For example, the notation for the schemes found in Appendix B would have been useful prior to the definitions of a C-TLP and SO-PoR. If we move the notation table to the paper’s main body, then the paper would exceed the page limit.

In addition, I would have liked some intuition after Theorem 1 and 2 in the main text to give an informal treatment of how the authors analysed the respective constructions. Done- Both theorems will have proof outline.  
  
I felt that the work on "Generic Constructions of Incremental and Homomorphic Time-Release Encryption" by Peter Chvojka, Tibor Jager, Daniel Slamanig and Christoph Strieks, should have been cited, as the core idea of solving TLPs incrementally is the same. Done- now the paper has been analysed and cited.  
  
Using C-TLPs instead of VDFs is somewhat limited in application. A better motivation for exploring this would be a good addition to the paper.  
  
Justification of recommendation  
-------------------------------  
A multi-lock time-lock puzzle accompanied by a publicly verifiable algorithm (C-TLP) is a useful primitive to be used as a building block in blockchain-based solutions. The SO-PoR is a new solution providing desirable properties for applications of PoR, such as real-time detection and fair payment. These properties have previously not been achieved simultaneously in publicly verifiable PoR. The authors provide a good understanding of the current literature and they demonstrate their constructions, analysis and the applications of such schemes clearly.  
  
  
\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*  
  
  
Review #115B  
===========================================================================  
  
Overall merit  
-------------  
3. Weak accept  
  
Paper summary  
-------------  
This work considers the setting of a single entity (a server, for example) that has to solve several instances of a time-lock puzzle scheme, where the solutions to different instances are to be revealed at different points in time. This is definitely an interesting problem, as motivated by the work in the settings of (a) mass release of confidential documents over time, (b) gradually revealing multiple secret keys, etc. The work points out the drawback to applying several independent instances of time-lock puzzles towards this goal. Indeed, if one were to create several independent instances of time-lock puzzles that would reveal their solutions after designated periods of time, to obtain the solutions at the said times, the solver must begin solving all instances simultaneously and continue solving them in parallel. This imposes severe demands computationally as well as in terms of the parallelism required, and clearly does not scale. Undoubtedly, a more elegant and fitting solution to this problem would be to chain the different instances together, in order of their reveal times. For instance, suppose I have three puzzles P\_1, P\_2 and P\_3 that would be solved in times t\_1 < t\_2 < t\_3 respectively, I would hope to chain these puzzles so that I work sequentially for time t\_1, solve P\_1, and then for time t\_2 - t\_1 and solve P\_2 and finally t\_3 - t\_2 and solve P\_3. This eases the demands on parallelism, reuses computation and hence scales well with the number of puzzles. Towards this end, this work formally defines a primitive called “multi-instance time-lock puzzle” which allows  
composing a puzzle’s instances, and proposes a candidate construction: “chained time-lock puzzle” (C-TLP).  Finally, it gives applications of this primitive/construction.  
  
While the problem itself is definitely well-motivated, the only drawback is that technically, the first attempt at solving this problem works (for the most part). The authors mention that "While chaining different puzzles may seem a relatively obvious approach to tackle the issues, designing a secure protocol that also can make black-box use of a standard time-lock puzzle scheme, supports public verifiability, and has low costs is challenging." However, it isn't clear what the technical challenges are imho. As it is illustrated in Remark 8, Appendix D, by simply embedding one puzzle into another one, we would not get a securely scheme. As, it allows an adversary to solve 2nd,…, z-th puzzle right after it solves 1st one. Moreover, embedding (a) puzzle and (b) its parameters into another puzzle would affect the puzzle scheme’s efficiency, as the combination unnecessarily increases the size of plaintext that results in higher computation and communication costs.

The idea is essentially to in some sense embed the puzzle that must be solved next inside the previous puzzle. That is, the solution to the first puzzle contains the parameter that defines the second puzzle, and so on. Thus, to solve the second puzzle, to even obtain the second puzzle in its entirety, one must solve the first puzzle. The times for each puzzle are basically the differences as alluded to above (in the work, the authors assume all differences are equal to \Delta, although I presume that this is just a simplifying assumption that can be easily generalized). An additional technical idea is to also publish commitments to the solutions in order to enable fast verification - for instance a hash computation if working with a commitment in the RO model.

The applications to PoR and VDF are also fairly straight-forward. The gains presented in comparison with other works are noteworthy, and expected by design. Overall, I feel that this work provides a thorough presentation and analysis of an important and applicable, yet fairly straight-forward idea.  
  
Comments for author  
-------------------  
The presentation is clear and well-motivated. It might be nice to also present the non-equal time difference case since it seems to be a trivial extension, unless I am missing something.  
  
Justification of recommendation  
-------------------------------  
Overall, I feel that this work provides a thorough presentation and analysis of an important and applicable, yet fairly straight-forward idea.  
  
  
\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*  
  
  
Review #115C  
===========================================================================  
  
Overall merit  
-------------  
3. Weak accept  
  
Paper summary  
-------------  
The authors present a time-lock puzzle construction in which a single puzzle  
produces a sequence of outputs, each released after some delay relative  
to the prior output's being revealed. They then describe how this puzzle  
can be used to construct a proof of retrievability for outsourced storage  
of data, and point out that in some applications their construction can  
be used to improve certain costs compared with a verifiable delay function.  
  
Comments for author  
-------------------  
This is a cool idea, and I appreciated the clarity of the write-up and the  
elegant simplicity of the construction. On that basis alone, this paper  
deserves to be disseminated.  
  
Beyond that, the survey of related work in Appendix A is excellent! Thank  
you very, very much for such a comprehensive write-up. This is valuable  
all on its own.  
  
I do have some concerns about the cost accounting for SO-POR and the comparison  
to VDFs, and there are some general editorial improvements that would make  
this paper even better.  Details below.  
  
# cost accounting for SO-POR / comparison to VDFs  
  
I appreciate the comparison against related work, but I'm not really convinced  
by it. The main reason is, it seems as if the argument with regard to  
efficiency is only concerned with verification cost. But, as the authors  
admit, this work and [5] require the server storing the data to grind  
away at time-lock puzzles, whereas [3] and [53] have no such requirement.  
Surely the client---who is paying the server, presumably, to hold onto the  
data---would also be required to pay for the server's processing time.  And  
once we account for that cost, the advantages of this scheme are considerably  
less obvious.  
  
It seems to me like the reader's question is: if I'm going to pay someone  
to hold onto my data, how much do I have to pay them? And surely you'd have  
to pay them for the CPU the timelock puzzle is occupying just as you'd  
have to pay for the disk space the data is occupying. Right?  
  
It's not totally clear to me how the comparison ought to look---these are  
different quantities, so it'll take some thought to compare them in a way  
that is fair and useful for answering the question above. (For example:  
cost in dollars on AWS?) But leaving it out entirely---indeed, not even  
really acknowledging this as a downside of the proposed scheme---seems  
a bit tendentious. In Section 5.4, we will explicitly state that (in SO-PoR) the client has to compensate the server for solving each puzzle.   
  
On a similar note, in Section 6 when comparing against VDFs, it's important  
to admit that using a VDF has the potentially significant advantage that  
it does not require choosing z ahead of time. Perhaps that's OK in some  
applications, but neglecting to admit it hurts the paper's credibility. We will explicitly highlight that point in section 6: ‘’ in general, VDF’s offer certain features that (C-)TLP schemes do not offer, e.g., in VDF’s parameter z or their outputs do not need to be fixed ahead of time. ‘’

# low-level nits  
  
- I know I'm spoiled, but: page numbers would be nice!  
  
- page 2, "Summary of our contributions": neither real-time detection nor fair  
  payment have been defined at this point, so reader gets confused. Done. Now we have added a short description of each property to the introduction section.  
  
- page 2, "Time-lock puzzles": "The scheme that May proposed lies on" should  
  probably say "relies" rather than "lies" done  
  
- page 3, "Outsourced Proofs of Retrievability": as above, "fair payments"  
  isn't defined here and appears to have a specific meaning. Done. Now we have added a short description of each property to the introduction section.  
  
- a few lines below: "polynomial arithmetics and error-correcting code"  
  should say "arithmetic" and "codes" done  
  
- a few lines below: in what sense is validating VDF outputs a significant  
  computational cost? Do you mean concretely? Compared to what? Yes concretely, in practice. We will clarify it is a concrete cost in the paper.  
  
- page 3, "Blockchain-based PoR": again "high verification overhead" is alluded  
  to here, but it's not clear what it's being compared to. We will clarify it is a concrete cost in the paper.  
  
- page 3, Preliminaries: wow! a notation table is excellent! Every paper should  
  have one.  
  
- page 5, "GenPuz" definition: "takes an input" should say "takes as input" done  
  
- page 5, Definition 3: small question: would it be possible to give a  
  simulation-based rather than an indistinguishability-based definition?  
  Might be slightly more general. We agree with the reviewer. However, in terms of the original time-lock puzzle definition, to make our C-TLP definition consistent with previous work, we followed previous work definition, e.g. [39,12] which rely on an indistinguishability-based one.  
  
- page 5, Setup and Generate Puzzle: there is a type mismatch here, namely,  
  k is being treated both as an integer mod N and as a key for a symmetric  
  encryption scheme. It might be better to be more careful about this:  
  
    k' <-$- ZZ\_N^\*  
    o2 <--- (k' \* r^a) mod N  
    k  <--- KDF(TO\_STRING(k'))  
  
  where TO\_STRING turns an integer mod N into a canonical byte representation  
  and KDF is a secure key derivation function, say, HKDF. We will highlight that point in the paper. i.e. we will explain that the key type must be converted to integer of $\mathbb{Z}\_N$ in step 2.b.  
  
- page 6, "Parallel composition problem": the footnote mark ("2") should go  
  after punctuation, not before, per Chicago Manual of Style 15th edition  
  section 16.30. done  
  
- page 7, "Adding efficient publicly verifiable algorithm": I'm not sure  
  if there is an application where you want public verifiability without  
  revealing the message hidden in the puzzle, but clearly a Chaum-Pedersen  
  ZK proof could be used for this purpose. Could be worth mentioning. It is an interesting idea. However, at this point we cannot think of any application in which such idea plays a role.   
  
- page 8, Definition 5: the letter 'j' is used for two separate purposes in this  
  definition. The third line to the right of the vertical bar inside the  
  brackets uses 'j' as a generic index while it's already being used for  
  something else in the definition. Would be clearer to use a different letter. Done-  
  
- page 9: sometimes the text says "GenPuZ" instead of "GenPuz" done  
  
- page 9, Generate Puzzle, step (d): o\_{j,1} and o\_{j,2} were not previously  
  defined, so this line is confusing. Done- it has been clarified in step 2.b  
  
- page 10, Prove: doesn't the Prove algorithm just \*output\* the proof,  
  according to the syntax? I don't think the syntax says anything about  
  transmitting to the verifier. Done--Now the syntax says it is given to the verifier.  
  
- page 10, Section 5: real-time detection and fair payment are still undef here.  
  They are not actually defined until the next page. Need to forward ref or  
  otherwise fix use-before-def, because reader gets frustrated Done. Now we have added a short description of each property to the introduction section.   
  
- page 10: "utilize" and "use" do not mean the same thing. It's clearer  
  to use the word "use" when that's what you mean. ("utilization" refers  
  to the \*degree\* to which something is used). done  
  
- page 11: the description in Section 5.1 is a high-level schematic, but  
  it actually misstates some details (e.g., "two puzzles" turns out not  
  to be true, right?). So maybe it would be useful to clarify this. As stated in 2nd paragraph in section 5.1, two puzzles are generated for ‘’each j-th verification’’. It is also consistent with the SO-PoR protocol, i.e. step 2.d. Thus, the high level description is correct and consistent with the protocol.

- page 12, Client-side Store, step (b): "PRF(.,.) mod p" is questionable  
  notation. If the resulting value needs to be uniformly random modulo  
  p, then the PRF's output depends on the bit length of p. This should  
  be clarified, because as written it isn't quite correct and may lead  
  to implementation errors later. Done- in section 5.3. step 1, we have clarified the size of p is equal to the size of PRF’s output.   
  
- page 12, Cloud-side proof generation: is H defined anywhere? Yes, it has been defined in section 3.2  
  
Justification of recommendation  
-------------------------------  
See first three paragraphs of "Comments for author."  
  
  
\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*  
  
  
Review #115D  
===========================================================================  
  
Overall merit  
-------------  
2. Weak reject  
  
Paper summary  
-------------  
The paper shows how one should chain time lock puzzles (TLPs) and does so generically.  It proposes applications of chained TLPs along the lines of recently fashionable blockchain protocols, like verifiable delay functions (VDFs) and the miss-named proof-of-retreavability (PoR).

In these, the paper ignores that VDFs and PoR amortize their cost by operating for many participants in parallel, while TLPs grow linearly in participants, making TLPs extremely expensive in practice. The focus and application of this work is on single-user setting.   
  
Comments for author  
-------------------  
If I understand, your SO-PoR protocol incurs CPU costs that dwarf the storage and networking costs.  I suppose PoSt from [5] is similarly costly.   
  
It appears PoR only proves the server possesses the data, not that the server would supply the data.  The ‘’poofs of retrievability’’ (PoR) schemes allows a client to check whether its ‘’entire’’ outsourced data/file is retrievable. On the other hand, the exist, a weaker (but more efficient scheme) called ‘’proof of data possession’’ (PDP) schemes, e.g. [4], that let a client verify if the sever possesses its intact data. The focus of our paper is the stronger scheme, i.e. PoR.

Isn't this kinda miss-named?  I suppose you inherited the name from [26], but maybe worth noting this. We preferred to adhere to the original title of the scheme (that has been used over a decade) to prevent any confusing.

I suppose PoR only handles lazy servers, not truly adversarial ones. Informally, as the original PoR definition (i.e. [46,26]) and SO-PoR definition indicate there are no assumptions on the behaviour of the adversary. It is allowed to deviate from the protocol arbitrarily. Therefore, in general, the adversary considered in PoR schemes (including our SO-PoR) is an ‘’active’’ one.  
  
VDF adversaries are always assumed 10 to 100 times faster than honest evaluators, possibly more if allowed super-conducting computing.  TLPs require similar.  We select VDF runtimes "only just bearable" for this reason, so chained TLP delays grow progressively further from the desired delay.  SO-PoR cannot provide real-time detection under this more realistic model. Note TLP parameters (i.e. maximum power of the adversary and delay parameters) are public, so are time points/window at which PoR proofs should be provided in SO-PoR. Therefore, in SO-PoR the client can always set TLP paramters such that a puzzle is solved at a certain time point/window, so the server can provide each PoR proof to the contract who verifies it within a certain time. Moreover, since the parameters are public, the server can check them and ensure they make sense, before it agrees to serve the client.

You never clarify the protocol being deployed for your cost comparison between VDFs and C-TLP.  If I understand, you assume only one user here, maybe even you mean simulating a TLP with a VDF, which is maybe this TDF construction.  Yet, VDFs are almost exclusively used to provide randomness that drives protocols with numerous/unlimited users, while RSA TLPs only supports one user per puzzle.  VDFs amortize away these costs, while TLP costs grow like O(participants).  Are you saying merely that VDF verifier rules are expensive overkill if used in a TLP?  If so, fair enough, this could be clearer and more concise. Yes, as we explicitly stated, in ‘’certain cases’’ we can substitute a VDF with C-TLP, to gain better efficiency. We have also provided a concrete use case ,and showed how to replace VDF in [5] with our C-TLP to improve costs.

You might skirt cost worries by arguing the client pays for the evaluation, but unclear.  In fact, it's rarely clear who pays for TLP or why they'd ever do so.  It's always hard to justify using a VDF, even after you amortize away the VDF evaluation costs, but actually using a TLP with its O(senders) cost scaling sounds inconceivable. In section 5.4. we will explicitly state ‘’the imposed cost has to be compensated by the client’’

Justification of recommendation  
-------------------------------  
The paper does specify exactly how to chain TLPs.  TLPs have a long history which this paper modernizes somewhat, although nothing unexpected emerges.   
  
I think TLPs have proven themselves fairly useless in practice, because they are expensive and their cost scales like O(senders).  Worse, the encrypted data has negligible value under most proposed TLP protocols.  In this paper, the authors ignore cost per participant everywhere, but compare with protocols that amortize cost well.  
  
An inexpensive solution would be threshold IBE with delayed issuing the IB secret keys, which yields zero marginal cost per sender and includes this chained staged release property, but under a threshold security assumption.  Any Byzantine agreement provides an implementation platform. The proposed solution, by the reviewer, requires a third-trusted party to issue IBE’s secret keys, which relies on much stronger security assumption. However, it has been shown that RSA modulus (and accordingly RSA-based TLP modulus) can be picked safely without relying on a trusted party. That means, in our proposed scheme no trusted third-party needs to be involved. Also, the applications of C-TLP we consider are in single-user setting. Moreover, isogeny VDF has a set of (serious) limitations, as stated in [\*]. In particular, in isogeny VDF (a) the time needed to setup public parameters is of the same order of magnitude as that is needed to evaluate the function, (b) validating public parameters takes the same time as evaluating the function, and (c) the evaluator is required to use O(T) storage for evaluating in optimal time. And (d) accordingly it is not suitable for the cases where a very long delays (e.g., months or years) are needed.

A recent paper shows isogeny VDF outputs provide similar delay issued IBE-like secret keys, but replaces the threshold assumption with a VDF assumption.  Isogeny VDF evaluators appear quite expensive, like $100k of RAM per hour of delay, but this memory hardness helps justify a smaller adversarial advantage than other VDFs or TLPs, and they also provide zero marginal cost per sender.   
  
In other words, threshold IBE and isogeny VDFs can do everything chained TLPs do, but also provide valuable collaborative randomness, and scale like O(1) while chained TLPs scale like O(senders).  In practice, you'd choose the threshold assumption over the VDF assumption's 10x adversarial advantage, especially for long delays, although a semi-centralized entity like a permissioned chain might prove their own honesty using the VDF I guess.   
  
Comment @A1 by Reviewer D  
---------------------------------------------------------------------------  
If I understand, your PoR protocol incurs CPU costs that dwarf the storage and networking costs.  
  
I think VDF adversaries are always assumed 10 to 100 times faster than honest evaluators, possibly more if allowed super-conducting computing, which makes chaining VDFs and TLPs problematic.  You might skirt this by arguing the user pays for the evaluation, but unclear.  In fact, it's rarely clear who pays for TLP or why they'd ever do so.  VDFs amortize this cost.  
  
You never clarify the protocol being deployed for your cost comparison between VDFs and C-TLP.  If I understand, you assume only one user here, maybe even you mean simulating a TLP with a VDF.  Yet, VDFs are almost exclusively used to provide randomness that drives protocols with numerous/unlimited users, while RSA TLPs only supports one user per puzzle.  Are you saying merely that VDF verifiers rules are expensive overkill if used in a TLP?  If so, fair enough, this could be clearer and more concise.

[\*] ‘’Verifiable Delay Functions from Supersingular Isogenies and Pairings’’, by Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso.