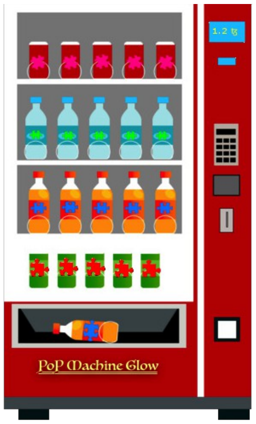


PoPMG: A Decentralized Oracle For Zero Knowledge Proof Verification On Tezos

March 22, 2020

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

https://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 puzzle authors 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 verifies this using 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 execution environment and what advantages they can provide both for Tezos network and the blockchain ecosystem at large. As a companion to this paper we've provided a working implementation of the PoPMG zero knowledge oracle contracts and a DApp implementation and frontend for using them. 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 sensitive data transfer.

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 original (stored) password hash. If the two hashes match the server can proceed with authentication. Since the server itself doesn't know the actual user login credentials, we can say it operates as a kind of centralized 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 $R + 1$. Given a puzzle Z 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 *Z* 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 *Z*'s creation. If the hashes match then *P*'s knowledge commitment has been verified by the oracle in zero knowledge and the oracle contract can safely call the rewards proxy contract which is capable of distributing both XTZ and NFT rewards to the sender of *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 rewards transaction to the Tezos network will make that particular S^i public to anyone on the network who inspects 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 of 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 once.

Secrecy:

Since anyone willing can check the knowledge commitment without being in possession of *S*, and because the value of *S* is never itself stored in the oracle, and having tested both *Soundness*

[1] E.g. the output of blake2b-256

and *Completeness* we can safely conclude our model is privacy perserving and verifiable in a 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 overall they make a very robust system for verifying zero knowledge proofs. zk-SNARKs are quite popular and see usage within Z-Cash and Ethereum; more recently there has been discussion to bring zk-SNARKs to the Tezos network in a proposed protocol upgrade titled Sapling.[2] zk-SNARKs come at a disadvantage of having a high compute cost to produce but have a low compute cost to verify. The basic method is described by Christian Reitwießner in his paper titled *zk-SNARKs in a Nutshell*

> 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]

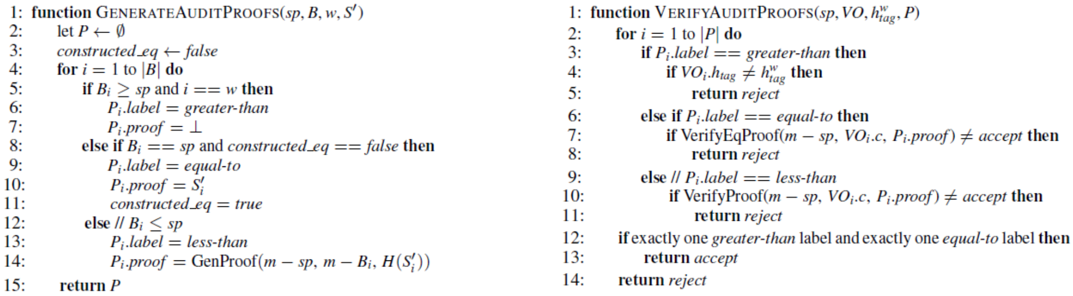
[2] Sapling proposal: https://gitlab.com/tezos/tezos/blob/1cd31972ed2de9deee77592b8ffc5fb3d0170d1a/vendors/ocaml-sapling/README.md

[3] Reitwießner, 2016

zk-SNARKs are wonderfully complex. They work well for general knowledge proofs and can be employed when you have the time and compute power to do so. They are harder to justify for smaller inputs or for high volume low-latency networks as it is not uncommon for proof generations to take tens of minutes even when given optimal resources. They are also not currently suitable for mobile devices because of their reduced compute power. PoPMG by comparison does not suffer from these hindrances and is lightweight enough to work on mobile devices. PoPMG is complimentary to zk-SNARKs since zk-SNARKs excels in places where PoPMG is completely impractical such as operations on big data or file inputs.[4]

4 PRIVATE INTEGER COMPARISONS & THE ANGEL WALFISH MODEL

The system we have described in 3.1 uses hash chains 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 one-way hash functions allows PoPMG proofs to use greater than or less than comparison operations. The format for verifying these claims and the 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).

[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.

This protocol works well for Angel and Walfish's VEX ad exchange because the zero knowledge integer comparisons—e.g. greater or less than proofs—prove that some bidder has produced a higher value bid than the next highest bidder.[7] When a bidder adds a valid bid, the hash chain of the auction is incremented by a subsequent hash round and bidder keeps as their proof. If a bidder tries to produce an invalid bid, which is not higher than the current winning bid, their bid gets rejected and it is not added to the hash chain. 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 hash refers to the VEX account which has won the auction item.[8]

It's important to recognize the limitations of Angel-Walfish's model for private integer comparisons since there are definite advantages in zk-SNARKs where the general knowledge proofs of can prove arbitrary data models where PoPMG can't be applied. As we'll see in 5.2 these limitations *can* be overcome if an element of trust is brought into the equation with a third party oracle, 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 on the PoPMG oracle, either directly or by making some insignificant changes to the core modeling:

5.1 THE PoPMG DAPP AND DIRECT APPLICATIONS

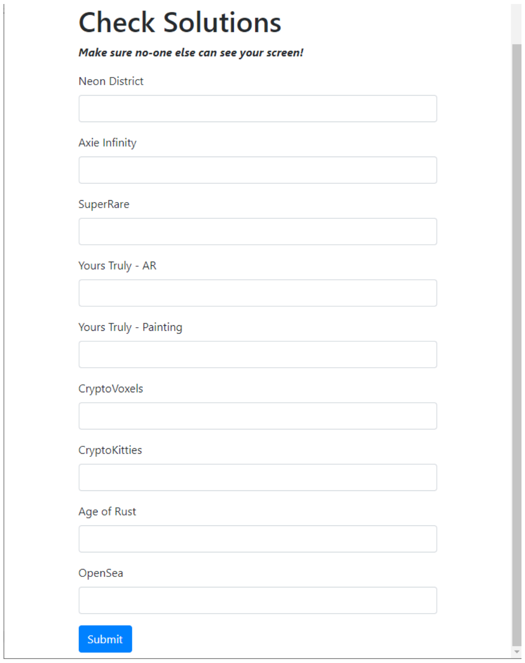
PoPMG was created to solve an immediate need within the crypto puzzles community for a transparent way to replace current puzzle checker systems that post passwords over regular TLS HTTP. The below image[9] is an example of one such "black box" puzzle checker system for a recent crypto puzzle game, called "Yours Truly", produced by crypto artist Josie Belini[10] in collaboration with Blockade Games and many other blockchain gaming and NFT companies.

[7] Note: while VEX uses only less than proofs, PoPMG is processed with greater than proofs but could just have easily used less than proofs.

[8] E.g. Final hash -1 round

[9] 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

[10] See: https://josie.io/



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

The *Yours Truly* puzzle included a prize pool of 10 ETH, 5000 Enjin coin and many NFT prizes including artwork from SuperRare,[11] cryptokitties[12] and axies.[13]

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

- **Technical bias**: The first is that 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 user 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 and a more technical user was able to solve the riddle and sweep the wallet almost immediately.
- **Prize management**: Secondly, since puzzles like *Yours Truly* can contain any number of private keys the puzzle author must securely manage and embed within their puzzles. Contrast this with the PoPMG oracle which 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 their rewards to the oracle contract during puzzle creation.
- **Rewards for n-th solver**: A third consideration is that since the current method of rewards distribution in crypto puzzles involves sweeping a private key, there was previously no accepted way to provide rewards to anyone other than the first claimant. PoPMG on the other hand provides puzzle authors with the ability to offer rewards any claim index.

[11] See: https://superrare.co/
[12] See: https://www.cryptokitties.co/
[13] See: https://axieinfinity.com/

- **Transparency and censorship resistance**: Finally, web servers can fail and even at the time of writing the *Yours Truly* puzzle checker has already been removed from neondistrict.io. Consider a situation where a user knows they are close to solving a puzzle and decide to perform a denial of service attack on the puzzle checker to bring it offline while the run their final calculation scripts. This would effectively block other solver from checking their results until DDoS attack is lifted by the attacker which of course they would have to do once they're finally sure their calculations have checked out and need to use the server again to check their own result. Processing zero knowledge claims on the Tezos blockchain removes this possibility. Even if the DApp version of the checker is brought offline by an attacker, it would still be possible for users to connect to the contracts directly. Using a blockchain implementation and zero knowledge hash chain proofs presents an overall transparent system which is the opposite of the close black box system comprised by most puzzle checkers available today that post sensitive data to a centralized private web server over HTTP.

Keeping the above concerns in mind, we've built the PoPMG DApp which is a frontend environment for creating puzzle, checking them, and claiming their rewards. While the PoPMG DApp is sufficient for solving puzzles and claiming their prizes securely, since our oracle is a built on Tezos smart contracts it is easy for any puzzle author to create their puzzle with our DApp but provide their solvers with a custom frontend that connects to our oracle in the same fashion as the PoPMG DaPP.

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 room exams for course work as long as there is some means for digital submission. Timed tests allow for private integer comparison using integer timestamps which are always moving forward in time. In 2018 members of our team built a online system for educational testing called *Buidling Blocks*.[14] While it was a fun project it lacked the ability to commit proof a student had passed the exam or how long it took them to complete, and was primarily intended for delivering educational modules with embedded tests for home schooled children. With the addition of PoPMG this could be easily corrected and a grading system can be created for *BUIDLing Blocks* that tracks both the students' grades and time to complete the exam.

5.2 THIRD PARTY ORACLE SYSTEMS

While at first it may seem like private integer comparisons using hash chains have a limited scope, it's possible to widen it to produce a limited kind of general knowledge based proof. These general proofs are “limited” because they come at the risk of including a trust model as our public proofs will be validated by a third party oracle.

Although it would be ideal to rely only upon 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

[14] See: https://github.com/drewstaylor/buidlingblocks

document itself operated as the trusted third party. In the case of a government issued passport, the scenario would operate as follows: a government *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 depth and 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, and they did so without requiring their passenger to divulge personal information. While cumbersome, 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 deftly proves the entry and exit timestamps of their recent travel history. Even photo verification can be handled in the same manner, but with one caveat: the passenger needs to show the airline a digital photograph, probably using their phone or tablet, and prove that its checksum exists in the database of their government's oracle, and that the file hasn't been tampered with. In this way it would be possible to board a passenger securely onto an international flight without requiring private information apart from their passport's photo. In the same fashion, this system for self-sovereign identity could just as easily be employed during the purchase of restricted goods such as tobacco or lottery tickets at a vendor point of sale without requiring the purchaser to display their private identity documents to the vendor.

5.3 A “BURN AFTER READING” MESSAGE PROTOCOL

Another novel idea for a project that could be built on the PoPMG oracle is a “burn after reading” message protocol. As its title suggests, this 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 for at depth 1, the oracle uses S^2 as the authentication proof needed to confirm *P* has access to decrypt the secret message encoded by the message's author. Using the PoPMG 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 author's encrypted secret message—from storage. The author's original message, in encrypted format mind you, might still be located by someone who knows the transaction hash of the NFT mint operation, but it won't be recoverable by anyone inspecting storage of of the oracle, rewards proxy or NFT contracts. In order to mitigate against this, we can use blinding to further obfuscate the author's secret message. Instead of storing a single message payload the oracle can store a dummy payload in storage which it will use to blind any secret message. Now, instead of storing a single bytes payload as the message content that's stored first in the oracle then later is moved to the NFT data, we can blind the real encrypted message by placing it inside a set type variable[15] along side multiple encrypted fake messages. The fake messages will need to be randomly seeded, not even the intended recipient *P* should be able to decrypt them. To recover the content's of the original message the messaging DApp needs to loop through the set of messages using S^1 to discover which 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 won't have any reasonable means of determining which is the blinded real message from among the 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. Another way in which our encrypted secret message can be obfuscated on the Tezos blockchain is it's advisable that each conversation should instantiate a new set of contracts, this is possible to do by the messaging DApp itself at conversation create time since the Taquito JavaScript SDK allows for contract origination.[16]

An interesting outcome of the above described message protocol is it eliminates the need to perform Diffie-Hellman key exchange. This means that messages can be exchanged between two parties with simple human readable password protection and without requiring access to a private key stored on a specific proprietary device. This effectively eliminates any need for importing keys or wallet mnemonics when transitioning between devices. It also means this system is sufficient for sending private data to yourself to be picked up from another device.

The following describes the Diffie-Hellman system which is the modern standard for private message authentication and is the authentication protocol used in 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.[17]

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 computation 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

[15] 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)

[16] See: https://tezostaquito.io/docs/originate/

[17] Ganjewar, 2010.

citizens' access to the Internet—as previously occurred in Egypt during the Arab Spring of 2011[18]—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[19] (and historically they have been), but in the case an activist's laptop and phone are seized and held captive by law authorities they could either become unable to access their messages or else worse, be compelled under duress or threat to unlock their devices and show the authorities the contents of their decrypted conversations. Conversely our proposed “burn after reading” message service with PoPMG managed messages i) continues to be accessible to *P* even if they are separated from their devices provided they find some way to access the Internet, ii) are resistant to government censorship because of decentralization; access to the messaging service can no longer be simply turned off by blocking a specific DNS, and iii) can allow for multiple parties to access the information without compromising security[20]—in such a case, the provers do not delete the message immediately from NFT storage but allow it to propagate first to the members of their group.

6 CONCLUSION

Privacy has long been a central aim in blockchain technology but what might have been considered private in 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.[21] 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 noteable achievement and one that we're very excited 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.

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

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

[20] 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)

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

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.