Quantum Contracts

On-chain Quantum Computing for Web3

Teik Guan Tan
pQCee Pte Ltd
teikguan@pqcee.com

***Abstract*—We introduce Quantum Contracts, a fully on-chain quantum emulator running within an Ethereum Virtual Machine (EVM) smartcontract which can be used to simulate different Quantum algorithms quickly and easily.**

I. Introduction

In this paper, we introduce a fully on-chain quantum emulator built as a smartcontract in Solidity and deployed on the Polygon Mumbai testnet. The objective is to enhance web3 developers' understanding of quantum circuits and gates, and explore how quantum algorithms can be used in web3 applications.

The 2 most-cited quantum algorithms that security cryptographers know are Shor's algorithm [1] and Grover's algorithm [2]. Shor's algorithm could solve the Interger Factorization problem and Discrete Logarithm problem in polynomial time, which meant that public key cryptography algorithms such as RSA and Elliptic-Curve Cryptography are no longer secure. Grover's algorithm provided a quadratic speed-up on an unsorted search operation which could effectively reduce the cryptographic bit strength of symmetric-key and hash cryptography by half. Both were proposed in the 1990s, and were proven theoretically before being experimentally realized some years later [3], [4], albeit with low number of qubits and under very controlled environments. Beyond code cracking, do we know of other applications of Shor's or Grover's algorithm? There are also newer quantum algorithms such as Quantum Approximate Optimization Algorithm (QAOA) [5] and Quantum Machine Learning (QML) [6] which are used to achieve optimizations in industrial applications. Can they be useful for web3 applications?

Although there has been several quantum programming languages [7] and their accompanying simulation environments including IBM's Qiskit (https://qiskit.org/) and Microsoft Q# (https://www.microsoft.com/en-us/quantum/development-kit) made available, we found that integrating these environments to a web3 chain required a fairly complex setup of connecting oracles [8] and consume both time and gas [9]. We wanted a tool that could run efficiently onchain without dependencies, without any pre-requisite knowledge of Quantum circuits or programming, yet powerful to emulate most of the Quantum algorithms (http://quantumalgorithmzoo.org/) in the field.

With Quantum Contracts, a web3 developer can quickly and interactively simulate different Quantum algorithms to facilitate in the understanding of the algorithm. Section II covers the different features and functionality of Quantum Contracts while Section III shows the usage of Quantum Contracts to demonstrate the various Quantum algorithms. We conclude in Section V.

II. Quantum Contracts Functionality

The Quantum Contracts engine is inspired by Terry Rudolph's book "Q is for Quantum" [10] which is targeted at readers without knowledge of Quantum mechanics. Central to Quantum mechanics is understanding of the qubit, represented as follows:

$$\alpha\left|0\right\rangle + \beta\left|1\right\rangle \text{ where } \alpha, \beta \in \mathbb{C}, \alpha^2 + \beta^2 = 1 \tag{1}$$

$Braket$ notation is used here to explain the functionality of circuits, and does not appear in the actual Quantum Contracts interface.

A. Quantum Gates

The gates currently available in Quantum Contracts v0.2a to work on the qubits are:

- *Pauli-X (X) Gate*. The *Pauli-X* or Not gate takes in an input qubit and inverses the qubit.

$$\alpha\left|0\right\rangle + \beta\left|1\right\rangle \xrightarrow{X} \beta\left|0\right\rangle + \alpha\left|1\right\rangle \tag{2}$$

- *Pauli-Y (Y), Pauli-Z (Z) Gate*. The *Pauli-Y* and *Pauli-Z* gates take in an input qubit and rotates the qubit around the Y-axis and Z-axis respectively.

$$\begin{aligned} \alpha\left|0\right\rangle + \beta\left|1\right\rangle &\xrightarrow{Y} -i\beta\left|0\right\rangle + i\alpha\left|1\right\rangle \\ \alpha\left|0\right\rangle + \beta\left|1\right\rangle &\xrightarrow{Z} \alpha\left|0\right\rangle - \beta\left|1\right\rangle \end{aligned} \tag{3}$$

- *Identity (I) Gate*. The Identity gate takes in an input qubit and does not change the qubit.

$$\alpha\left|0\right\rangle + \beta\left|1\right\rangle \xrightarrow{I} \alpha\left|0\right\rangle + \beta\left|1\right\rangle \tag{4}$$

- *Control^n-Not (C^nN) Gate*. The Control^n-Not gate takes in n control input qubits, and performs a Not operation on the Not input qubit if all the n control input qubits are "1". If there are 2 control inputs (CCN), then this gate is also known as the "Toffoli" gate.

$$\begin{aligned} |000\rangle &\xrightarrow{CCN} |000\rangle, \quad |100\rangle \xrightarrow{CCN} |100\rangle \\ |001\rangle &\xrightarrow{CCN} |001\rangle, \quad |101\rangle \xrightarrow{CCN} |101\rangle \\ |010\rangle &\xrightarrow{CCN} |010\rangle, \quad |110\rangle \xrightarrow{CCN} |111\rangle \\ |011\rangle &\xrightarrow{CCN} |011\rangle, \quad |111\rangle \xrightarrow{CCN} |110\rangle \end{aligned} \tag{5}$$

- *Hadamard (H)*. The Hadamard gate is the quantum gate that puts a qubit into a super-position.

$$\alpha|0\rangle + \beta|1\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(\alpha+\beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha-\beta)|1\rangle \tag{6}$$

- *Phase Shift (P) and (T) Gate*. The Phase shift P and T gates take in an input qubit and rotates the qubit around Z-axis by $\frac{\pi}{4}$ and $\frac{\pi}{8}$ respectively.

$$\begin{aligned} \alpha|0\rangle + \beta|1\rangle &\xrightarrow{P} \alpha|0\rangle + i\beta|1\rangle \\ \alpha|0\rangle + \beta|1\rangle &\xrightarrow{T} \alpha|0\rangle + e^{i\frac{\pi}{4}}\beta|1\rangle \end{aligned} \tag{7}$$

- *Measure (m) operator*. This operator forces the qubit to be measured in the computational basis during the circuit operation.

As the Hadamard, Control-Not and Phase-shift gates together make up a universal gate set [11], we are therefore confident to emulate all Quantum circuits as required.

B. Quantum Circuits

Quantum circuits are represented as a sequential set of text strings where each string is dynamically interpreted and applied on all the qubits serially.

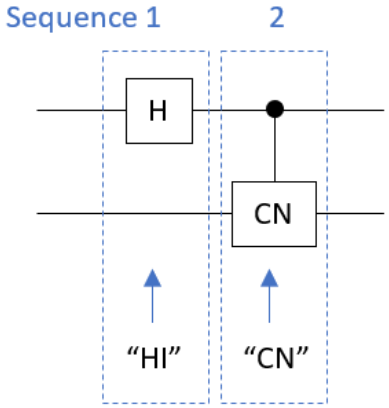

Fig. 1. Quantum circuit to create Bell state.

For example, the Quantum circuit in Fig 1 executing from left to right to create an entangled Bell state [12] would require 2 text strings, "HI" and "CN". In the first sequence string "HI", the Hadamard gate operates on the 1_{st} qubit, while there is no gate for the 2^{nd} qubit. In the second sequence string "CN", the control bit evaluates on the 1^{st} qubit and marks the 2^{nd} qubit if the "1" is detected.

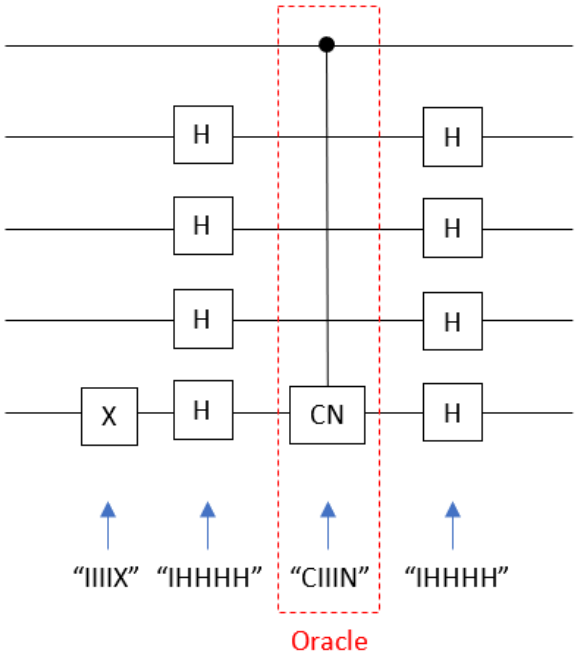

Fig. 2. Quantum circuit for constant Deutsche-Jozsa.

In another example in Fig 2 for a constant Deutsch-Jozsa [13] circuit, the text string comprise of 4 sequence strings, "IIIIX, HHHHI, CIIIN, HHHHI" where "CIIIN" represents the Oracle.

C. Quantum Circuit String Definition

We define the BNF notation for Quantum Circuit sequence string in Table I.

TABLE I
BNF definition for circuits used in Quantum Contracts

```
<Circuit>     ::= <Gate Set> | <Circuit><Delim><Gate Set>
<Gate Set>    ::= <Gate> | <Gate><Gate Set>
<Gate>        ::= H | C | N | X | Y | Z | P | T | I | m
<Delim>       ::= , | .
```

III. Using Quantum Contracts

Quantum Contracts v0.2a is currently deployed as a standalone smartcontract on the Polygon Mumbai testnet, address *0x3A2A12b422B653f9a228DfC4bCE1C27609EF36d8*. The function to be called is *runQScript()* (for Application Binary Interface (ABI), see Figure 4)

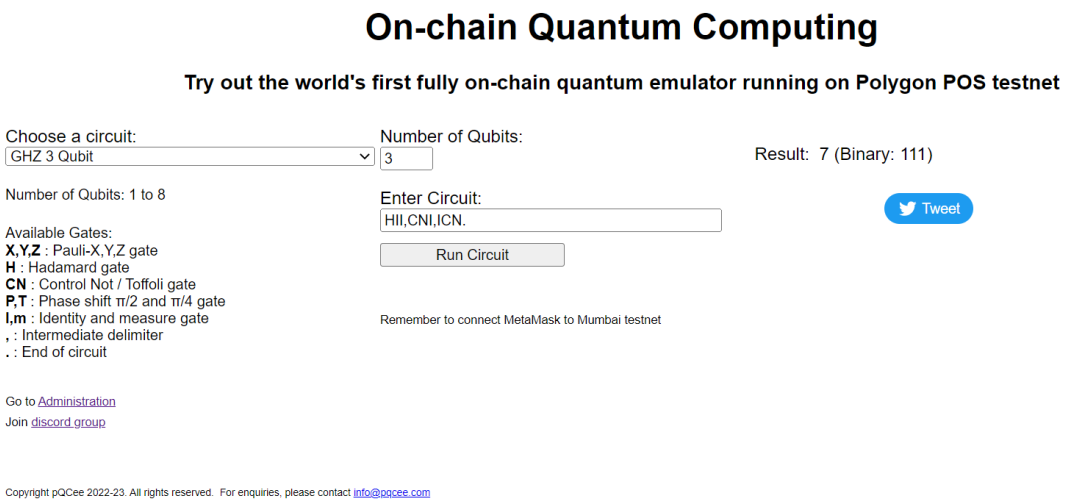

Fig. 3. Screenshot of front-end DApp.

which takes in 3 input parameters, namely Number of Qubits (ranging from 1 to 8), the Quantum circuit as defined in Section II-C, and an external integer random seed.*runQScript()* is a view function that does not require gas to operate if called from a user DApp. A sample front-end DApp (see Figure 3) for Quantum Contracts *runQScript()* function can be found at https://tanteikg.github.io/QC.

We demonstrate the use of Quantum Contracts to run some of the popularly known algorithms and show the results of execution in Table II.

IV. Next Steps

We are still working to improve the accuracy and computational capability of Quantum Contracts to be able to express more algorithms easily. Some of the planned constructs include conditional loops and Oracle functionality. We will also embarking on the next phase of the project to identify use-cases of quantum algorithms for web3 applications.

V. Conclusion

Quantum Contracts is a fully onchain 8-qubit quantum emulator running in a smartcontract. The current version is open-source and can be found at https://github.com/tanteikg/QuantumContract. We intend to implement a freemium-subscription model in the medium term.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[2] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

```
[
  {
    "inputs": [
      {
        "internalType": "uint8",
        "name": "numQubits",
        "type": "uint8"
      },
      {
        "internalType": "string",
        "name": "s",
        "type": "string"
      },
      {
        "internalType": "uint256",
        "name": "randomSeed",
        "type": "uint256"
      }
    ],
    "name": "runQScript",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  }
]
```

Fig. 4. ABI of *runQScript()*.

TABLE II
Example circuits of various Quantum algorithms to run on Quantum Contracts

Algorithm	Description	Quantum Circuit	Quantum Contracts Output
Bell [12]	Basic 2 qubit entanglement	`HI,CN.`	`Result: 0 or 3`
GHZ [14]	3 qubit entanglement	`HII,CNI,ICN.`	`Result: 0 or 7`
Simon's [15]	Exponential speedup to find "11"	`HHII,CINI,CIIN,ICNI,ICIN,IImm, HHII,mmII.`	`Result: 0, 3, 12 or 15`
2-bit Grover [2]	Grover Amplitude Amplification searching for string "11"	`HHI,IIX,IIH,XXI,CCN,XXI,IIH, IIX,HHI,XXI,IHI,CNI,IHI,XXI,HHI.`	`Result: 6`
3-bit Grover [2]	Grover Amplitude Amplification searching for string "111"	`HHHI,IIIX,IIIH,IIII,CCCN,IIII, IIIH,IIIX,HHHI,XXXI,IIHI,CCNI, IIHI,XXXI,HHHI,IIIX,IIIH,IIII, CCCN,IIII,IIIH,IIIX,HHHI,XXXI, IIHI,CCNI,IIHI,XXXI,HHHI.`	`Result: Mostly 14`
Shor's [1]	Quantum Fourier Transform to factorize 21	`HHHII,IICIN,ICIIN,IIINC,ICICN, IIIIX,CIINC,IIIIX,IIINC,CIICN, IIINC,IIHII,ICPII,CITII,IHIII, CPIII,HIIII.`	`Please refer to [16]`
Shor's [1]	Quantum Fourier Transform to factorize 15	`HHHIIII,IICINII,IICIINI,IIICINI, ICINICI,IIICINI,IIIINIC,ICIICIN, HIIIIII,CPIIIII,IHIIIII,CITIIII, ICPIIII,IIHIIII.`	`Please refer to [3]`

[3] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[4] V. L. Ermakov and B. Fung, "Experimental realization of a continuous version of the grover algorithm," *Physical Review A*, vol. 66, no. 4, p. 042310, 2002.

[5] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[7] D. A. Sofge, "A survey of quantum programming languages: History, methods, and tools," in *Second International Conference on Quantum, Nano and Micro Technologies (ICQNM 2008)*, pp. 66–71, IEEE, 2008.

[8] QCentoid, "Connecting Quantum computing to Web3. A tech approach.," 2023. Available Online: https://qcentroid.xyz/blog/connecting-quantum-computing-to-web3-a-tech-approach/.

[9] API3Foundation, "API3 QRNG — Web3 Quantum Random Numbers.," 2022. Available Online: https://medium.com/api3/api3-qrng-web3-quantum-random-numbers-4ca7517fc5bc.

[10] T. Rudolph, *Q is for Quantum*. Terry Rudolph, 2017.

[11] D. Aharonov, "A simple proof that toffoli and hadamard are quantum universal," 2003.

[12] S. L. Braunstein, A. Mann, and M. Revzen, "Maximal violation of bell inequalities for mixed states," *Physical Review Letters*, vol. 68, no. 22, p. 3259, 1992.

[13] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[14] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond bell's theorem," in *Bell's theorem, quantum theory and conceptions of the universe*, pp. 69–72, Springer, 1989.

[15] D. R. Simon, "On the power of quantum computation," *SIAM journal on computing*, vol. 26, no. 5, pp. 1474–1483, 1997.

[16] U. Skosana and M. Tame, "Demonstration of shor's factoring algorithm for n= 21 on ibm quantum processors," *Scientific reports*, vol. 11, no. 1, p. 16599, 2021.