QAN - An energy-efficient quantum-resistant blockchain platform

Endre (Silur) Abraham
QAN Research

Johann Polecsak
QANtum Global OÜ

December 18, 2019

Abstract

We propose a novel blockchain platform with a fully quantum-resilient cryptography stack based on Lattices. QAN utilizes most of the Algorand consensus mechanism and is designed to benefit most from the upcoming 5G technology to enable seamless integration to V2X, Enterprise use cases and industry4 applications.

1 Introduction

Since their first implementations in 2015, smart-contract based blockchain platforms proved their relevance on various markets where participants can gain game-theoretic advantage by handling data privately. In the following years, initial problems like privacy, scalability, offchain protocols and ease of development were subject to exhaustive research. On October 23 2019, Google announced it's Quantum Supremacy by solving a decisional problem about the distribution of a given dataset of extreme size. While this is still far from a practical threat to conventional cryptosystems, a rigorous and sound system to address this future problem is needed especially in the blockchain ecosystem since it's adaptivity to changes relies mostly on it's community. We introduce QAN, a generic metered smartcontract platform with the main distinguishing feature that it arguably resists quantum adversaries. Most of our cryptographic primitives rely on worst-case Lattice problems that are known to be hard in AWPP and not in BQP.

1.1 Notation

In this paper we denote the finite field of integers modulo p with $\mathbb{Z}_p$. We use $\mathbb{Z}_p[x]$ to denote polynomials with coefficients from this field.
Let $\Phi(x)$ be an irreducible polynomial and $\mathbb{Z}_p[x]/\Phi(x)$ be a quotient ring with n nonzero coefficients. From now on, we assume n to be a power of 2 thus $\Phi(x) = x^n + 1$. We indicate uniform random sampling from a ring with $\xleftarrow{\$}$ and

discrete gaussian sampling with parameter σ with $\xleftarrow{D_\sigma}$. We also denote vectors and matrices in bold: $\mathbf{x} \in \mathbb{Z}^n$.

1.2 Primer on smart contracts

Following the notation of ethereum[1] we model smart contracts as transition functions between deterministic states:

$$\sigma_{t+1} = \Upsilon(\sigma_t, tx)$$

where σ and Υ are storage and computation components respectively. On a valid transaction (input) tx the global state σ advances according to the transition function that allows arbitrary operations on the state. Υ is also metered in atomic instructions and limited (by the network) to prevent DoS attacks. Upon a threshold-agreement protocol (consensus) of parties of opposing interests, the new value of σ is distributed among these parties. The history of these values forms a chain and their integrity is protected by inclusion of $H_1(\sigma_{t-1})$ into σ_t where H_1 is a cryptographic hash function. By their nature, smart contracts are not capable of accessing and querying information beyond their state and transactions as most I/O operations cannot be metered and they introduce nondeterminism and security issues. Instead, this information is also fed in forms of transactions and we call the parties feeding this information (which does not benefit the originator directly) oracles.

1.3 Primer on Quantum computing

A quantum system A is associated to a (finite-dimensional) complex Hilbert space H_a with an inner product $\langle\cdot|\cdot\rangle$. The state of the system is described by a vector $|\varphi\rangle \in H_a$ such that the Euclidean norm $\||\varphi\rangle\| = \sqrt{\langle\varphi|\varphi\rangle}$ is 1. Given quantum systems A and B over spaces H_a and H_B, respectively, we define the joint or composite quantum system through the tensor product $H_A \oplus H_B$. State changes of these systems are usually modelled with unitary transformations. Recall that a matrix is unitary if it's complex conjugate is equal to it's inverse. This form of modelling is convenient because unitary transformations are norm-preserving: $||Ux|| = ||x||$. This also implies that all quantum operations are linear and reversible. The product state of $|\varphi_A\rangle \in H_A$ and $|\varphi_B\rangle \in H_B$ is denoted by $|\varphi_A\rangle \oplus |\varphi_B\rangle$ or simply $|\varphi_A\rangle|\varphi_B\rangle$. An n-qubit system lives in the joint quantum system of n two-dimensional Hilbert spaces. The standard orthonormal computational basis $|x\rangle$ for such system is given by $|x_1\rangle \otimes \cdots \otimes |x_n\rangle$ for $x = x_1 \dots x_n$. Any (classical) bit string x is encoded into a quantum state as $|x\rangle$. An arbitrary pure n-qubit state $|\varphi\rangle$ can be expressed in the computational basis as $|\varphi\rangle = \Sigma_{x\in\{0,1\}}\alpha_x|x\rangle$ where α_x are complex amplitudes obeying $\Sigma_{x\in\{0,1\}^n}|\alpha_x|^2 = 1$ which we call the basis state. Otherwise we say that the qubit is in a superposition. Upon measuring a qubit, it will collapse into a 0-state with probability $|\varphi_A|^2$ and into a 1-state with probability $|\varphi_B|^2$.

1.4 Relations with cryptography

While the practical relevance of quantum computation on mainstream cryptography at the time of writing is still under debate, the theoretical capabilities and recent advances on the field are undeniable. We recall the two biggest theoretical threats to cybersecurity of today and argue on it's efficiency and pragmatic value.

1.4.1 Shor's algorithm

Given an integer n, the integer factorization problem is to find p, q such that $1 < p, q < N$ and $n = pq$. With p and q being primes of approximately same large bit-length, the problem is so hard that no polynomial algorithm exists in the bit-length of n. To best of our knowledge the best conventional algorithm for the task runs in $O(exp(\sqrt[3]{(\frac{64}{9}n(log\, n)^2)}))$ time[2]. In contrast, Peter Shor's novel work[3] introduced a quantum algorithm running in $O(n^3 logn)$ and using only $O(n^2 logn\, loglogn)$ gates. The first implementation of this algorithm was in 2001[4] and the largest number factored to date is 21[5].

It works by reducing the problem to period finding by exploiting the property that for all $k \xleftarrow{\$} \mathbb{Z}_n$ that are relative primes with n, there exists an x such that $k^x \equiv 1 \pmod{n}$. If this x is even then trivially $k^x - 1 = (k^{\frac{x}{2}} - 1)(k^{\frac{x}{2}} + 1)$ is a divisor of n. Otherwise if the difference between the two terms is 2 then it is also their largest common factor. We also know that p has to be a factor of one of these terms thus with finding x one can find factors of n too.

Let $s_a = a_i = x^i \pmod{n} \,|\, a_i \in 0, \ldots, n-1$. By the birthday paradox with $\approx \sqrt{n}$ trials we find indices $i_1, i_2 \,|\, a_{i_1} = a_{i_2}$. By selecting the smallest such indices the sequence, S_a becomes periodic and we can extract this period with the Quantum Fourier Transform. With the extracted period, by applying the rule above we found a factor of n. But this task is not easier with conventional approaches than directly finding factors with number sieves or Pollard's techniques. By further reducing the problem to evaluation of Jones polynomials, which is BQP-complete, and by the strong assumption that PH does not contain BQP[6], we argue that only quantum algorithms can solve the factoring problem with this period-finding strategy in polynomial time. We refer to the work of Aharonov et al. on the matter[7].

1.4.2 Grover's algorithm

The Grover search[8] is an optimal algorithm(no other quantum algorithm can achieve the same result with better complexity) to find a specific element in an unordered list of n elements in $O(\sqrt{n})$. In contrast with conventional methods, the minimal steps required for this is trivially n assuming a random ordering. While the generic description of this method is a search, it's applicable to a broader context, not restricting to lists or database elements. For example it's also capable of factorizing large numbers with a quadratic speedup to traditional methods. Let ψ be the uniform superposition and O an oracle operator. The

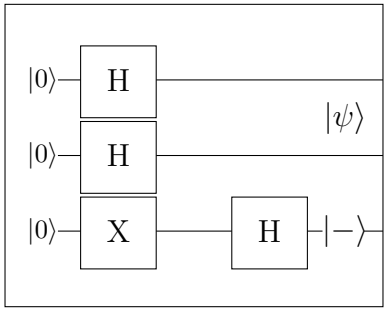

Figure 1: Initialization of states

key concept to the algorithm is the Grover operator:

$$G = (2|\psi\rangle\langle\psi| - I)O.$$

Denoting the finite sum $\frac{\sum_i a_i}{\sum_i 1}$ by $\langle a\rangle$, applying the operator on any state results in:

$$\sum_i (2\langle a\rangle - a_i)|i\rangle$$

We also have to initialize $|x\rangle|q\rangle$ into the uniform superposition ψ and into $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ respectively by using the product of a Hadamard transform $H^{\otimes_n(|x\rangle)}$, a Pauli-X transform and another Hadamard transform1. After reaching these states, applying the Oracle operator negates the overall state amplitude while the Grover operator decreases the amplitude of all states we are not interested in. After $\approx \lceil \frac{\pi 2^{\frac{n}{2}}}{4} \rceil$ iterations, we should be able to measure a non-negligible amplitude difference on our desired state with probability $> \frac{1}{2}$.

1.5 Lattices

1.5.1 Lattices in general

A lattice is the set of all possible linear combinations of a finite real basis with integer coefficients:

$$c \in \mathbb{Z}^n, \mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^m, \Lambda = \{\mathbf{Bc} = \textstyle\sum_{i\in[n]} c_i \mathbf{b}_i\}$$

The dimension of a lattice is m and it's rank is n. In case $n = m$ we call the lattice full-rank. For cryptographic applications we mostly use full-rank lattices. The dual of a lattice is $\{x \in \mathbb{R}^n : \forall v \in \Lambda, \langle x, v\rangle \in \mathbb{Z}\}$ and the bidual is itself. The shortest nonzero vector in the lattice is called the minimum distance: $\lambda_1(\Lambda) = min_{0\neq x\in\Lambda}||x||$. A measurable set $\mathcal{R}(\Lambda) \subset \mathbb{R}^m$ satisfying that $\cup_{\lambda\in\Lambda}\mathcal{R}(\Lambda) + \lambda = \mathbb{R}^m$ is called a fundamental region. The fundamental parallelepiped is a

fundamental region, namely the generating set $\mathbf{G} \in \mathbb{R}^m$ where $\sum_{i\in[n]} \mathbf{g}_i a_i \in \mathbb{Z}$. All fundamental region's volume is equal to $det(\mathbf{B})$. We use the notation $\mathcal{B}$ for the closed Euclidean ball of radius 1 around the origin: $\mathcal{B} = \{\mathbf{w} \in \mathbb{R}^n : ||\mathbf{w}|| \leq 1\}$. Besides being (so far) resistant to quantum attacks (not restricting to Shor's and Grover's algorithm) their significance in cryptography is stronger, as they offer better asymptotic complexity (most lattice algorithms only involve linear operations on 16bit integers), and it's the ideal platform group for today's most exotic cryptosystems like FHE, IO or (H)IBE.

1.5.2 Discrete Gaussians and the Smoothing parameter

The connection between gaussian distributions and maps to the lattice's fundamental parallelepiped allows us to prove security easier than trying to use uniform distribution (for e.g. key-generation) directly. We recall the properties of discrete gaussian sampling over lattices and other related definitions based on the works of Regev[9]:
For all $\mathbf{c}, \mathbf{x}, s > 0$, let:

$$\rho_{s,\mathbf{c}}(\mathbf{x}) = e^{-\pi||(\mathbf{x}-\mathbf{c})/s||^2}$$

be the Gaussian function with center $\mathbf{c}$ and scale s. Then the continuous gaussian distribution is: $\frac{\rho_{s,\mathbf{c}}(x)}{\mathbf{s}^n}$. This distribution is generalized for lattices as: $\frac{\rho_{s,\mathbf{c}}(x)}{\rho_{s,\mathbf{c}}(\Lambda)}$. As the scale parameter gets larger, the statistical distance between the continuous and discrete variants of the gaussian narrows, and the threshold for s where the distance from $\mathbf{c}$ is (almost) $\frac{s^2 n}{2\pi}$ is called the smoothing parameter:

$$\epsilon > 0 \in \mathbb{R}, \eta_\epsilon(\Lambda) = \rho_{1/s}(\Lambda^* \setminus \{0\}) \leq \epsilon$$

where Λ^* is the lattice's dual. This smoothing parameter was the key to Regev's results[9] as they proved that by using discrete Gaussian distributions beyond this parameter, reducing the results into the fundamental parallelepiped resulted in a statistically close to uniform distribution over $\mathbb{Z}_q^n$. From Peikert's lemma[10] we know a lower bound on η:

$$\eta_\epsilon(\Lambda) \leq \frac{\sqrt{log(2n/(1+\frac{1}{\epsilon}))/\pi}}{\lambda_1^\infty(\lambda^*)}$$

where $\lambda_1^\infty(\lambda^*)$ is the minimum distance of the lattice's dual using the infinity norm.

1.5.3 Hard problems and Trapdoors on lattices

We recap some problems on full-rank and regular lattices which are assumed no to be in BQP as no quantum algorithm currently exists that can solve an instance in polynomial time.

1. Shortest Vector Problem: *With a lattice basis* $\mathbf{B}$ *and* $l \in \mathbb{R}$*, determine whether* $\lambda_1(\mathbf{B}) \leq l$

2. Closest Vector Problem: *With basis* $\mathbf{B}$*,* $t \in \mathbb{R}^n$*,* $d \in \mathbb{R}$*, and security parameter* k *(polinomial in* n*) determine whether* $dist(t, \Lambda(B)) > kd$

3. Covering Radius Problem: *With* $d \in \mathbb{R}^n$ *determine whether* $max_{x \in \mathbb{R}^n}\{dist(x, \Lambda(B)\} \leq d$

4. Shortest Indepentend Vectors Problem: *Search for vectors* $\mathbf{S} \subset \mathbf{B}$ *for basis* $\mathbf{B}$ *that satisfies* $||\mathbf{S}|| < k \cdot min\{r : dim(span(\Lambda \cap r\mathcal{B})) \geq n\}$

5. Shortest Integer Solution: This is a worst-case generalization of (approximate) SVP. *For* $m = poly(n), q \geq \beta\sqrt{n}\omega(\sqrt{logn})$ *find* $\mathbf{x} \in \mathbb{Z}^m$ *such that* $\mathbf{Az} = \mathbf{0} \wedge ||x|| < \beta$

6. Learning with errors[11]: *For an arbitrary* $\mathbf{s} \in \mathbb{Z}_q^n$*, the decisional LWE asks to differentiate a "sample"* $(\mathbf{a}, as + e \mod q)$ *from a completely uniform random sample where* $a \xleftarrow{\$} \mathbb{Z}_q^n$ *and* $e \xleftarrow{D_\sigma} \mathbb{Z}_q$. LWE is proven to be as hard as SIVP when $q \geq 2\sqrt{n}/\sigma$.

7. Ring-Learning with errors: *Let* $\Phi(x)$ *be a cyclotomic polynomial and* $\mathbb{G} = \mathbb{Z}_q[x]/\Phi(x)$ *be a quotient polynomial ring. For an arbitrary* $s(x) \in \mathbb{G}, a(x) \xleftarrow{\$} \mathbb{G}, e(x) \xleftarrow{D_\sigma} \mathbb{G}$*, differentiate the sample* $(a(x), a(x) \cdot s(x) + e(x) \mod q)$ *from a uniformly selected one.*

While Ajtai's groupdbreaking work[12] on connecting the average and worst case complexities of these problems (especially SVP) allowed researchers to construct hashes and encryption systems, the problems by themselves were not sufficient for e.g. a signature system. An abstraction called "preimage sampleable functions"[13] were introduced by Gentry et al. as a replacement for trapdoor permutations over lattices to serve as more generic trapdoors, and some improvements on complexity and size is due to Peikert and Micciancio[14].

1.6 The Glyph signature system

QAN transaction signing uses the Glyph scheme[15], which is a generalization of GLP[16] with some updated parameters and under a stricter set of assumptions from NewHope[17]. Glyphs security assumption relies on the hardness of RLWE. It's parametered by $m = 2048, n = 1024, q = 59393, \omega = 16, k = 256, B = 16383$, utilize a collision-resistant hash function $H : \{0,1\}^* \to 0, 1^k$, an encoding function $E : \{0,1\}^k \to S_\omega$ where S_ω is the collection of ω-sparse elements of an RLWE cyclotomic ring: $c_i \pm 1 \mod q$. Glyph consists of 3 PPT algorithms. In KeyGen1.6 we choose the secret and error polynomials such that their infinity norm (in case of $\mathbb{G}$, it's the largest coefficient) is at most 1. Our secret key is the pair of these polynomials and the public key is the RLWE sample $a(x) \cdot s(x) + e(x)$.

Algorithm 1: KeyGen

Data: $a \xleftarrow{\$} \mathbb{G}$ (as in RLWE7)
Result: Keypair (sk, pk)

1 $\mathbf{s}, \mathbf{e} \xleftarrow{D_\sigma} \mathbb{G} : ||s_i||_\infty < 1 \wedge ||e_i||_\infty < 1$
2 $\mathbf{pk} \leftarrow \mathbf{as} + \mathbf{e}$
3 $sk \leftarrow (s, e)$
4 return (sk, pk)

Algorithm 2: Sign

Data: message m, public polynomial coeffs a,
private key (s, e), public key $t = as + e$
Result: Signature (z_1, z_2, c)

1 $y_1, y_2 \xleftarrow{D_\sigma} : ||y_i||_\infty < B$
2 $c' \leftarrow H(\lceil ay_1 + y_2 \rfloor_{B-\omega} | m)$
3 $c \leftarrow E(c')$
4 $z_1 \leftarrow sc + y_1$
5 $z_2 \leftarrow ec + y_2$
6 **if** $||z_1||_\infty > B - \omega \vee ||z_2||_\infty > B - \omega$ **then**
7 goto 1;
8 **end**
9 $z_2 \leftarrow Compress((az_1 - tc), z_2)$ return (z_1, z_2, c)

In Sign4 we select two "small" polynomials y_1, y_2 make an RLWE sample and use it as a commitment "salt" for m in our hash function. We also provide reconciliation information z_1 and z_2 which will allow the verifier to ensure authenticity of the RLWE sample. We use rejection sampling for these reconciliation polynomials. The probability of rejection is $\approx \frac{1-(2(B-\omega)+1)}{2B+1}$. Memory requirements of a signature is $nlog_2(2(B-\omega)+1)$ bits for z_1, $2n+(\lceil log_2(2(B-\omega)+1) \rceil)\frac{6(B-\omega)n}{q}$ bits for z_2 and $\omega log_2(2n)$ for c. For $Compress$ we define the K-rounding function[15] on $\mathbb{G}$

$$\forall x_i, r_i \in \mathbb{Z}, s_i \in [0, 2K] \,|\, x_i \mod q = r_i(2K+1) + s_i$$

and

$$\lfloor \textstyle\sum_i x_i \zeta^i \rceil_k = \sum$$

With good parameter selection, the $Compress9$ function has much better probability to succeed with rejection sampling than the original GLP scheme[16]

And finally in the verifier step15 we reconstruct the one-time masking sample we used together with the message using the reconciliation information given by the signature

We refer to the original GLYPH paper[15] for security proofs and efficiency considerations.

Algorithm 3: Compress

Data: $y \in \mathbb{G}, z \in \mathbb{G} : ||z||_\infty < K$
Result: $z' \in \mathbb{G}$

```
for i ← 0 to n − 1 do
    if ⌊y_i + z_i⌋_K = ⌊y_i⌋_K then
        z'_i = 0
    end
    if y_i ∈ [0, K) then
        z'_i = −K
    end
    if y_i ∈ [q − K, q) ∧ z_i > 0 then
        z'_i = K
    end
    if ⌊y_i + z_i⌋_K < ⌊y_i⌋_K then
        z'_i = K
    end
end
return z'
```

Algorithm 4: Verify

Data: message m, CRS a, public key t,
signature (z_1, z_2, c)

```
if ||z_1||_∞ > B − ω ∨ ||z_2||_∞ > B − ω then
    return Reject
end
d' ← H(⌈az_1 + z_2 − tc⌋_{B−ω}|m) d ← F(d' return
d == c
```

1.7 Fully Homomorphic encryption with GSW and IBE

We define homomorphic encryption as a public key cryptosystem where there exists an operation $\otimes$ such that $Enc_{sk}(m_1) \otimes Enc_{sk}(m_2) = Enc_{sk}(m_1 + m_2)$. In other words homomorphic encryption allows to preserve some operations called gates on the ciphertexts under the same key. We have different types of these schemes like *somewhat homomorphic encryption* where one is only allowed to evaluate a subset of gates, *leveled* where arbitrary operations are allowed but for a limited depth, and *fully homomorphic encryption* where there's no bound on depth either (or it's negligible). We stress the importance of FHE in the blockchain ecosystem even with their enormous storage requirements since publicly available plaintext data on the chain restricts the application of privacy-respecting logic.
We also introduce the definition of ABE and IBE. With abuse of formal definition we can say that Attribute based encryption (ABE) is a form of encryption

where Alice's public key have special attributes which enables her to decrypt messages only directed towards users with the same attribute(s). IBE is a restriction of ABE where only one user has the subject attribute - which is her identity in fact. In these systems one can for example "ask" that her public key would represent her email address, phone number or passport number which would eliminate the need of exchanging keys and matching QR codes for payments. Key generation for both GLYPH and the following scheme is (optionally) possible in an IBE setting but only in a trusted setup, meaning that there is an issuer of the identity or attributes in question who helps with key generation. While there exists trustless solutions for the problem[18] they utilize Indistinguisability Obfuscation which is still unfeasible on smart contracts in practice.

We selected the GSW cryptosystem as our basis for private on-chain storage which is capable of both FHE, IBE. Of the other promising and more efficient candidates like TFHE[19], even though GSW operates on bit-level ciphertexts (or $\mathbb{Z}_q$ in the generic case[14]), it also has the advantage that it's homomorphic operations don't require a bootstrapping or public key. Given that our Lattice systems already require enormous storage, eliminating the need for large "evaluation" keys and making FHE operations more CPU-intensive is a compromise from our end. Note that GSW security is modelled on LWE[20] in contrast with our signatures. We need to introduce some some utility functions before defining GSW. With parameters shown in section 1.5.36 and $l = \lfloor log2(q) \rfloor + 1$, we define $Powers2(\mathbf{x}) = (b_1, 2_b2, 4b_3, \ldots, 2^l - 1b_1, b_2, 2b_2, 4b_2, \ldots, 2^l b_2, \ldots, 2^{l-1}b_k)$. Then let $BitDecomp(\mathbf{x}) = (b_{i,0}, b_{i,1}, \ldots,$
$b_{i,l}, \ldots, b_{k,l})$ where $b_{i,j}$ is the jth bit of b_i in the vector, and the inverse function $BitDecomp^{-}1(\mathbf{y}) = (\sum 2^j a_{1,j}, \ldots, \sum 2^j a_{k,j})$. Now, because in most homomorphic constructions from LWE, the error term grows doubly exponentially, we need a way to bound this error so they won't affect decryption even after lots of operations evaluated. The function $Flatten(\mathbf{x}) = BitDecomp(BitDecomp^{-1}(\mathbf{x}))$ serves this function and it's important to note that $\langle BitDecomp(\mathbf{x}), Powers2(\mathbf{y}) \rangle$
$= \langle x, y \rangle$.
Now with all functions present, we define the GSW cryptosystem of 3 PPT algorithms $keygen1.7$, $encrypt8$, $decrypt3$ as shown:

For a version of *decrypt* that's capable of recovering m from a larger plaintext space we refer to the algorithm developed my Micciancio and Peikert[14]
GSW's homomorphic operations are:

1. Constant multiplication with α - $\alpha C = Flatten(Flatten(\alpha I_n)C)$
2. Addition - $C_1 + C_2 = Flatten(C_1 + C_2)$
3. Multiplication - $C_1 C_2 = Flatten(C_1 C_2)$
4. NAND - $C_1 \uparrow C_2 = Flatten(I_n - C_1 C_2)$

The final error bound of a L-level NAND circuit is $(N+1)^L \cdot B$ *where* B *is the original error bound of a fresh encryption*[20]

Algorithm 5: Keygen

Data: Λ
Result: $sk \in \mathbb{Z}_q^{n+1}$, $pk \in \mathbb{Z}_q^{m \times n+1}$, $\mathbf{v} \in \mathbb{Z}_q^{n+1}$

$\mathbf{t} \xleftarrow{\$} \mathbb{Z}_q^n$
$\mathbf{sk} \leftarrow (-1, -t_1, -t_2, \ldots, -t_n) \in \mathbb{Z}_q^{n+1}$
$v \leftarrow Powers2(\mathbf{sk})$
$\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$
$\mathbf{e} \xleftarrow{D_\sigma} \mathbb{Z}_{\|}{}^m$
$\mathbf{b} \leftarrow \mathbf{Bt} + \mathbf{e}$
$\mathbf{pk} \leftarrow [\mathbf{b}|\mathbf{B}]$
return (sk, v, pk)

Algorithm 6: Encrypt

Data: Λ, pk, message m
Result: ciphertext $\mathbf{C} \in \mathbb{Z}_q^{N \times N}$

$\mathbf{R} \xleftarrow{\$} \{0,1\}^{N \times m}$
$C \leftarrow Flatten(mI_N + BitDecomp(Rpk))$
return C

1.8 SIS-hash and accumulation

In this section we recall the hash function proposed by Regev[9] that relies on the hardness of SIS, and show its accumulating properties. We argue that the current security of SHA against quantum adversaries rely on an architectural assumption, not a mathematical one. It's current resilience is due to the short decoherence time of logical qubits, and the bounds shown by Amy et al.[21] applies to a periodic surface coded error-correcting model. In their research they also showed that such implementation would require 256 qubits to run (reversibly). Also, there are other models like Topological quantum computing where the environmental noise does not intervene with the coherence of qubits as braids are topologically invariant to most distortion. With SHA lacking more generic security assumptions against quantum attacks, "lattice hashers" started to emerge from Ajtai's famous example[12] which doesn't rely on architectural and implementation limits. Let $A \in \mathbb{Z}^{m \times n}$ as in SIS5. Then finding a preimage of the operation $Ax \mod q$ where $x \in \{0,1\}^m$ is equivalent of solving a worst-case instance of SIS.

Proof:[22]: *If we can find two distinct binary strings s_1, s_2 such that $As_1 \equiv As_2 \pmod q$ then we have $M(s_1 - s_2) \equiv 0 \pmod q$. Since $s_1, s_2 \in \{0,1\}^m$, we have $x \overset{def}{=} (s_1 - s_2) \in \{-1, 0, 1\}^m$ which constitutes to a solution of SIS*

As for using the SIS-hasher for hash accumulation we see that $y_{i+1} = y_i + Ax \mod q$ can also be reduced to an instance of SIS and satisfies the needed

Algorithm 7: Decrypt

Data: Λ, $\mathbf{v}$, ciphertext $\mathbf{C}$
Result: plaintext $m \in \mathbb{Z}_q$
$\mathbf{x} \leftarrow (C_1 v, C_2 v, \ldots, C_N v)$
select v_i from $\mathbf{v}$ where $\frac{q}{4} < 2^i <= \frac{q}{2}$
$m \leftarrow \lfloor x_i / v_i \rceil$
return m

(quasi)commutativity properties. The accumulator value is y_i, witness is generated for value z as $y_i + (Az)^{-1} \mod q$, and revocation is done by accumulating a value's modular inverse. We see that this method is not only intuitive, hardware-efficient and resistant to all current (theoretical) quantum adversaries but is constant time in append and revocation complexity too. In later sections we use the notation SIS_y^x to denote a hash accumulator which accepts inputs of length x and the resultin hash is of length y. Wee see that to produce an y bit output we have to reduce a vector $\mathbb{Z}_q^l$ into element-wise binary representation such that $y = l \cdot \lceil log_2(q) \rceil$.

2 QAN specification

We now present the specifications for QAN platform, that we split into three layers:

2.1 Epidemic pubsub gossip

Our p2p network consists of an epidemic-style gossip pubsub protocol where nodes are only assumed to have knowledge only about their neighbours. Initial PEX (peer exchange) happens trough a selection of bootstrap nodes that are preferably discarded after the bootstrap phase to encourage decentralization. Nodes maintain a heartbeat protocol only with their neighbours and do not distinguish between them while communicating. All nodes are to broadcast every message to every neighbour and filtering based on authenticity, validity and topic is up to the receiver. While the selection of a pubsub protocol within the constrained world of blockchains seem unintuitive we went with the idea to allow more complex protocols on the existing overlay network like state channels or synchronous subnetworks. From the built-in topics $blocks, txs$, synchronizing $blocks$ enjoy priority in implementations. About syncronicity assumptions, the implementation of this protocol should be asynchronous and synchrony is formed on a higher abstraction level explained in the next section.

2.2 PPoS consensus

The distributed nature of blockchains require a much more restricted form of transaction ordering and consensus than what "conventional" distributed com-

puting is used to. Users usually are not aware of network topology and are game-theoretically working against each other. This eliminates the possibility of Paxos or RAFT-like agreement protocols. Users should be able to self-propose themselves with a cryptographically sound proof of authenticity (to create a block) rather than rely on others to be selected. The first form of such sound proof used in blockchains is HashCash[23] Proof-of-Work. Here, peers append to the chain by searching for a nonce that produces a hash value to fall below a certain difficulty and the longest such chain is considered authorative. While this method easily satisfies the non-interactive and polylogarithmic verifiability requirement for a blockchain consensus, it has its own downsides:

1. PoW schemes rely on too much trial-and-error on high difficulty which doesn't scale in computational power.
2. PoW schemes assume that at least 51% of the networks computational power comes from honest nodes.
3. While the underlying hash functions are statistically unbiased, one can achieve game-theoretic advance of others based on computational power which leads to centralization[24][25][26].
4. PoW allows the possibility of forking[27][28]

Algorand (or Pure PoS) is a novel random-committee selection based PoS consensus algorithm developed by Silvio Micali[29] capable of 750Mb of transactions per hour which is 125 times more than what BTC is capable of at time of writing. The unbiased randomness of the leader election is provided through VRFs[30]. VRFs can be thought of HMACs with asymmetric keys or commitments with two openings where one witness (the VRF proof) does not leak any information about the other (the input). First, parties agree on a cryptographically secure pseudorandom function (like SHA) and later on, prove that a preimage x resulted in value y without actually revealing the input, which would be the case in plain SHA commitments. VRFs generally consist of 4 PPT algorithms:

1. $G(\lambda) := (sk, pk)$ a key generator
2. $E(sk, m)$ an evaluator of the pseudorandom function
3. $P(sk, m)$ a proof generator
4. $V(pk, m, \alpha)$ a verifier

We selected the quantum-safe generalization of the Franklin-Zhang scheme from our previous work[31] as a basis VRF.

The original BA$\star$ protocol assumes strong-synchronicity for final consensus which we expect not to scale to large number of users if approached directly on the network level. To mitigate this, we instead emulate a synchronous design on a fully asynchronous network using Threshold Logical Clocks[32]. TLC's allow

higher levels of an asynchronous protocol to act as it was a synchronous one, not requiring common coins, fixed rounds and epochs for coordination. While similar to Awerbuch's synchronizers[33], TLCs allow faulty and byzantine nodes into its abstraction of synchronicity.
Like Lamport clocks but unlike vector or matrix clocks, within TLC, nodes are only allowed to step their internal logical clock after processing (and gossiping) a threshold number of other nodes' messages. We briefly recap some adversarial behaviours in TLC[32], where a node may:

1. advance its clock before reaching the target threshold
2. not advance its clock after reaching a witness threshold
3. claim a message has been threshold witnessed
4. turn back its clock

By using TLC in a blockchain setting, we see that we can mitigate some of these adversarial behaviours with its inherent properties (like append-only logging). TLC allows the usage of threshold signatures for cryptographically verifying witnesses but such system assumes topological knowledge about the network on node level which also undermines scalability. For this reason we address the problem of false threshold witnessing in the next steps of our protocol. By the asynchronous nature of our system and the FLP result, the only defense for Eclipse attacks against our gossip and TLC network is the final consensus, meaning we consider an eclipsed or DoS-ed full-node faulty or byzantine.

2.2.1 The voting precompile

The cornerstone of our PPoS algorithm is the precompiled contract under the address $0x00000000000000000001$. This contract is responsible for endorsing stakes (or more precisely consensus tickets) and VRF public keys and selection thresholds for later verification. This means a user cannot take part in the consensus mechanism without registering their public key in the precompile. Although the precompile has a fixed stake "price" for participation, our threat model allows an adversary with multiple addresses registered, which is why we use a PoS game model. This price is defined in Spice which is the fixed measure of execution on our virtual machine explained later. While the amount of Spice to be paid is fixed for all contracts, outside the virtual machine it is settled between parties with QARKS (QAN's internal hard currency) to allow economic flexibility and dynamic pricing.

After the VRF public key vpk and the address is registered to the precompile trough a SIS accumulator, a block proposer in block B_{i-1} serialize it's mempool of transactions, seals it with a VRF proof $(vrfHash = E(vsk, block), vrfProof = P(vsk, block))$ and makes a witness of public key inclusion from the current precompile accumulator value $w = acc - SIS(pk)$. This VRF sealed block is gossiped to the network using the TLC model. Upon receiving a proposal for B_i on the network, a node searches for the sending peer's public key from the

precompile's publicly available list and verify the VRF proof. Upon having t VRF proofs in the gossip network where the VRF output falls below d where t and d is present in the precompile we either output the empty block if no majority is reached within the t proposals, or output the block that reached majority. We see that in contrast with PoW, uniformity on our proposals are achieved in a single calculation of a hash function (as opposed to PoW's millions of nonce-trials) and the overhead of a VRF proof generation. This means we reach the same non-interactive uniformity requirements with magnitudes less computational advantage.

2.3 Virtual machine

QAN's virtual machine is a metered generic smart contract VM as seen in ethereum[1]. This means that the machine consist of atomic operations with corresponding fixed price of execution representative of its complexity and resource needs on the blockchain. We call this soft internal currency Spice. We also follow the "Account-based" state model meaning there is a storage root (of SIS-hashes) and a code hash that gets executed with the transaction data as an argument if present. QAN's VM consists of a WebAssembly runtime to allow developers flexibility on lanaguages while targeting smart contracts and different language implementations shall provide a namespace of blockchain-specific functionality injected into the webassembly binary instance. As LLVM has a webassembly backend this allows a wide number of mature languages to target the QAN platform.

Webassembly operates on 32 or 64bit words and a Memory table that grows on explicit request. Additional, non-wasm instructions like SSTORE and FHEL are provided with in the interpreter's import object on instantiation, along with necessary Memory table entries like the caller, origin and balance. Addresses are 20 bytes long and calculated by hashing the public key and dropping the first 12 bytes. Memory and stack are volatile meaning they is discarded after smartcontract execution. Permanent storage is addressed by the SSTORE and SLOAD opcodes and is part of the global VM state.

2.3.1 Handling private data

QAN's virtual machine is capable of entering a private context trough the $FHEE$ and $FHEL$ opcodes. Entering this context and setting the context flag notifies the runtime that compatible opcodes should be interpreted in their homomorphic context on ciphertexts. $PUSH$ instructions in FHE context expects a GSW-encrypted ciphertext which can be decrypted locally by the key holder. Smart contracts can operate on private encrypted data on-chain this way and still able to preserve internal integrity. Some example use-cases for such contracts are:

1. Anonymous voting
2. Medical/Epharma databases

3. Asset management

4. Dark pools

2.4 Transaction lifecycle

Creating a plain transaction (sending QARK or interacting with a contract) happens by filling up the structure shown in Appendix A4. After SIS-hashing and GLP-signing it, it gets broadcasted to the TLC-pubsub network. Nodes validate the authenticity and integrity of the transaction, run it trough their runtime, and update the global state root and account storage root according to the VM output. After a TLC and VRF threshold is achieved, the transaction is included in the block and it is accumulated into a SIS_{32} accumulator in the block header. We note that because of the commutativity of our hash accumulator, this value is not sufficient in itself to check the ordering of transactions!
Creating a contract, again following the conventions laid by ethereum, happens by making a transaction to the precompile $0x0000000000000000000$ and filling the *data* field with code that returns the contract's bytecode. The resulting contract's address is computed from the creator's address and nonce as $SIS^{256}_{160}(SHA(addr|nonce))$ Transactions are collected in a node's mempool and ordered in a merkle tree. On receiving a transaction and running it trough the VM, a receipt is calculated from it's Spice usage and result state and these are also collected in a merkle tree. Upon falling within the VRF threshold by the PPoS system, the root of this tree is appended to the block header along with it's parent block hash.

2.5 Reducing storage with Secure Fountain Codes

Secure Fountain Codes[34] are a generalization of Luby Transform codes[35] with the addition that its peeling-decoder is resilient against malicious code blocks by using the block headers. Storing the blockchain with SeF is done in epochs, which is a predefined number of block intervals. At every k-th block, nodes select $d \xleftarrow{D_d} [1, k]$ where D_d is a robust degree distribution[35]. Then it selects d random blocks from the epoch interval, and forms a droplet by XORing them together (with necessary padding) and storing the the indices d. This process is repeated s times which is also a system-wide parameter.
Decoding a blocks is done as follows[34]:

1. Make a bipartite graph G with k left vertices B_i and ns right vertices $C_i$where n is the number of nodes in the peer's neighborhood

2. Connect B_i with C_i if B_i was XOR-ed into C_i with knowledge of d in every C_i

3. Find a singleton droplet C_i which as exactly one edge. If there's no such droplet halt with error.

4. Check the singleton's integrity using the block headers

5. XOR the singleton droplet to all of it's neighbours to recover B_i
6. Disconnect the singleton and all it's neighbors from its corresponding B_i.
7. If all B_i are recovered halt, otherwise goto step 3

The expected storage ratio between the full chain and the SeF encoded one on a single node is $\approx k/s + O(1)$ which complements the enormous storage assumptions of our cryptographic primitives from a node's perspective. We let the decision whether to utilize SeF codes in a QAN implementation to the reader, as the indirect economical tradeoff between storage costs and network communication vary on many topics such as geolocation.

3 Discussions

QAN's experimental blockchain is designed to research quantum adversaries in distributed systems. There are undeniable shortcomings of utilizing post-quantum and especially Lattice-based cryptography in such ecosystems. First and most importantly, while these topics have been around for a while in literature, the industry is yet to prove its relevance and limits. We designed QAN to be a collection of such academic advances to collect practical results on the matter. Generally, the most relevant security "proof" in cryptography is time, and since PQC is currently under standardization within NIST, our efforts on exploring a fully post-quantum cryptographic stack come with risks.
Secondly, although Lattice-based signatures and encryption systems improved a lot in computational complexity, they are still gargantuan in public key and signature size. Even Glyph1.6, which is considered a state-of-the-art signature system, comes with public key sizes of 2Kb in compressed form. Ethereum at the time of writing consumes $1.4Tb$ of storage using a parity "archive" node, which QAN is expected to reach in 55 days (without SeF codes), assuming PPoS 750Mb/hour performance is maxed. While mitigations to the problem is under heavy research, the compromise between quantum safety and storage costs are clearly visible and should be taken into consideration with all use cases.

4 Appendix

4.1 Transaction layout

```
Transaction ::= SEQUENCE {
        nonce INTEGER,
        timestamp DATE-TIME,
        spicePrice INTEGER,
        spiceLimit INTEGER,
        to OCTET STRING SIZE(20),
        value INTEGER,
        data OCTET STRING,
        signature OCTET STRING
}
```

4.2 Block layout

```
Block ::= SEQUENCE {
        parentHash OCTET STRING SIZE(32),
        index INTEGER,
        timestampe DATE-TIME
        stateRoot OCTET STRING SIZE(32),
        transactionsRoot OCTET STRING SIZE(32),
        receiptsRoot OCTET STRING SIZE(32),
        spiceLimit INTEGER,
        spiceUsed INTEGER,
        extradata OCTET STRING,
        vrfHash OCTET STRING SIZE(32),
        vrfProof OCTET STRING SIZE(32),
        vrfPubkey OCTET STRING,
        vrfAccWitness OCTET STRING,
        sig OCTET STRING
}
```

4.3 Wasm operations and Spice costs

We refer to the original Webassembly specification (*https://webassembly.github.io/spec/core/syntax/instructions.html*) for terminology and notation of instructions while defining our Spice costs:

instruction	Spice cost
inn.const	2*(nn/32)
fnn.const	3*(nn/32)
inn.iunop	2*(nn/32)
fnn.funop	3*(nn/32)
inn.ibinop	2*(nn/32)
fnn.fbinop	3*(nn/32)
inn.itestop	2*(nn/32)
inn.irelop	2*(nn/32)
fnn.frelop	3*(nn/32)
i32.wrap_i64	8
i64.extend_i32_sx	8 + 2*sx
i32.trunc_fmm_sx	2*(nn/32) + 2*(mm/32) + 2*sx
f32.demote_f64	12
f64.promote_f32	12
fnn.convert_imm_sx	2*(nn/32) + 2*(nn/32) + 2*sx
inn.reinterpret_fnn	8
fnn.reinterpret_inn	8
local.get, local.set	8
global.set, global.get	8
local.tee	8
inn.load	80
inn.store	800
inn.loadB_sx	B*25
inn.loadB_sx	B*250
CREATE	32000
CALL	700
BALANCE	700
CHAINID	2
FHEE	12000
FHEL	12000
SSTORE	20000
SLOAD	2000

References

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger,"

[2] C. Pomerance and P. Erdös, "A tale of two sieves," 1998.

[3] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, 10 1996.

[4] L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood, and I. Chuang, "Experimental realization of shor's quantum factoring algo-

rithm using nuclear magnetic resonance. nature, 414:883," *Nature*, vol. 414, pp. 883–7, 12 2001.

[5] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X. Zhou, and J. O'Brien, "Experimental realisation of shor's quantum factoring algorithm using qubit recycling," *Nature Photonics*, vol. 6, 11 2011.

[6] R. Raz and A. Tal, "Oracle separation of bqp and ph," in *STOC 2019 - Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (M. Charikar and E. Cohen, eds.), Proceedings of the Annual ACM Symposium on Theory of Computing, pp. 13–23, Association for Computing Machinery, 6 2019.

[7] D. Aharonov, V. Jones, and Z. Landau, "A polynomial quantum algorithm for approximating the jones polynomial," *Algorithmica*, vol. 55, pp. 395–421, Nov 2009.

[8] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING*, pp. 212–219, ACM, 1996.

[9] D. Micciancio and O. Regev, "Worst-case to average-case reductions based on gaussian measures," vol. 37, pp. 372– 381, 11 2004.

[10] C. Peikert, "Limits on the hardness of lattice problems in lp norms," *Computational Complexity*, vol. 17, pp. 300–351, 05 2008.

[11] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, 01 2009.

[12] M. Ajtai, "Generating hard instances of the short basis problem," in *Automata, Languages and Programming* (J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds.), (Berlin, Heidelberg), pp. 1–9, Springer Berlin Heidelberg, 1999.

[13] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions." Cryptology ePrint Archive, Report 2007/432, 2007. `https://eprint.iacr.org/2007/432`.

[14] D. Micciancio and C. Peikert, "Trapdoors for lattices: Simpler, tighter, faster, smaller." Cryptology ePrint Archive, Report 2011/501, 2011. `https://eprint.iacr.org/2011/501`.

[15] A. Chopra, "Glyph: A new instantiation of the glp digital signature scheme." Cryptology ePrint Archive, Report 2017/766, 2017. `https://eprint.iacr.org/2017/766`.

[16] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," vol. 7428, pp. 530–547, 09 2012.

[17] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange - a new hope." Cryptology ePrint Archive, Report 2015/1092, 2015. https://eprint.iacr.org/2015/1092.

[18] S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi, "Registration-based encryption: Removing private-key generator from ibe." Cryptology ePrint Archive, Report 2018/919, 2018. https://eprint.iacr.org/2018/919.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds." Cryptology ePrint Archive, Report 2016/870, 2016. https://eprint.iacr.org/2016/870.

[20] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based." Cryptology ePrint Archive, Report 2013/340, 2013. https://eprint.iacr.org/2013/340.

[21] M. Amy, O. D. Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. M. Schanck, "Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3," in *SAC*, 2016.

[22] O. Goldreich, S. Goldwasser, and S. Halevi, "Collision-free hashing from lattice problems," 02 2000.

[23] A. Back, "Hashcash - a denial of service counter-measure," 09 2002.

[24] L. Luu, R. Saha, I. Parameshwaran, P. Saxena, and A. Hobor, "On power splitting games in distributed computation: The case of bitcoin pooled mining." Cryptology ePrint Archive, Report 2015/155, 2015. https://eprint.iacr.org/2015/155.

[25] I. Eyal, "The miner's dilemma," *2015 IEEE Symposium on Security and Privacy*, pp. 89–103, 2014.

[26] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, "Bitcoin mining pools: A cooperative game theoretic analysis," in *AAMAS*, 2015.

[27] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis, "Blockchain mining games," pp. 365–382, 07 2016.

[28] N. Courtois and L. Bahack, "On subversive miner strategies and block withholding attack in bitcoin digital currency," 01 2014.

[29] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, "Algorand agreement: Super fast and partition resilient byzantine agreement." Cryptology ePrint Archive, Report 2018/377, 2018. https://eprint.iacr.org/2018/377.

[30] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *In Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS*, pp. 120–130, IEEE, 1999.

[31] E. Abraham, "Post-quantum verifiable random functions from ring signatures." Cryptology ePrint Archive, Report 2018/1231, 2018. `https://eprint.iacr.org/2018/1231`.

[32] B. Ford, "Threshold logical clocks for asynchronous distributed coordination and consensus," *ArXiv*, vol. abs/1907.07010, 2019.

[33] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese, "Time optimal self-stabilizing synchronization," in *STOC*, 1993.

[34] S. Kadhe, J. Chung, and K. Ramchandran, "Sef: A secure fountain architecture for slashing storage costs in blockchains," *ArXiv*, vol. abs/1906.12140, 2019.

[35] M. Luby, "Lt codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pp. 271–280, 2002.