General Partially Fair Multi-Party Computation with VDFs

Bolton Bailey[1], Andrew Miller[1], and Or Sattath[2]

[1]University of Illinois Urbana-Champaign
[2]Ben-Gurion University of the Negev

October 8, 2022

Abstract

Gordon and Katz, in [GK10], present a protocol for two-party computation with partial fairness which depends on presumptions on the size of the input or output of the functionality. They also show that for some other functionalities, this notion of partial fairness is impossible to achieve. In this work, we get around this impossibility result using verifiable delay functions, a primitive which brings in an assumption on the inability of an adversary to compute a certain function in a specified time. We present a gadget using VDFs which allows for any MPC to be carried out with $\approx 1/R$ partial fairness, where R is the number of communication rounds.

1 Introduction

A desirable property for a multi-party computation is *fairness* - that either all parties should receive their output or none should. Unfortunately, by a result of Cleve [Cle86], it is impossible in general to achieve fairness with all-but-negligible probability with a dishonest majority. As an alternative, the literature has considered protocols that achieve fairness with inverse-polynomial probability, which is known as *partial fairness*.

The reason for the impossibility of general complete fairness might be described as follows: There is always some communication round in which some party gains, for the first time, knowledge of their output. If an adversary knows which round this is, they can cease communication in that round, obtaining their output but without the honest party having learned their output. Gordon and Katz [GK10] mitigate this as much as possible by obscuring the round in which this critical knowledge is passed. This works by adding a second phase to the end of the protocol where secret shares to potential outputs are passed one at a time, and at some point these shares reveal the true outputs. This does not make the protocol completely fair - the adversary may still randomly guess the round - but if the adversary cannot identify a true output just by looking at it, they will send the honest party output share before they realize that this contains the critical information. This at least gives a probability of $1/R$ that this naive attack will succeed, where R is the number of rounds of share-passing.

Unfortunately, if the functionality to be computed is complicated enough, this obscuring process is again impossible. It might be that the adversary *can* inspect the information revealed in each round and determine if it matches what the output is expected to look like. Gordon and Katz make this rigorous in their Theorem 11, which shows that for a particular functionality, there is no protocol to compute it even with partial fairness.

In this work, we show how Verifiable Delay Functions (VDFs) [BBBF18] can be used to get around this obstacle. The conceit of a delay function is that it acts essentially as a random oracle, but one which can only be computed in time t, even by an adversary with access to parallel computation resources. VDFs make this more convenient by also providing a way of quickly proving that the output of the VDF indeed comes from a given input. VDFs have been proposed for applications requiring an unbiasable source of randomness, such as leader selection in consensus mechanisms [CPNX20].

Here, we use VDFs to create a protocol, similar to [GK10], but that works for general multi-party functionalities. The main idea of our protocol is to let the random choice of the critical round depend on VDF outputs, which means that it cannot be determined until after all the communication of the protocol is over. This ensures that the best an adversary can do is to arbitrarily guess a round to halt in, and hope that this round ends up being the critical round. Still, by computing the VDF, an honest party can always eventually decrypt their output in the event of an attack.

1.1 Prior work

A variety of approaches to fair Multi-Party Computation have been put forward over the years.

As we have mentioned, our approach uses time-delay encryption, a technique which was perhaps first brought to the literature by Rivest et al. in [RSW96]. This is not the first time time-delay encryption has been used for MPC; the approach of Pinkas in [Pin03] makes use of a variant of the "timed commitments" of [BN00] for fairness in general two-party computation. In that case however, Pinkas does not try to achieve a protocol that runs in a fixed amount of time polynomial in the security parameter. Rather, that work obtains a protocol where the parties create timed commitments that reveal after shorter and shorter time periods, walking down from an exponential amount of time. With this, the honest party does an amount of work proportional to the adversary's work to obtain their input. The adversary can choose how much work this is though, and [GK10] points out that if the adversary knows the honest party's computational limits they may choose an amount that is barely higher than the honest party is willing to invest.

Another approach, from [BG89, GL90], takes the tack of revealing some information about the output in each round - but only a limited amount. In each round, parties learn a bit which is obscured by noise; as the protocol progresses they gain greater and greater confidence in the value of their output. If an adversary stops halfway through, then both parties may have a guess about the value of their output, but no certainty. This setting is undesirable when the adversary is happy learning even a little information about their secret, but where we personally only feel comfortable if we obtain full knowledge of our secret. In fact, the Gordon and Katz protocol has a similar issue, in that it is not obvious to the honest party whether the adversary has succeeded in violating fairness when they stop early.

In Table 1, we lay out the performance of these approaches.

	Prob. adversary output correct	Prob. honest output correct	Honest work to recover output	General functionalities	Correct outputs obvious?
[Pin03]	1	1	$O(2^{R-r})$	Yes	Yes
[BG89]	$\approx \Phi(k\sqrt{r+1})$	$\approx \Phi(k\sqrt{r})$	O(1)	Yes	No
[GK10]	$(r+1)/R$	r/R	O(1)	No	No
Ours	$(r+1)/R$	r/R	O(R)	Yes	Yes

Table 1: Performance of various approaches to general fair MPC, when the adversary halts in round r out of R. Φ represents the Gaussian CDF, and k is a constant chosen by the protocol designer.

Our approach might be seen as a synthesis of the Gordon and Katz approach with the time-delay technique of Pinkas. Rather than letting the time-delay hide the output all on its own, we can add the time-delayed data to the secret shares. We gain the ability to deal with general outputs in the functionality, and we only need to delay the data for the length of the communication phase.

2 The Protocol

As a preliminary, we note that we use the "trapdoor" VDF definition of Wesolowski [Wes19] rather than the original formulation of [BBBF18] (a choice we explain in subsection 2.2). This defines *Verifiable Delay Function* as a quadruple of algorithms:

$$\mathsf{VDF.KeyGen}() \to (pk, sk), \mathsf{VDF.Trapdoor}(sk, x, \Delta) \to (y, \pi),$$

$$\mathsf{VDF.Eval}(pk, x, \Delta) \to (y, \pi), \mathsf{VDF.Verify}(x, y, \pi) \to \{0, 1\}$$

The VDF.KeyGen function generates a public and a secret key. The VDF.Trapdoor and VDF.Eval functions can be modeled as a random oracles which computes y as a function of an input x. VDF.Eval is meant to take time Δ for the adversary to compute (even with a parallel computer), while VDF.Trapdoor computes the same y, but which uses the secret key sk, and may take less time. Both additionally output π, a quickly-verifiable "proof". The $\mathsf{VDF.Verify}$ function can then take such a proof along with x, y, and outputs 1 if this is the result of a valid call to $\mathsf{VDF.Trapdoor}$ or $\mathsf{VDF.Eval}$, otherwise it returns 0.

We consider the following protocol for a set of N parties, with inputs $x_i \in X$, who wish to securely compute functions $f_i : X^N \to \{0,1\}^n$ with no other information being leaked, and with fairness $1/R$.

1. Each party sets up VDF parameters and gets the keys $pk_i, sk_i \leftarrow \mathsf{VDF.KeyGen}()$ and broadcasts the public key to all other parties. The parties individually choose random values v_i to input to their VDF. They compute the VDF on these values with $\Delta = \delta R$, where δ is the period of time taken up by one round of communication. They get an output in the form of a uniformly random pair of a share randomness and a round randomness $s_i, r_i \in \{0,1\}^n \times [R]$, as well as a VDF proof π_i output $((s_i, r_i), \pi_i) := \mathsf{VDF.Trapdoor}(sk, v_i, \Delta)$. They broadcast commitments to the v_i values to the other parties.

2. The parties supply their inputs x_i, along with s_i, r_i, to a secure-with-abort MPC protocol which computes the following:

   (a) The MPC computes the critical round as a mixture of the individual round randomnesses $r^* := \left(\sum_{i \in N} r_i\right) \mod R$.

   (b) The MPC computes a encryption pad s as a mixture of the individual share randomnesses $s := \bigoplus_{i \in N} s_i$.

   (c) For each pair of parties i, j, and each round r, the MPC sends a message $m_{i,j,r}$ to party i. This message is intended to be sent from party i to party j in the rth round of the reveal phase below:

      - For $r = r^*$, the the messages $(m_{i,j,r^*})_{j \in [N]}$ form a random secret share of party j's output OTP encrypted with the VDF pad s with the $\bigoplus_{j \in [N]} m_{i,j,r^*} = s \oplus f_j(\bar{x})$.
      - If $r \neq r^*$, the value is chosen uniformly at random $m_{i,j,r} \leftarrow \{0,1\}^n$.

   (d) Each party j additionally receives from the functionality a vector of commitments to all the entries of all messages $m_{i,j,r}$ intended for them, and party i receives an opening for that commitment. Parties additionally receive commitments to s_i, r_i that party i can open.

3. The parties, using the v_i, s_i, r_i, π_i values they hold, complete zero-knowledge proofs for each other that $\mathsf{VDF.Verify}$ yields a pass on these values and that the commitments to these values match the commitments issued by the MPC.

4. They then start the VDF timer and open their commitments to v_i to each other, and the reveal phase begins.

   (a) In round r of the reveal phase, if all previous rounds have been successful, and the VDF timer has not elapsed, each party i opens the commitment to each $m_{i,j,r}$ they hold to the corresponding j. Party j verifies the opening, and halts if the verification fails.

5. After the reveal phase ends, all parties compute $\mathsf{VDF.Eval}$ on the v_i values to obtain s_i, r_i, and from this, s, and r^*. (Optionally, instead, parties can open their commitments to the s_i, r_i to save each other the trouble of recomputing the VDF).

6. If a party j received all messages m_{i,j,r^*} in the r^*th round of the reveal phase, then that party outputs $f_j(\bar{x}) = s \oplus \bigoplus_{j \in [N]} m_{i,j,r^*}$.

2.1 Partial fairness argument

We now argue that the above protocol meets our design criteria:

Theorem 1. *The protocol presented above is a secure $1/R$-fair MPC protocol.*

Proof. We first show that the protocol is secure: An adversary controlling some set of parties cannot learn the inputs or outputs of any parties not in the set, beyond what can be learned from the inputs and outputs of parties in the set. This follows from the fact that for any honest party j, the adversary will never receive m_{j,j,r^*}. This message is a component of the secret share hiding $f_j(\bar{x})$. Without it, all other messages m_{i,j,r^*} are jointly distributed uniformly at random, and so they provide no information about $f_j(\bar{x})$.

To see that the protocol is $1/R$-fair, observe first that no party can successfully send a message in the reveal phase other than what the protocol tells them to, since all data sent takes the form of opening a commitment previously provided by the MPC. Since at least one party i must be honest, parties do not possess s_i or r_i during the reveal phase, so the adversary's best guess distribution over s and r^* is uniform until the phase is complete. Lack of knowledge of s means that all $m_{i,j,r}$ are independent and uniformly randomly distributed from the adversary's perspective. Thus, we may assume without loss of generality that the adversary's strategy is to select some round $r^{\mathcal{A}}$ to halt in, so that they receive only the messages up to and including that round, and honest parties receive only message from before that round. If $r^{\mathcal{A}} = r^*$ (which may happen with probability at most $1/R$) then the adversary learns their outputs while the honest parties do not, but if $r^{\mathcal{A}} < r^*$, no party learns their output, and if $r^{\mathcal{A}} > r^*$, all parties learn their outputs. □

Note that we must hide both the value of r^* and s within the VDF output: r^* because it is critical that the adversary not know the precise round to stop, and s because if it is revealed early, a party might compute $s \oplus \bigoplus_{j \in [N]} m_{i,j,r^*}$ at the beginning of each communication round to see if it matches what the output is expected to look like, and then only send their data in the latter half of the round if it does not match.

It's worth explaining how this construction is possible, given Theorem 11 from [GK10]. That theorem presents a particular functionality based on verifying a MAC and shows that this functionality cannot be computed with partial fairness. The proof of the theorem constructs an adversary which checks the putative output from each round (that is, the output if the other party were to halt after that round), and halts when the output passes the MAC verification.

The time-delay encryption model turns out to be critically important for circumventing this attack. In the standard model, it is always possible for the adversary to compute the putative output from each round instantly, and make decisions on that basis. But with our protocol, this involves computing $\mathsf{VDF.Eval}$, so it would take up all the time remaining in the reveal phase. Honest parties only continue the reveal phase while the VDF timer has not elapsed, so any response taking this long will always be ignored, and running this attack on our protocol would essentially be the same as halting.

2.2 Choice of VDF

We note that the above protocol requires the use of the VDF in a non-black-box way: While the $\mathsf{VDF.Eval}$ function is computed by individual parties, the $\mathsf{VDF.Verify}$ function must be evaluated within some zero-knowledge proof system. This proof system may be interactive or non-interactive, but it must be zero-knowledge to hide the value of the s_i, r_i outputs. We could modify the formalism to make a VDF-with-commit, in which $\mathsf{VDF.Verify}$ takes a commit to the VDF output rather than the output itself, and then this could be used as a black-box.

Indeed, recalling that the earliest design for a VDF was simply an iterated hash function evaluated within a SNARK, one might describe the protocol without any VDF at all, and use instead any time delay function with the SNARK. We feel it makes sense to characterize the primitive as a VDF, though, as many VDF constructions have more efficient proof systems than the naive approach of running a generic zk-SNARK over the evaluation function.

To elaborate on why the Wesolowski VDF-with-trapdoor formulation is slightly superior to the Boneh et al. formulation for our purposes, we note that if the parties call $\mathsf{VDF.Eval}$ in step 1 rather than $\mathsf{VDF.Trapdoor}$, we would expect step 1 to take Δ time, as long as the entire reveal phase. The VDF of [Pie18], for example (like the constructions in [Wes19, RSW96]), is based on RSA groups of unknown order pq, and uses the evaluation function $x \mapsto x^{2^T} \mod pq$: If a party knows the factors of the modulus, they may compute the output of the VDF by evaluating $2^T \mod \phi(pq)$, and as a result it is possible for $\mathsf{VDF.Trapdoor}$ to be computed in less time than $\mathsf{VDF.Eval}$ ($O(\sqrt{T} \log T)$ time, in fact). This is useful for our protocol, since it may avoid the need for any party to undertake any slow VDF computation at all, approximately halving the protocol runtime.

2.3 Reducing VDF Usage

In service of simplicity, the protocol above is described symmetrically, so that all parties take the same actions in every step of the protocol. If we were to relax this, could we improve the protocol, perhaps in a way that allows the participants to compute fewer VDFs all told?

In fact this is possible. Considering the two-party setting, with two participants Alice and Bob, we describe a modification of the protocol where only Alice initializes a VDF, but the protocol still remains partially fair: In Step 1, let Alice alone create and compute her VDF, and let the outputs of this evaluation be themselves r^* and s. In the reveal phase, instead of letting both parties broadcast their round r messages simultaneously, let Alice send her message first, and Bob send his message second (this might be seen as switching the communication model and doubling the number of rounds). While Alice will be able to decrypt her result immediately after Bob sends his message in round r^* and potentially determine if this is the critical round from its contents, this does not help Alice break the fairness, since she has already sent Bob's critical round message.

A more extreme approach to "reducing" the number of VDF computations might be to take a different approach of evaluating a single VDF within the MPC, or of passing inputs to a single party to run the VDF evaluation in a homomorphic encryption. While these approaches would technically require only polynomial computation in the security and fairness parameters, in practice, the slowdown from computing a VDF as a circuit within another protocol would likely be prohibitive.

3 Rational Security

A deficiency of the above protocol is that its fairness is not only partial but unincentivized. By the end of the final round, each participant has received all of the messages that their counterparties intend to send, and no action of theirs at that point can change what these messages contain. Since we are studying fairness, it is natural to assume any malicious parties have a goal of preventing honest parties from obtaining their output. This being the case, one might expect the malicious parties to always refuse to send their last message.

Can this be avoided? Similar to Groce and Katz [GK12], we show that if we move to a setting where a participant's preference for receiving their own inputs outweighs their preference over others' outcomes, the answer is yes. We outline a modification of the protocol to ensure better delivery, based on the idea that the participants will act rationally according to a linear utility function on the delivery of the outputs to themselves and to others. This makes it possible to motivate malicious actors to continue the protocol by making their reward contingent on them continuing to send messages.

For simplicity, we consider the setting of two parties where one is honest and the other is dishonest, but where the dishonest party adversary assigns a utility a to receiving their own output and utility -1 for the honest party receiving their output. We will also assume the dishonest party has some small incentive to complete the protocol honestly c (we might think of this as corresponding to the cost of developing a hacked client).

Rather than computing a uniformly random r^* as $\left(\sum_{i \in N} r_i\right) \mod R$, we let the r_i pseudorandomly generate a draw from an *arbitrary* distribution. We can keep the protocol ϵ-fair as long as the weight on each round is at most ϵ. But we can also make continuing the protocol incentive compatible for the adversary: As long as $1 < a$, we will see that if we let the probability decay exponentially with rate $\frac{1}{a}$ as we approach the final round, the adversary will continue to send.

Theorem 2. *Given $1 < a, 0 < c$ and $0 < \epsilon < 1$, let R_a be the smallest n such that $ca^{n+1} > \epsilon$, and let R_ϵ be minimal such that $R_\epsilon \epsilon \geq 1 - \sum_{n=0}^{R_a} ca^n$. Then the optimal number of rounds is $R_\epsilon + R_a$, and this is met by the following distribution on r^*:*

- *For $1 < i \leq R_\epsilon$, we have $\Pr[r^* = i] = \epsilon$.*
- *For all $i > R_\epsilon$ we have $\Pr[r^* = i] = c\,(a)^{R-i}$*
- *For $i = 1$, we have $\Pr[r^* = i] = 1 - R_\epsilon \epsilon - \sum_{n=0}^{R_a} ca^n$*

Proof. In order for the protocol to be incentive compatible, the adversary must prefer completing the protocol to halting in any round. If the adversary halts in round i, then their utility is $a \Pr[r^* \leq$

$i] - \Pr[r^* < i]$. Incentive compatibility is therefore equivalent to the following condition on the distribution holding for all $i \leq R$.

$$a \Pr[r^* \leq i] - \Pr[r^* < i] \leq a - 1 + c$$

Rewriting this in terms of $\Pr[r^* > i]$ and $\Pr[r^* = i]$, we see this is equivalent to the condition

$$\Pr[r^* = i] \leq (a-1) \Pr[r^* > i] + c. \tag{1}$$

From fairness, we get the condition

$$\Pr[r^* = i] \leq \epsilon. \tag{2}$$

We see that the distribution we have described satisfies these constraints: Equation 1 is obeyed with equality for $i > R_\epsilon$, and holds for all $i \leq R_\epsilon$ by virtue of the fact that

$$\Pr[r^* = i] \leq \epsilon < (a-1) \Pr[r^* > R_\epsilon] + c \leq (a-1) \Pr[r^* > i] + c,$$

where the second inequality comes from the choice of R_a. Equation 2 holds with equality on $i < R_\epsilon$, on $i > R_\epsilon$ by choice of R_a, and on $i = R_\epsilon$ by choice of R_ϵ.

To see that our distribution is round-optimal, note that equation 1 for $i = R$ gives us $\Pr[r^* = R] \leq c$. By induction on $R - i$, we then get that for any $1 \leq i \leq R$, we actually *must* have

$$\Pr(r^* = i) \leq ca^{R-i} \tag{3}$$

Thus, the distribution we have described maximizes the value of $\Pr(r^* = R - n)$ for all $n < R$. With fewer rounds, these probabilities would not sum to 1, so $R_\epsilon + R_a$ must be the the optimal number of rounds.

$\square$

Just as in the Groce and Katz work, we see that an exponentially decaying distribution on the critical round allows the protocol to be cognizant of the motivations of the players. It is interesting to see that this can be combined with the simple Gordon and Katz approach to produce a distribution that is partly uniform, partly geometric, and which respects both definitions of fairness.

4 Conclusion

We have presented a VDF-based protocol for $1/R$-fair MPC supporting arbitrary functionalities, secure against any number of corruptions. We have also discussed considerations for modifying this protocol, both to reduce the number of times the VDF evaluation function is called, as well as to make the protocol more effective in the presence of a rational adversary.

A possible avenue for future work might be to examine the communication model with more care. For the purposes of stating the protocol and the proving of 2, it was convenient to deal with a synchronous communication protocol in which all parties send their messages for a given round at once. But this leaves an honest party who is sending their message at the start of a round without the knowledge of whether their counterparties will send to them, and so they have less information than if we proceeded with alternating communication rounds. We saw that we can get rid of at least one of the VDF evaluations if we switch to this model - is it possible to improve this more?

5 Acknowledgements

The authors would like to thank Or Sattath for several stimulating discussions during the formulation of the main ideas of this paper.

This material is based upon work supported by the National Science Foundation under the Graduate Research Fellowship Program with Grant No. DGE – 1746047.

References

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *Conference on the Theory and Application of Cryptology*, pages 589–590. Springer, 1989.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[CPNX20] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR)*, 53(3):1–43, 2020.

[GK10] S Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 157–176. Springer, 2010.

[GK12] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 81–98. Springer, 2012.

[GL90] Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In *Conference on the Theory and Application of Cryptography*, pages 77–93. Springer, 1990.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[Pin03] Benny Pinkas. Fair secure two-party computation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–105. Springer, 2003.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.