim does not output $\mathsf{corrupted}_B$ in H_4, then the only situation in which the output y_A^4 in H_4 differs from y_A^5 is the following: B already obtained the OT_2 message corresponding to his OT_1 message and will get his output (*i.e.*, $\pi_A(i_B) \leq j$), but A did not (*i.e.*, $\pi_B(i_A) \geq j$) – this corresponds to the situation where fairness fails to hold in H_4. Let us denote $\mathsf{BadEvent}_j$ this event. In all other situations, either $y_A^4 = y_A^5 = x_B$, or $y_A^4 = y_A^5 = \perp$. We now analyze the probability

$$\begin{aligned}&\Pr[\mathsf{BadEvent}_j \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}]\\ =&\Pr[(\pi_A(i_B) \leq j) \wedge (\pi_B(i_A) \geq j) \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}].\end{aligned}$$

The core observation toward proving the claim is that, conditioned on B aborting at round $j \leq p$, B *is a T-bounded adversary.* This allows us to invoke the semantic security of the timed-release encryption scheme. Indeed, consider the following hybrids:

- **Hybrid H'_4.** In H'_4, Sim plays exactly as in H_4, except that he computes E_A as a random encryption of a dummy string of the appropriate length.

– **Hybrid H_4''.** In H_4'', Sim plays exactly as in H_4', except that he computes $c_{1,A}$ using the algorithm $\mathsf{ReceiverSim}$ (given by the receiver simulatability property of the OT scheme) with input τ_A.

Note that both H_4' and H_4'' do not correspond to "proper simulations", in the sense that the external distinguisher D will easily distinguish H_4' and H_4'' from H_4 (in particular because he can run in time longer than T). However, they intuitively correspond to partial simulations that cannot be distinguished from H_4 *by* B, who aborts early.

We first observe that the probability of $\mathsf{BadEvent}_j$ happening in H_4' (conditioned on $\mathsf{Abort}_j \wedge \mathsf{NotCorrupt}$) must be essentially the same as the probability of $\mathsf{BadEvent}_j$ happening in H_4, *i.e.*:

$$\begin{aligned}|\Pr[\mathsf{BadEvent}_j \text{ in } H_4 \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] - \\ \Pr[\mathsf{BadEvent}_j \text{ in } H_4' \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}]| = \mathsf{negl}(\lambda).\end{aligned}$$

This follows from the semantic security of the timed-release encryption scheme: as Sim can observe when $\mathsf{BadEvent}_j$ happens, a party B aborting at round j (hence T-bounded) causing $\mathsf{BadEvent}_j$ to happen with significantly different probability in H_4' immediately allows Sim to distinguish with non-negligible probability between the "real" E_A and the dummy one given access to the T-bounded adversary B, contradicting the semantic security of the scheme.

The indistinguishability of H_4'' and H_4' follows directly from the receiver simulatability property of the OT scheme, hence we get:

$$\begin{aligned}|\Pr[\mathsf{BadEvent}_j \text{ in } H_4' \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] - \\ \Pr[\mathsf{BadEvent}_j \text{ in } H_4'' \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}]| = \mathsf{negl}(\lambda).\end{aligned}$$

Now, observe that in H_4'', i_A and π_A are chosen uniformly at random (respectively from $[1,p]$ and from the set of permutations of $[1,p]$) by Sim. Furthermore, in H_4'', B gets *no information* about i_A and π_A (the content of E_A being replaced by a dummy string, and $c_{1,A}$ being simulated without using i_A). This implies that the probability of $\pi_B(i_A) \geq j$ in H_4'' is exactly the probability that a random integer in $[1,p]$ is bigger than j, *i.e.*, j/p. Similarly, the probability of $\pi_A(i_B) \leq j$ in H_4'' is equal to $(p+1-j)/j$. Importantly, both probabilities are also *independent* with each other, and independent of the events $\mathsf{Abort}_j \wedge \mathsf{NotCorrupt}$ (they only depend on random coins of Sim which are kept perfectly hidden from B). We therefore obtain:

$$\Pr[\mathsf{BadEvent}_j \text{ in } H_4'' \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}]| = \frac{j}{p} \cdot \frac{p+1-j}{p},$$

which concludes the proof of the claim. We can now finish the proof of Theorem 9: by the previous claims, using the fact that $\frac{j}{p} \cdot \frac{p+1-j}{p} \leq 1/p$ for every $j \in [1,p]$, we have for every $j \in [0,p+1]$,

$$\Pr[y_A^4 \neq y_A^5 \mid \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] \leq 1/p + \mathsf{negl}(\lambda),$$

which gives via Bayes rule

$$\begin{aligned}
&\sum_{j=0}^{p+1} \Pr[(y_A^4 \neq y_A^5) \wedge \mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] \\
&\qquad = \Pr[(y_A^4 \neq y_A^5) \wedge \mathsf{NotCorrupt}] \\
&\qquad \leq (1/p + \mathsf{negl}(\lambda)) \cdot \sum_{j=0}^{p+1} \Pr[\mathsf{Abort}_j \wedge \mathsf{NotCorrupt}] \\
&\qquad \leq (1/p + \mathsf{negl}(\lambda)) \cdot \sum_{j=0}^{p+1} \Pr[\mathsf{Abort}_j] \\
&\qquad = 1/p + \mathsf{negl}(\lambda).
\end{aligned}$$

We therefore obtain

$$\begin{aligned}
&\Pr[(y_A^4 \neq y_A^5) \wedge \mathsf{NotCorrupt}] \leq 1/p + \mathsf{negl}(\lambda) \\
\implies &\Pr[(y_A^4 \neq y_A^5)] \cdot (1 - \Pr[\mathsf{Corrupt}]) \leq 1/p + \mathsf{negl}(\lambda) \\
\implies &\Pr[(y_A^4 \neq y_A^5)] - (1/p + \mathsf{negl}(\lambda)) \leq \Pr[(y_A^4 \neq y_A^5)] \cdot \Pr[\mathsf{Corrupt}] \\
\implies &\Pr[(y_A^4 \neq y_A^5)] - (1/p + \mathsf{negl}(\lambda)) \leq \Pr[\mathsf{Corrupt}].
\end{aligned}$$

Plugging this last inequality in our initial observation that $\Pr[(y_A^4 \neq y_A^5)] \geq |\Pr[D(\text{HYBRID}_4) = 1] - \Pr[D(\text{HYBRID}_5) = 1]|$, we obtain

$$\Pr[\mathsf{Corrupt}] \geq |\Pr[D(\text{HYBRID}_4) = 1] - \Pr[D(\text{HYBRID}_5) = 1]| - 1/p - \mathsf{negl}(\lambda).$$

By the analysis of $H_0 = \text{REAL}_{\Pi_{\mathsf{sfe}},B}(x_A, x_B, \lambda)$ to H_4, we had

$$|\Pr[D(\text{REAL}_{\Pi_{\mathsf{sfe}},B}(x_A, x_B, \lambda)) = 1] - \Pr[D(\text{HYBRID}_5) = 1]| = \mathsf{negl}(\lambda),$$

which finally gives us

$$\begin{aligned}
\Pr[\mathsf{Corrupt}] \geq |\Pr[&D(\text{REAL}_{\Pi_{\mathsf{sfe}},B}(x_A, x_B, \lambda)) = 1] \\
&- \Pr[D(\text{IDEAL}_{\mathcal{F}_{\mathsf{fe}},\mathsf{Sim}}(x_A, x_B, \lambda)) = 1]| - \frac{1}{p(\lambda)} - \mathsf{negl}(\lambda),
\end{aligned}$$

which concludes the proof of Theorem 9.

3.6 Extensions

We sketch how the protocol Π_{sfe} can be naturally extended to more complex functionalities. Observe that, after the initialization phase, both parties hold (equivocable and weakly-extractable) commitments to the values of their opponent. At this stage, the parties can rely on zero-knowledge proof to prove arbitrary statements of their choice regarding their committed value to their opponent, or execute any two-party computation protocol (satisfying only security

with abort) to guarantee any specific property of the committed value without disclosing them – as long as this phase is completed before a time T elapses. In particular, the parties can for example rely on a generic two-party computation protocol satisfying security with abort to check that the committed value of their opponent verifies correctly with respect to their secret-key of a one-time MAC scheme; the $1/p$-fairness of the resulting scheme follows immediately from the $1/p$-security of Π_{sfe} and the security with abort of the generic two-party computation protocol. This shows that our minimal use of a delayed encryption scheme already suffices to get around the impossibility result of [18], which was established exactly for this primitive.

More generally, the two parties can compute arbitrary functionalities with $1/p$-fairness as follows: first: they execute a generic two-party computation protocol which computes a modified functionality, whose output is a *random xor sharing* of the desired output. This protocol only needs to be secure with abort, since no early abortion during its execution can allow the adversary to learn the output, each share revealing nothing about the output. Then, the two parties execute the protocol Π_{sfe} on those outputs shares, and rely on a generic zero-knowledge proof system (before the start of step 2) to demonstrate that the value committed in the initialization phase is the correct output of the modified functionality on their private input. After completion of the protocol Π_{sfe}, both parties reconstruct the output by XORing the exchanged values. It immediately follows from the security-with-abort of the generic two-party protocol, the security of the zero-knowledge proof system, and the $1/p$-security of Π_{sfe}, that the resulting protocol does $1/p$-securely compute the desired functionality.

3.7 Other Applications of Partially-Fair Exchange

The partially-fair exchange mechanisms proposed here can find application in other contexts, for example contract signing. Note that there are some interesting issues of incentives here: in the application to PAKEs the attacker wants to provide the correct confirmation value if possible. There is no incentive for him to provide an "invalid" V value.

This is in contrast to, say, contract signing, where each party may well be incentivised to submit invalid signatures. The standard way to handle this is to introduce optimistic protocols: that will invoke a judge or TTP in the event of problems. In this context it is not clear that our partially-fair exchange construction provides any advantage over such optimistic protocols.

A protocol may satisfy fair-exchange and still admit the possibility that at some point in the execution one party has the power to determine whether or not the protocol will terminate successfully, and furthermore be able to prove this to a third party. This may be an issue if this party can use this a leverage to bargain more favourably with the third party. *Abuse-freeness* seeks to counter this by requiring that neither party can demonstrate to a third party that they can control whether or not the protocol will complete. Our SFE construction denies the parties knowledge of the point at which they acquire the desired terms and so could provide the basis for abuse-freeness.

4 Conclusions

In this paper we have presented a new construction for partially-fair exchange, based on oblivious transfer and a delay primitive. While the original motivation came from making PAKE protocols "auditable", the protocol we obtain achieves a very interesting property, which we believe to be of broader independent interest: it provides the first generic method to achieve $1/p$-fairness, for every functionality (without limitations on the input size or the output size as in the paper of Katz and Gordon), while guaranteeing at the same time a polynomial upper bound on the running time of all parties. In particular, this means that the fairness guarantee that we obtain does also extend to the scenarios where the adversaries might be considerably more powerful than the honest parties; to our knowledge, every previous paper achieving some form of fairness for all functionalities (without the help of a trusted third party, or a smart contracts) could not guarantee this highly desirable property. In addition, our protocol is not purely of theoretical interest: it remains practical, and its building blocks can be instantiated efficiently from a variety of standard cryptographic assumptions.

References

1. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Heidelberg, February 2007.
2. Gildas Avoine and Serge Vaudenay. Optimistic fair exchange based on publicly verifiable secret sharing. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 74–85. Springer, Heidelberg, July 2004.
3. Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 589–590. Springer, Heidelberg, August 1990.
4. Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.
5. Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
6. Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, Heidelberg, August 2013.
7. Manuel Blum. How to exchange (secret) keys (extended abstract). In *15th ACM STOC*, pages 440–447. ACM Press, April 1983.
8. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Heidelberg, August 2000.
9. Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 93–111. Springer, Heidelberg, August 2000.

10. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.
11. Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 200–217. Springer, Heidelberg, May 1994.
12. Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. Optimistic fair exchange in a multi-user setting. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 118–133. Springer, Heidelberg, April 2007.
13. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982.
14. Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 135–155. Springer, Heidelberg, August 1988.
15. Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. *ACM Transactions on Information and System Security*, 9(2):181–234, 2006.
16. Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 77–93. Springer, Heidelberg, August 1991.
17. S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. pages 157–176, 2010.
18. S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1):14–40, January 2012.
19. Joseph Jaeger, Thomas Ristenpart, and Qiang Tang. Honey encryption beyond message recovery security. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 758–788. Springer, Heidelberg, May 2016.
20. Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. *Journal of Cryptology*, 26(4):714–743, October 2013.
21. Eike Kiltz and Hoeteck Wee. Quasi-adaptive NIZK for linear subspaces revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 101–128. Springer, Heidelberg, April 2015.
22. Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 30–41. ACM Press, November 2014.
23. Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 195–206. ACM Press, October 2015.
24. Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 252–267. Springer, Heidelberg, March 2010.
25. Matt Lepinski, Silvio Micali, Chris Peikert, and abhi shelat. Completely fair sfe and coalition-safe cheap talk. In Soma Chaudhuri and Shay Kutten, editors, *23rd ACM PODC*, pages 1–10. ACM, July 2004.
26. Andrew Y. Lindell. Legally-enforceable fairness in secure two-party computation. In Tal Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 121–137. Springer, Heidelberg, April 2008.

27. Michael Luby, Silvio Micali, and Charles Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *24th FOCS*, pages 11–21. IEEE Computer Society Press, November 1983.
28. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.
29. Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 87–105. Springer, Heidelberg, May 2003.
30. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
31. A. W. Roscoe and Peter Y. A. Ryan. Auditable PAKEs: Approaching fair exchange without a TTP. In *Security Protocols XXV - 25th International Workshop, Cambridge, UK, March 20-22, 2017, Revised Selected Papers*, pages 278–297, 2017.