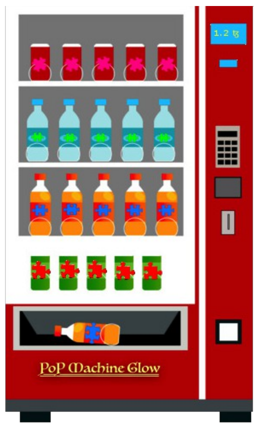


PoPMG: A Decentralized Oracle For Zero Knowledge Proof Verification On Tezos

March 24, 2020

Chain of Insight Team—D. Taylor, J. Dechant, J. May

https://beta.popmachineglow.io

Version 1.0.0

1 ABSTRACT

Introducing "Proof of Puzzle"...All your answers are ~~belong~~ unknown to us

PoP Machine Glow (PoPMG) is a zero knowledge encryption oracle and DApp built on the Tezos network. It provides users and content creators with verifiable proof of passing a test. More specifically, PoPMG verifies a user knows the correct answer to a question without revealing the question or its answers. Since PoPMG uses zero knowledge proofs it's safe to publicly commit them to the Tezos blockchain so everyone can verify them should they choose to run the proof calculations on their own hardware. In this paper we explain how PoPMG proofs are calculated and verified in a zero knowledge and what advantages they can provide both for the Tezos network and the blockchain ecosystem at large. As a companion to this paper we've provided a working implementation of the PoPMG oracle contracts and a DApp implementation and frontend for using them. In this paper we'll analyze our DApp's effectiveness in delivering the goals for zero knowledge cryptography of: *Completeness*, *Soundness*, & *Secrecy.* Finally, we'll end with a discussion of ways our oracle might be applied to solve real world problems regarding identity verification and secret sharing.

2 CONTENTS

3 Zero Knowledge Proof Verification

3.1 UNDERSTANDING ZERO KNOWLEDGE PROOFS IN PoPMG

To understand how PoPMG is using zero knowledge proofs we can start by thinking about how modern web applications verify users logging into their platforms. A typical scenario involves storing a hash of the user's password in a standard database. When a user logs into the platform their password and user credentials are sent in an HTTP POST request to the server; the web server retrieves the stored database password hash, and hashes the password from the HTTP POST request using the same algorithm as the stored hash. If the two hashes are identical the server can proceed with authentication. Since the server itself doesn't know the actual user login credentials, we can say it operates as one kind of zero knowledge prover. Even a web administrator with access to view the entire database won't be able to login as another user, as knowing the password hash and email address of a particular user won't allow you to login as them. To perform a successful login you need to know the unencrypted password text that will hash to the stored database value.

With a good grasp on the above example, understanding how the PoPMG prover works is well within reach. We begin by taking a secret *S* and encrypting it multiple times to produce a resultant hash *H*. Aside from passing *S* through multiple rounds of encryption, we haven't yet diverged much from our basic web server example but that's about to change: the amount of encryption rounds to apply to *S* is determined by the total amount of rewards plus one $R + 1$. Given a secret puzzle with $R = 3$, we apply $R + 1$, or 4, rounds of encryption to *S* to produce *H* which is our public hash. All verifiers *V* need to know *H* when they are verifying the solution of any prover *P*. In order for *P* to claim Tezos rewards offered by the puzzle they need to produce a knowledge commitment which they will send to the PoPMG oracle smart contract to be verified in zero knowledge. The content of this knowledge commitment contains two pieces of information: the hash S^i of a previous encryption round, and the depth *i* at which it occurred. To verify this information the oracle hashes S^i an amount of rounds equal to $(R + 1) – i$ and checks whether the resultant hash matches the public hash *H* that was stored by the oracle contract at the time of puzzle's creation. If the two hashes are identical, then *P*'s knowledge commitment has been verified by the oracle and the contract can safely call the rewards proxy contract which is capable of distributing both XTZ and NFT rewards to *P* at an average gas cost of 0.031 ꜩ. Once a particular depth *i* is claimed by *P*, it is retired by the oracle contract and can't be used again. Proofs are retired this way because, similar to *H* which is made public by the puzzle creation transaction, sending a claim reward transaction to the Tezos network will make that particular S^i proof public to anyone on the network who sees the Tezos operation.

Practical Demonstration:

- Puzzle creation: $R = 3$

1. blake2b("test secret message") : = (H^1)

0xa056d12c78c34f05d1a0aa0467ae8dcbafd429d1e94ea14981dac07e5d2f2ac2

2. blake2b(H^1) : = (H^2)
   0x44554082187b746f18d57f401a3d202e6cfa0852376b39b50b0cf085a043bd56
3. blake2b(H^2) : = (H^3)
   0x94ed0fcc6d0c66e5195bf9cab99e511c9a3415ed2484485a1d26511bbe22b4c7
4. blake2b(H^3) : = (H^4)
   0x3b8fc413a27ec4ed9cb37536fc0f3e4b1424510c033d6ec8dd11f8e3807b6563
5. $R + 1 = 4 \therefore H = H^4$

- Prover:
  - *P = 0x44554082187b746f18d57f401a3d202e6cfa0852376b39b50b0cf085a043bd56*
  - $i = 2$

1. *Encryption Depth = (R + 1) – i*
2. blake2b(P) : = (H^3)
   94ed0fcc6d0c66e5195bf9cab99e511c9a3415ed2484485a1d26511bbe22b4c7
3. blake2b(H^3) : = (H^4)
   3b8fc413a27ec4ed9cb37536fc0f3e4b1424510c033d6ec8dd11f8e3807b6563
4. Since $R + 1 = 4$, and $i = 2$, and P hashed $(R + 1) – i$ rounds is equal to H, then P's knowledge commitment is verified

Completeness:

If *P* is valid than *P* hashed *(R + 1) – i* rounds is equal to *H*, the public hash, then *P*'s knowledge commitment is verified.

Soundness:

If *P* is not valid then *P* hashed *(R + 1) – i* rounds is not equal to *H*.

Note: Since cryptography is essentially an heuristic approach, any weakness to completeness or soundness is the same as finding a vulnerability in the cryptographic algorithm itself. The added distinction of depth would require an attacker to reproduce either the original source input payload of *S*—which is equivalent to knowing the plain text answer anyway—or else the encrypted output of a particular depth which happens to be a payload of 66 characters[1] and can only be claimed from the oracle contract once.

Secrecy:

Since anyone willing can check the veracity of the knowledge commitment *P* without being in possession of *S*, and because the value of *S* is never stored in the oracle, and having tested both *Soundness* and *Completeness* we can safely conclude our model is privacy preserving and

1 E.g. the output of blake2b-256

verifiable in zero knowledge.

3.2 LOOKING AT EXISTING SOLUTIONS

zk-SNARKs:

The acronym zk-SNARK stands for “Zero-Knowledge Succinct Non-Interactive Argument of Knowledge,” and they are a robust system for producing and verifying zero knowledge proofs. zk-SNARKs already see usage within Z-Cash and Ethereum. More recently, there has been discussion to bring zk-SNARKs to Tezos in a protocol upgrade titled Sapling.[2] zk-SNARKs come at a disadvantage of having a high cost to produce but come with a low cost to verify. A high level description of zk-SNARKs by Ethereum core developer Christian Reitwießner explains:

> As a very short summary, zk-SNARKs as currently implemented, have 4 main ingredients [...]:
>
> A) Encoding as a polynomial problem
>
> The program that is to be checked is compiled into a quadratic equation of polynomials:
>
> $t(x)h(x) = w(x)v(x)$, where the equality holds if and only if the program is computed correctly. The prover wants to convince the verifier that this equality holds.
>
> B) Succinctness by random sampling
>
> The verifier chooses a secret evaluation point s to reduce the problem from multiplying polynomials and verifying polynomial function equality to simple multiplication and equality check on numbers: $t(s)h(s) = w(s)v(s)$
>
> This reduces both the proof size and the verification time tremendously.
>
> C) Homomorphic encoding / encryption
>
> An encoding/encryption function E is used that has some homomorphic properties (but is not fully homomorphic, something that is not yet practical). This allows the prover to compute
>
> $E(t(s)), E(h(s)), E(w(s)), E(v(s))$ without knowing s, she only knows $E(s)$ and some other helpful encrypted values.
>
> D) Zero Knowledge
>
> The prover obfuscates the values $E(t(s)), E(h(s)), E(w(s)), E(v(s))$ by multiplying with a number so that the verifier can still check their correct structure without knowing the actual encoded values.[3]

zk-SNARKs work well for general knowledge proofs that can be employed when you have the time and compute power to do so. They are harder to justify for smaller inputs or high volume low-latency networks, they are also not currently suitable for mobile devices because of their reduced compute power. PoPMG however is lightweight enough to work on mobile devices but, as we'll see in section 4, it's limited in other ways. PoPMG is complimentary to zk-SNARKs since zk-SNARKs excels in places where PoPMG is completely impractical such as operations on big

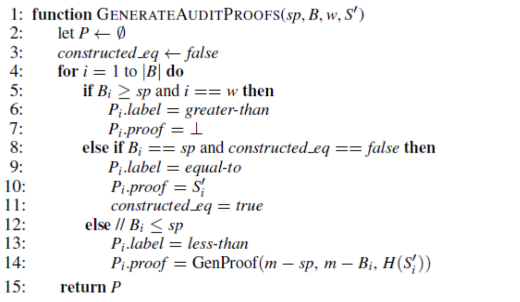
2 Sapling proposal: https://gitlab.com/tezos/tezos/blob/1cd31972ed2de9deee77592b8ffc5fb3d0170d1a/vendors/ocaml-sapling/README.md

3 Reitwießner, 2016

data or file inputs.[4]

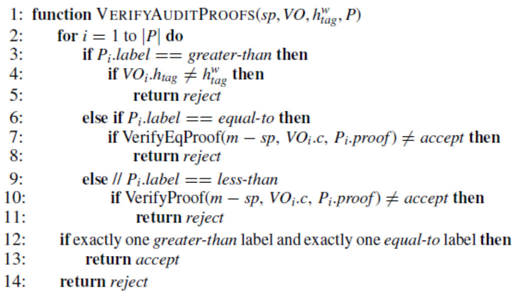
4 PRIVATE INTEGER COMPARISONS & THE ANGEL WALFISH MODEL

The system we have described in 3.1 uses hash chains and depth indexes for producing proofs. Since the inputs published to the Tezos network by provers are previous outputs from blake2b, recognizing how the secret message has been obfuscated is easy enough—all public inputs are not discernible text but instead are cryptographic hashes. Understanding how encryption depth is used for the production of proofs within a hash chain is slightly more involved: it is computationally infeasible for any user to find an input such that it is a valid reward proof by reverse engineering a chain of hashes generated from a secure one-way hash function.[5] This unidirectional nature of the hash functions allows PoPMG proofs to use greater than or less than comparison operations on integers. The format for verifying these claims and model of using hash chains to produce them, was originally published by Sebastian Angel and Michael Walfish in a research paper titled *Verifiable Auctions for Online Ad Exchanges* in 2013.[6]



```
function GenerateAuditProofs($(sp, B, w, S')$)
    let $P \leftarrow \emptyset$
    $constructed\_eq \leftarrow false$
    for $i = 1$ to $|B|$ do
        if $B_i \geq sp$ and $i == w$ then
            $P_i.label = greater\text{-}than$
            $P_i.proof = \perp$
        else if $B_i == sp$ and $constructed\_eq == false$ then
            $P_i.label = equal\text{-}to$
            $P_i.proof = S'_i$
            $constructed\_eq = true$
        else // $B_i \leq sp$
            $P_i.label = less\text{-}than$
            $P_i.proof = \text{GenProof}(m - sp,\ m - B_i,\ H(S'_i))$
    return $P$
```

Figure 6—Pseudocode for proof generation. *sp* is the auction's sale price, *B* is the set of bids, *w* is the index of the winning bidder's bid in *B*, S' is the set of secret seeds, and *m* is the maximum allowed bid. This procedure uses the integer comparison protocol described in Section 3 and an extension (Section 4.5).



```
function VerifyAuditProofs($(sp, VO, h^w_{tag}, P)$)
    for $i = 1$ to $|P|$ do
        if $P_i.label == greater\text{-}than$ then
            if $VO_i.h_{tag} \neq h^w_{tag}$ then
                return $reject$
        else if $P_i.label == equal\text{-}to$ then
            if $\text{VerifyEqProof}(m - sp,\ VO_i.c,\ P_i.proof) \neq accept$ then
                return $reject$
        else // $P_i.label == less\text{-}than$
            if $\text{VerifyProof}(m - sp,\ VO_i.c,\ P_i.proof) \neq accept$ then
                return $reject$
    if exactly one $greater\text{-}than$ label and exactly one $equal\text{-}to$ label then
        return $accept$
    return $reject$
```

Figure 7—Pseudocode for proof verification. *sp* is the auction's sale price, *VO* is the set of VEX objects, h^w_{tag} is the hash of the winning bidder's ad tag, and *P* is the set of labeled proofs provided by the auctioneer. This procedure relies on the protocol described in Section 3 and an extension (Section 4.5).

This protocol works well for Angel and Walfish's VEX ad exchange because the zero knowledge integer comparisons—greater or less than proofs—prove that some bidder has produced a higher value bid than the next highest bidder. When a bidder adds a valid bid, the hash chain of the auction is incremented by a subsequent hash round which the bidder keeps as proof of their bid index. If a bidder tries to produce an invalid bid, which is not higher than the current winning bid, it gets rejected and is not added to the hash chain of valid bids. Since each subsequent bid represents a link in the final hash chain, the VEX oracle can calculate and verify the proof hashes of any bidder who participated in the auction, and can also identify which

4 E.g. DIZK, a system that distributes the generation of a zk-SNARK proofs across multiple machines in a compute cluster, which was tested on 2048 x 2048 pixel image file inputs. (Wu, Zheng et. al. 2016)

5 E.g. A function H that maps an arbitrary length message M to a fixed length message digest MD is a one-way hash function if, 1. It is a one-way function. 2. Given M and H(M), it is hard to find a message M^'!=M such that H(M^')=H(M). (*One-Way Hash Function*. Wolfram Alpha Mathworld, accessed 3/23/2020)

6 Angel, Walfish, 2013.

hash refers to the VEX account that has won the auction item.[7]

It is important to recognize the limitations of Angel-Walfish's model for private integer comparisons since there are definite advantages in zk-SNARKs because general knowledge proofs are able to prove arbitrary data models whereas the PoPMG prover is limited to using integer comparisons. However, as we'll see in 5.2, these limitations can be overcome if an element of trust is brought into the equation in the form of a trusted third party oracle. For now it's suffice to say integer comparison proofs excel at proving linear data such as time sensitive conditions like the first student to complete a test correctly, or auction style scenarios where a specific value is always incrementing or decrementing only in one direction.

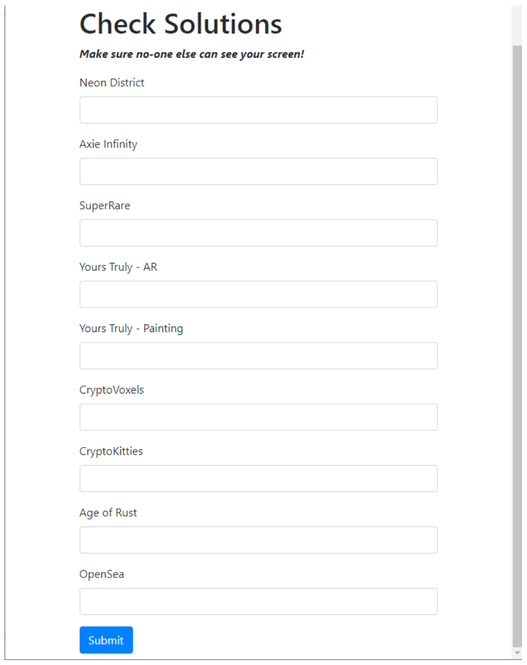
5 APPLICATIONS FOR THE PoPMG ORACLE

The following are a few practical ideas for applications which could be built using the PoPMG oracle, either directly or by making some insignificant changes to the core modeling:

5.1 THE PoPMG DAPP AND DIRECT APPLICATIONS

PoPMG was created to solve an immediate need within the crypto-puzzles community for a transparent way to replace current puzzle checker systems that post passwords over regular TLS HTTP. The below image[8] is an example of one such "black box" puzzle checker system for a recent crypto-puzzle game, called "Yours Truly", produced by artist Josie Belini[9] in collaboration with Blockade Games and many other blockchain gaming and NFT companies. The puzzle included a prize pool of 10 ETH, 5000 Enjin coin and many NFT prizes including artwork from SuperRare,[10] and collecitbles like cryptokitties[11] and axies.[12] A full walk-through and explanation of the puzzle design can be found here: https://buer.haus/2020/03/03/josiebellinis-yours-truly-puzzle-walkthrough/ (accessed March 24, 2020).

7 E.g. Final hash -1 round
8 This puzzle checker was originally served at: https://resultchecker.neondistrict.io/ but it's since been taken down. A cached copy can still be viewed at: https://web.archive.org/web/20200323195152/https://webcache.googleusercontent.com/search?q=cache%3AnD6NdzouUCIJ%3Ahttps%3A%2F%2Fresultchecker.neondistrict.io%2F+&cd=1&hl=en&ct=clnk&gl=us
9 See: https://josie.io/
10 See: https://superrare.co/
11 See: https://www.cryptokitties.co/
12 See: https://axieinfinity.com/


Check Solutions

Make sure no-one else can see your screen!

Neon District

Axie Infinity

SuperRare

Yours Truly - AR

Yours Truly - Painting

CryptoVoxels

CryptoKitties

Age of Rust

OpenSea

While solutions like the above puzzle checker serve their purpose well enough, there are a number of areas where they can be improved:

- **Technical bias**: Finding a correct set of solutions with the checker is not correlated in any way with claiming the prize. It is possible to envision a scenario where a player solves some riddle first but is unable to claim its reward because while they struggled with importing the private key containing the prize, they lost precious minutes until a more technical user was able to solve the riddle and sweep the wallet almost immediately.
- **Prize management**: Since puzzles can contain any number of private keys the puzzle author must securely manage and embed each of these within their puzzles. The PoPMG oracle on the other hand, manages the storage and distribution of all XTZ and NFT rewards on the author's behalf. Using PoPMG, an author doesn't need to create or manage any wallets but instead simply transfers rewards to the oracle contract during puzzle creation.
- **Rewards for n-th solver**: A third consideration is since the current method of rewards distribution in crypto puzzles involves sweeping a private key, there's no way to provide rewards to anyone other than the first claimant. PoPMG however provides puzzle authors with the ability to offer rewards at any claim index.
- **Transparency and censorship resistance**: Web servers can fail. Even at the time of writing the *Yours Truly* puzzle checker has been retired from neondistrict.io as it's no longer an active puzzle. Consider a situation where a user knows they are close to solving a puzzle so they decide to perform a denial of service attack on the puzzle checker to bring it offline. While the server is offline they finish running their final calculations. This would effectively block other solvers from using they checker in the meantime and potentially this misstep will cause them to lose out on recovering the

> reward first. At some point the DDoS attack is lifted by the attacker—once they're final calculations have checked out—and now the puzzle checker is accessible again, unfortunately it's too late because the DDoS attacker was able to stall everyone long enough until their dictionary attack on the puzzle provided them with the missing password. Processing zero knowledge claims on the Tezos blockchain removes the possibility of these sorts of attacks. Even if the DApp version of the checker is brought offline, players could connect to the smart contracts directly. Using zero knowledge hash chain proofs that are displayed in public, and can be tested and verified by anyone, also presents the crypto-puzzles community with a totally transparent system for puzzle rewards and answer submission. This is the opposite of black box systems employed by many puzzle checkers such as the recent *Yours Truly* 10 ETH puzzle.

Keeping the above concerns in mind, we've built the PoPMG DApp which is a frontend implementation for creating puzzles, checking them, and claiming their rewards. Since our oracle is built on Tezos smart contracts it is easy for any puzzle author to create their own puzzle with the PoPMG DApp then provide solvers with a custom frontend that connects to our oracle in the same fashion as the PoPMG DaPP—this allows for a seemless integration of the puzzle solving and password checking experience directly embedded in their game that's running on just about any platform. While the PoPMG DApp is already sufficient for allowing users to solve and claim prizes securely, a custom frontend that displays the author's target puzzle without listing every puzzle in the oracle can provide a more customized user experience. The PoPMG DApp beta frontend is currently available at: https://beta.popmachineglow.io

Another interesting direct application for PoPMG would be to use the oracle contract for grading timed tests; these could be educational tests for home learning, or in class exams so long as there is some means for digital submission. Timed tests allow private integer comparison using timestamps as they're moving forward in time. In 2018 members of our team built a online system for educational testing called *BUIDLing Blocks*.[13] While it was a fun project it lacked the ability for a student to prove they had passed an exam and did not track how long it had taken them to complete it. *BUIDLing Blocks* was primarily intended for delivering educational modules and graded tests to home schooled children with a target age group of 4 to 16 years old. In hindsight it's easy to see that with the addition of graded tests powered by the PoPMG oracle, *BUIDLing Blocks* could become a very interesting project.

5.2 THIRD PARTY ORACLE SYSTEMS

While at first it may seem private integer comparisons using hash chains have a limited scope, it's possible to widen that scope to produce a limited kind of general knowledge proof. These general proofs are “limited” because they come at the risk of including a trust model as our public proofs will need to be validated by a third party oracle. Although it would be ideal to rely only on zero knowledge proofs, there are use cases—especially for identity verification—where a third party oracle can create an acceptable standard of security. The most compelling example would be if the issuer of a particular identity document operated as the trusted third party. In the case of a government issued passport, the scenario might unfold as follows: a government

13 See: https://github.com/drewstaylor/buidlingblocks

G is given ownership of a Tezos contract where they store hashes of digital copies of citizens' passports at a given depth *D*. Since the information is stored on the Tezos network, any airline can query the ID record of any citizen boarding a flight. Instead of traveling with their physical identity document, the passenger produces a knowledge commitment at a specific depth—less than *D*—as requested by the airline. Knowing the requested depth and having received back a knowledge commitment, the airline can test the passenger's proof by hashing it ($D — i$) times and verifying the resultant hash is identical to *D* which is publicly stored in *G*'s oracle contract. While it might not be safe yet to board such a passenger, the airline has already ensured they are in possession of an identical copy of the digital passport stored on *G*'s oracle. Importantly, they verified this information without requiring the passenger to divulge personal information. While tedious, the remaining verification problems to board our passenger safely (such as proof of travel to other countries within the last 12 months) can of course be handled in the same manner. By providing separate knowledge commitments for each leg of travel and verifying them with those governments' oracle contracts, the passenger proves the entry and exit timestamps of their recent travel history. Even photo verification can be handled using this same method, but with one caveat: the passenger needs to show the airline a digital photograph, likely using their phone or tablet, and prove its checksum exists in the database of their government's oracle. This way the airline knows it is an identical image to the government's copy and the file hasn't been tampered with. This is how it would be possible to board a passenger securely on an international flight without requiring private information apart from a digital copy of a passenger's passport photo. In the same fashion, this system could be used for purchasing restricted goods such as tobacco or lottery tickets at a vendor point of sale without requiring the purchaser to disclose their identity documents to prove their age.

5.3 A “BURN AFTER READING” MESSAGE PROTOCOL

Another novel idea for a project that could be built on PoPMG is a “burn after reading” message protocol. As its title suggests, it would be a system where messages auto-destruct from the blockchain. This is possible in Tezos because Michelson map and big map types allow for the removal of items. The proposed system would work using the PoPMG rewards claim function for secrets encrypted at a depth of 3 where S^2 represents the proof hash and S^3 is equal to *H* the public hash used for proof verification. Once a prover *P* has claimed the proof at S^2, the oracle uses it as authentication to confirm *P* has access to decrypt the secret message encoded by the author. Using the grant rewards function, *P* is rewarded an NFT token, the data property of which is the encrypted payload of the author's message and can be decrypted by *P* using S^1 as the decryption key. Finally, once *P* has successfully received and decrypted the author's intended message, the messaging DApp sends a notice of receipt to the NFT contract to burn their token which removes all its data—including the encrypted secret message—from storage. The original message, in encrypted format mind you, might still be located by someone who knows the transaction hash of the NFT mint operation or the puzzle create operation, but going forward it won't be recoverable by anyone inspecting the storage of the oracle, rewards proxy or NFT contracts.

In order to mitigate against a bad actor watching for transactions to the oracle contract, we can use blinding to further obfuscate the author's secret message. Instead of storing a single message payload, the oracle can store a series dummy payloads which it will use to blind the secret message. We achieve this by taking the real encrypted message and placing it at a random index inside a set type variable where it sits along side multiple encrypted fake messages.[14] The fake messages need to be randomly seeded, and not even the intended recipient *P* should be able to decrypt them. To recover the contents of the original message *P*'s DApp needs to loop through the entire set of messages using S^1 to discover which one is the real intended message. Since only *P* and the original author can calculate the value of S^1, anyone inspecting the Tezos operation that minted the message NFT, or the operation that created the conversation, won't have any reasonable means of determining which is the blinded real message among the hoard of fakes. What was once an already monumental bruteforce task to discover a 66 character bytes input is now exponentially more difficult proportionate to the number of items in the set; however, the cost for *P* to loop through a set of say 100 entries to find the correct decryption is not a terrible computation cost and can still be derived in a matter of seconds on a modern computer. Another way our encrypted secret message can be obfuscated on the blockchain is if each conversation instantiates a new set of contracts, this is possible to do at the level of the messaging DApp itself when a new conversation is opened since the Taquito JavaScript SDK allows for contract origination.[15] However, it should be noted to create contracts makes the conversation more costly since the message author will pay a larger transaction fee to pay to deploy the contract's storage cost and burn cap.

An interesting outcome of the above described message protocol is the elimination of any need to perform Diffie-Hellman key exchange. That means messages can be exchanged between two parties secured by a human readable password without requiring a private key stored on a specific device. Since there is no need for importing keys or wallet mnemonics when transitioning between devices, this system is also sufficient for sending private data to yourself to be picked up from another device. The following describes the basic gist of the Diffie-Hellman system which is used by popular messaging apps such as WhatsApp and Signal:

> Diffie–Hellman key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. The initial implementation of the protocol used the multiplicative group of integers modulo p, where *p* is prime and *g* is primitive root $\mathrm{mod} p$. Both parties arrive at the same value g^{ab} and g^{ba} both equal to $\mathrm{mod} p$, and use this value to encrypt and decrypt data. Only a, b and $\mathrm{mod} p$ are kept secret (private key). All the other values viz., p, g, g^a mod p, and g^b $mod\ p$ are sent in the clear (public key) […] The Diffie–Hellman protocol can be strengthened by using elliptic curve cryptography, which makes the discrete logarithm problem almost impossible to solve.[16]

It should be noted our proposed system is not meant to replace existing secure messaging apps. Taking the example of a direct message conversation, it is infeasible to expect users to pay gas costs for sending every message, or to expect them to wait for block confirmations in a real time setting. However there are some interesting advantages and use cases for the single-use message system we've here described. In the case of a government censoring its citizens' access

14 E.g. "set (t): Immutable sets of values of type (t) that we write as lists { item ; ... }, of course" (see: https://tezos.gitlab.io/whitedoc/michelson.html#core-data-types-and-notations)
15 See: https://tezostaquito.io/docs/originate/
16 Ganjewar, 2010.

to the Internet—as occurred in Egypt during the Arab Spring of 2011[17]—it would be useful to have a decentralized protocol for messaging. Not only can secure messaging services be blocked by governments at the DNS level (and historically they have been doing so[18]), but in the case that a journalist's laptop or phone are seized and by law authorities, they could become unable to access their message. Perhaps even worse is they could be compelled under duress or threat to unlock their devices and show the authorities the contents of their decrypted conversations. However, using our proposed “burn after reading” message service this problem ceases to exist as our journalist's secret messages continue to be accessible even if they're separated from their devices (provided they can find some way to access the Internet). Furthermore, these messages are resistant to government censorship because of decentralization. Access to the messaging service can no longer be turned off by a government blocking a specific DNS. Lastly, our message DApp can allow for multiple parties to access group information without compromising security which is something Diffie-Hellman based systems historically have struggled with.[19] In the case of a group, all of the provers do not delete the message immediately from NFT storage but allow it to propagate first to all of the members of their group, and the mechanism which determines the final recipient—and which will trigger the message to be burned—is the same method that's already used in PoPMG to determine the final recipient of a puzzle reward.

6 CONCLUSION

Privacy has long been a central aim in blockchain technology, but what might have been considered private in the past is often no longer seen to be the case. For example, the claim that Bitcoin transactions are anonymous is now often refuted as blockchains are transparent and open data sources. While it might be difficult to link some Bitcoin address with a specific person in the network, all of the network's traffic can be tracked and mapped and it's this sort of analysis that has come under scrutiny in recent government policy attempts to crack down on various cryptocurrency related crimes and legal cases.[20] zk-SNARKs is a recent cryptographic tool for privacy preservation that has garnered much academic and industrial interest. While it can produce generalized knowledge commitments for zero knowledge proofs it comes with a high compute cost for producing them. However, we've shown it's possible to use the Angel-Walfish integer comparison model for a variety of zero knowledge proofs at a significantly lower compute cost, but with a limited scope. That said, when taken in combination with a trusted third party oracle, even claims outside of Angel-Walfish's model for private integer comparisons can be verified. Having a model for lightweight zero knowledge proofs that can be verified and claimed on the Tezos blockchain is a notable achievement and one that we're very excited

17 E.g.: https://www.aljazeera.com/indepth/features/2016/01/arab-spring-anniversary-egypt-cut-internet-160125042903747.html (accessed: March 24, 2020)

18 E.g. https://www.nytimes.com/2017/09/25/business/china-whatsapp-blocked.html (accessed: March 24, 2020)

19 Note: since group chats defy the possibility to authenticate users with normative Diffie-Hellman key exchange it group chats are widely known to be less secure. A simple news search unleashes a trove of publications that describing vulnerabilities for group chats on WhatsApp and Signal. E.g. https://www.wired.com/story/whatsapp-group-chat-crash-bug/ (accessed: March 24, 2020)

20 E.g. Supreme Court of Nova Scotia, Quadriga CX application for credit relief, 2019.

about. We have also provided a proof of concept DApp and frontend to help understand the capabilities of these zero knowledge proofs using hash chains, and to demonstrate their feasibility and usefulness to prove and verify claims on the Tezos blockchain.

7 REFERENCES

1. Angel, Walfish. *Verifiable Auctions for Online Ad Exchanges.* University of Texas at Austin. 2013.

2. Reitwießner. *zk-SNARKs in a Nutshell*. Ethereum.org. 2016.

3. Wu, Zheng, Chiesa, Ada Popa, Stoica. *DIZK: A Distributed Zero Knowledge Proof System*. UC Berkeley, 27th USENIX Security Symposium. 2018.

4. One-Way Hash Function. Wolfram Alpha Mathworld, accessed 3/23/2020. https://mathworld.wolfram.com/One-WayHashFunction.html

5. Fifth Report of the Monitor, *Application by Quadriga Fintech Solutions Corp., Whiteside Capital Corporation and 0984750 B.C. Ltd. dba Quadriga CX and Quadriga Coin Exchange, for relief under the Companies' Creditors Arrangement Act.* Supreme Court of Nova Scotia. June 19, 2019.

6. Ganjewar, *Diffie Hellman Key Exchange*. CS 290G: Secure Computation, UC Santa Barbara, 2010.