Experimental Cryptographic Verification for Near-Term Quantum Cloud Computing

Xi Chen,[1, 2, 3, *] Bin Cheng,[4, *] Zhaokai Li,[1, 2, 3] Xinfang Nie,[1, 2, 3] Nengkun Yu,[5, †] Man-Hong Yung,[4, 6, 7, ‡] and Xinhua Peng[1, 2, 3, §]

[1]*Hefei National Laboratory for Physical Sciences at the Microscale and Department of Modern Physics, University of Science and Technology of China (USTC), Hefei 230026, China*
[2]*CAS Key Laboratory of Microscale Magnetic Resonance, USTC, Hefei 230026, China*
[3]*Synergetic Innovation Center of Quantum Information and Quantum Physics, USTC, Hefei 230026, China*
[4]*Institute for Quantum Science and Engineering, and Department of Physics, Southern University of Science and Technology, Shenzhen 518055, China*
[5]*Centre for Quantum Software and Information, School of Software, Faculty of Engineering and Information Technology, University of Technology Sydney, NSW, Australia*
[6]*Shenzhen Key Laboratory of Quantum Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China*
[7]*Central Research Institute, Huawei Technologies, Shenzhen, 518129, P. R. China*

Recently, there are more and more organizations offering quantum-cloud services, where any client can access a quantum computer remotely through the internet. In the near future, these cloud servers may claim to offer quantum computing power out of reach of classical devices. An important task is to make sure that there is a real quantum computer running, instead of a simulation by a classical device. Here we explore the applicability of a cryptographic verification scheme that avoids the need of implementing a full quantum algorithm or requiring the clients to communicate with quantum resources. In this scheme, the client encodes a secret string in a scrambled IQP (instantaneous quantum polynomial) circuit sent to the quantum cloud in the form of classical message, and verify the computation by checking the probability bias of a class of output strings generated by the server. We provided a theoretical extension and implemented the scheme on a 5-qubit NMR quantum processor in the laboratory and a 5-qubit and 16-qubit processors of the IBM quantum cloud. We found that the experimental results of the NMR processor can be verified by the scheme with about 2.5% error, after noise compensation by standard techniques. However, the fidelity of the IBM quantum cloud is currently too low to pass the test (about 42% error). This verification scheme shall become practical when servers claim to offer quantum-computing resources that can achieve quantum supremacy.

Introduction.— Quantum computation promises a regime with unprecedented computational power over classical devices, offering numerous interesting applications, such as factorization [1], quantum simulation [2, 3], and quantum machine learning [4, 5]. However, before quantum computers become prevalent to the public, one might expect that only organizations with sufficient resources could operate a full-scale quantum computer, analogous to today's supercomputers. Furthermore, individuals who have demands for quantum computation could access the service through the internet, i.e., cloud quantum computing. In fact, several small-scale quantum cloud services have already been launched [6–8], which can be operated by remote clients through the internet. As a result, many simulations performed from quantum cloud servers have been reported (see Ref. [9] for a summary).

In the near future, it is not impossible that these clouds may claim to offer 100 or more working qubits and many layers of quantum gates, where quantum supremacy [10–13] could be achieved. However, one may naturally ask, is there a real quantum computer behind the cloud? Or, would it just be a classical computer simulating quantum computation? For ordinary clients who only have control and access of classical computer, a natural task is to verify whether these cloud servers are truly quantum.

Alternatively, the question can be formalized as follows: *is it possible for a purely-classical client to verify the output of a quantum prover?* This question has been extensively explored for more than ten years. In 2004, Gottesman initialized this question, which Aaronson wrote down in his blog [14]. A straight-forward idea is to run a quantum algorithm solving certain NP problems, for example, Shor's algorithm for integer factorization [1]. Such problems might be hard for classical computation, but are easy for *classical verification* once the result is known. However, the challenge is that a full quantum algorithm typically requires thousands of qubits and quantum error correction to be implemented, which is out of question in the NISQ [15] (Noisy Intermediate-Scale Quantum) era.

Note that the verification problem have different variants. For example, one may assume that the supposedly "classical" client may actually have a limited ability to perform quantum operations on a small number of qubits. This line of research has already attracted much attention [16–22]. Without any quantum power, the client might still be able to verify delegated quantum computation which is spatially separated and entanglement can be shared [23, 24]. Currently, this approach does not seem to fit the setting of the available quantum cloud services, but it does reveal the outstanding challenge for establishing a rigorous verification scheme based on a classical client interacting with a single server using only classical communication [25].

Until recently, Mahadev has made important progresses [26, 27], assuming that the learning-with-errors problem [28] is computationally hard even for quantum computer. The protocol allows a classical computer to *interactively* verify the results of an efficient quantum computation, achieving a fully-homomorphic encryption scheme for quantum circuits with classical keys. Despite these great efforts, we are still facing the problems of "non-interactively" verifying *near-term* quantum clouds, which would be too noisy for implementing full quantum algorithms but may be capable of demonstrating quantum supremacy.

Here we report an experimental demonstration of a simple but powerful cryptographic verification protocol, originally proposed by Bremner and Shepherd [29] in 2008. We extended the theoretical construction in terms of n-point correlation. The implementation was first performed with a 5-qubit NMR quantum processor in the laboratory. Additionally, we also benchmarked the performance of the verification scheme by actually implementing the protocol with the IBM quantum cloud processors [6].

The verification protocol implemented is based on a simplified circuit model of quantum computation, called IQP (instantaneous quantum polynomial) model [29]; the qubits are always initialized in the '0' state. The IQP circuits contain three parts. In the first and the last part, single-qubit Hadamard gates are applied to every qubit. The middle part of an IQP circuit does not contain an explicit temporal structure, in the sense that diagonal (and hence commuting) gates acting on single or multiple qubits are applied. On one hand, the IQP model represents a relatively resource-friendly computational model to be tested with near-term quantum devices. On the other hand, the IQP model has been proven to be hard for classical simulation [30, 31], under certain computational assumptions, similar to Boson sampling [32].

Verification protocol.— In the cryptographic verification protocol [29], there are two parties labeled as Alice (the client) and Bob (the server). Alice is assumed to be *completely classical*; she can only communicate with others through classical communication (e.g., internet). Suppose Bob claims to own a quantum computer and Alice is going to test it. In reality, of course, there is no need for Alice to inform Bob about her intention; she may just pretend to run a normal quantum program. The protocol can be succinctly summarized as follows (depicted by Fig. 1).

Step 1: Alice first generates a matrix (called X-program [29]) associated with a secret string $s \in \{0,1\}^n$, which is only kept by Alice.

Step 2: Alice then translates the X-program into an IQP circuit of n qubits, and sends the information about the IQP circuit $U_{\rm IQP}$ to Bob.

Step 3: Bob returns the outputs to Alice in terms of the bit strings $x \in \{0,1\}^n$, which should follow the distribution of the IQP circuit, i.e., $\Pr(x) = |\langle x| U_{\rm IQP} |0^n\rangle|^2$, if Bob is honest.

Step 4: Ideally, Alice should be able to determine if the probability distributions $\Pr(x)$ for a subset of strings orthogonal to the secret string, where $x \cdot s \equiv x_1 s_1 + x_2 s_2 + \cdots + x_n s_n = 0 \mod 2$, add up to an expected value 0.854. Otherwise, Bob fails to pass the test.

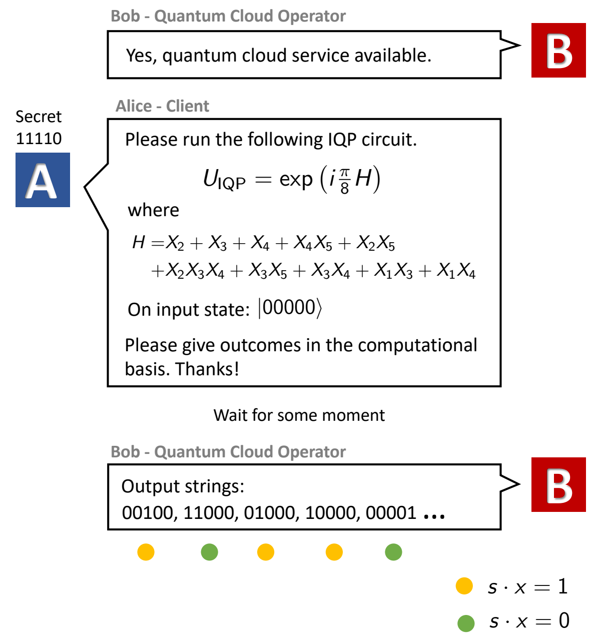

Bob - Quantum Cloud Operator
Yes, quantum cloud service available.
B
Secret 11110
A
Alice - Client
Please run the following IQP circuit.
$U_{\rm IQP} = \exp\left(i\frac{\pi}{8}H\right)$
where
$H = X_2 + X_3 + X_4 + X_4X_5 + X_2X_5 + X_2X_3X_4 + X_3X_5 + X_3X_4 + X_1X_3 + X_1X_4$
On input state: $|00000\rangle$
Please give outcomes in the computational basis. Thanks!
Wait for some moment
Bob - Quantum Cloud Operator
Output strings: 00100, 11000, 01000, 10000, 00001 ...
B
$s \cdot x = 1$
$s \cdot x = 0$

FIG. 1. Schematic representation of the protocol. Alice generates a description matrix, according to the construction of quadratic residue code. This description matrix has an associating secret vector s and determines an X-program circuit. Bob runs this circuit, measures and sends back the measurement data to Alice. From Bob's data, Alice computes the probability bias $\mathcal{P}_{s\perp}$, with respect to the secret vector s, and sees whether it is close to 0.854, to decide whether Bob has a true quantum device or not.

More specifically, the key quantity of interest is the following probability bias defined by,

$$\mathcal{P}_{s\perp} \equiv \sum_{x\in\{0,1\}^n} |\langle x| U_{\rm IQP} |0^n\rangle|^2 \, \delta_{x\cdot s=0} \, , \tag{1}$$

where $\delta_{x\cdot s=0} = 1$ if it is true that $x \cdot s = 0$, and $\delta_{x\cdot s=0} = 0$ otherwise. For a perfect quantum computation, the value of the probability bias should be $\mathcal{P}_{s\perp} = 0.854$. The best known classical algorithm [29] would instead produce a value of 0.75, which is relevant when n is sufficiently large. This quantum-classical gap in the probability bias makes it possible to apply such a resource-friendly cryptographic verification scheme for testing quantum cloud computing in the regime where quantum supremacy would be achieved.

Overview.— To illustrate our experimental demonstrations, we shall first provide a concise and self-contained theoretical description of the cryptographic protocol. Particularly, our computer code implemented in the experiment is open source and available online [33] and in Supplemental Materials; interested readers can readily reproduce our results with it, and can also apply it to generate X-programs of different variations for testing other quantum-cloud services.

On the other hand, we also provide a theoretical extension of the original work [29], transforming it into a form more familiar to the physics community. Specifically, we connect the probability bias in Eq. (1) with the Fourier coefficient of

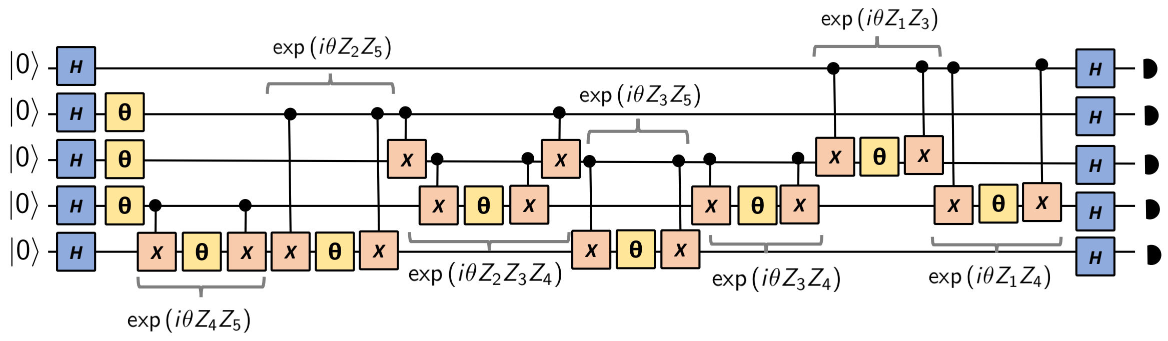


FIG. 2. IQP circuit for the X-program matrix (3). There are two layers of Hadamard gates at the beginning and the end of the circuit. The gates in between are all diagonal in the Z basis. θ stands for $\exp(i\theta Z)$.

the probability $\Pr(x)$ of the output strings. As a result, we can express the probability bias through the n-point correlation function (see Supplemental Materials):

$$\mathcal{P}_{s\perp} = \frac{1}{2}\left(1 + \langle Z^{s_1} Z^{s_2} \cdots Z^{s_n}\rangle\right) . \qquad (2)$$

Since the string $s = s_1 s_2 \cdots s_n$ is not known to Bob, the verification protocol can be regarded as a game where Alice tests the outcomes in terms of a particular correlation function unknown to Bob.

In addition, as we will see later, the X-program consists of two part, the main part and the redundant parts, and the representation of Eq. (2) provides a straight-forward way to understand why the redundant part of the X-program does not affect the probability bias—they commute with the n-point correlation function.

Our theoretical extension in Eq. (2) allows us to take into account the effect of noises. More precisely, if one models [34, 35] the decoherence by a dephasing channel (with an error rate ϵ) applied for each qubit at each time step, then the probability bias becomes $\mathcal{P}_{s\perp} \to \frac{1}{2}\left(1 + (1-2\epsilon)^{|s|}\langle Z^{s_1} Z^{s_2} \cdots Z^{s_n}\rangle\right)$, where $|s|$ is the Hamming weight of s, that is the number of 1's in s.

Experimentally, our data were taken separately from two different sources, namely a five-qubit NMR processor in the laboratory, and the IBM cloud services, aiming to benchmark the performances of the IQP circuit implementation under the laboratory conditions and that from the quantum-cloud service.

Our results show that the laboratory NMR quantum processor can be employed to verify the IQP circuit after noise compensation by standard techniques, but the IBM quantum cloud was too noisy. The probability bias obtained from the IBM's processors are close to 0.5, which is the result of uniform distribution. The main reason is that IBM's system has many constraints on the connectivity between the physical qubits; we had to include many extra SWAP gates to complete the circuit, causing a severe decoherence problem.

Theoretical construction.— Here Alice's secret vector of string is given by $s = (1, 1, 1, 1, 0)$. An X-program can be represented by a matrix with binary values, which is constructed from the quadratic residue code (QRC) [36]. In the experiment, the matrix $\mathcal{X}$ associated with the X-program is given by the following (see Supplemental Material for the construction method),

$$\mathcal{X} = \left(\begin{array}{ccccccc|ccc} 0&0&0&0&0&0&0&0&1&1 \\ 1&0&0&0&1&1&0&0&0&0 \\ 0&1&0&0&0&1&1&1&1&0 \\ 0&0&1&1&0&1&0&1&0&1 \\ 0&0&0&1&1&0&1&0&0&0 \\ \multicolumn{7}{c|}{\underbrace{\hphantom{0\;0\;0\;0\;0\;0\;0}}_{\text{Main}}} & \multicolumn{3}{c}{\underbrace{\hphantom{0\;0\;0}}_{\text{Redundant}}} \end{array}\right) . \qquad (3)$$

Here the X-program has a layer of security for protecting the knowledge of the secret string s from Bob. Explicitly, there are two parts in the matrix, (i) the main part and (ii) a redundant part. Columns in the main part are not orthogonal to the secret vector s, i.e., $x \cdot s = 1$, while columns in the redundant part are, i.e., $x \cdot s = 0$. Both parts have to be changed if the secret string is changed.

However, an important property of the X-program is that the probability bias $\mathcal{P}_{s\perp}$ depends only on the main part. So Alice can append as many redundant columns that are orthogonal to s to this matrix as she wishes. Of course, later she would need to scramble the columns, in order to hide the secret s from Bob.

Next, the X-program has to be translated into an IQP circuit [29], which is a subclass of quantum circuits with commuting gates before and after the Hadamard gates. Equivalently, the unitary transformation associated with the IQP circuit can be casted as follows:

$$U_{\text{IQP}} = \exp(i\theta H) , \qquad (4)$$

where $\theta = \pi/8$, and H is the effective Hamiltonian constructed by the elements of the X-program. For example, a column $(1, 0, 1, 1, 0)$ represents a term $X_1X_3X_4$, where X_i is a Pauli-X acting on the i-th qubit. As a result, the full Hamiltonian corresponding to $\mathcal{X}$ reads,

$$\begin{aligned} H &= X_2 + X_3 + X_4 + X_4X_5 + X_2X_5 \\ &\quad + X_2X_3X_4 + X_3X_5 + X_3X_4 + X_1X_3 + X_1X_4 . \end{aligned} \qquad (5)$$

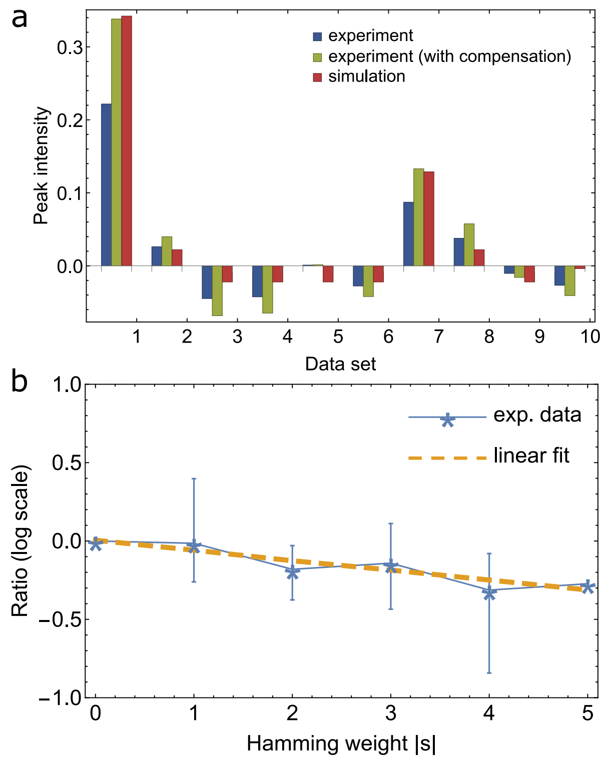


FIG. 3. **(a)** Part of the peak intensities from the readout pulses. There are 80 groups of peak intensities in total and we only present the first 10 groups here. Each peak intensity is a linear combination of probabilities. **(b)** The ratio of the experimental value to the theoretical value of the n-point correlation function $\langle Z^{s_1} Z^{s_2} \cdots Z^{s_n} \rangle$ in log scale, versus Hamming weights.

Note that if we take $\theta = \pi/4$, then the evolution can be simulated classically by the Gottesman-Knill algorithm [37]. Fig. 2 shows the circuit diagram.

In the ideal case, the probability bias for the IQP circuit should be given by $\mathcal{P}_{s\perp} = 0.854$. If Bob outputs random bits, the value of the probability bias would be $\mathcal{P}_{s\perp} = 0.5$. However, although it is scrambled, the X-program is correlated with the secret string. The classical algorithm provided in Ref. [29] can yield $\mathcal{P}_{s\perp} = 0.75$, which was conjectured to be optimal [29]. As a result, it becomes possible to verify the quantum hardware behind the quantum clouds by simply collecting the statistics of the outputs to check if we can get $\mathcal{P}_{s\perp} = 0.854$.

For the purpose of benchmarking, we performed a total of three separate implementations of the same X-program on an NMR quantum processor and on IBM quantum processors, including the 5-qubit one and the 16-qubit one [6].

Verification with NMR in the laboratory.— The experiments with the NMR quantum processor were carried on a Bruker AV-400 spectrometer at 303K. The 5-qubit quantum processor consists of two ^1H nuclear spins and three ^{19}F nuclear spins in 1-bromo-2,4,5-trifluorobenzene dissolved in the liquid crystal N-(4-methoxybenzylidene)-4-butylaniline (MBBA) [38]. The molecular structure and equilibrium spectra of ^{19}F nuclear spins are provided in the Supplemental Materials.

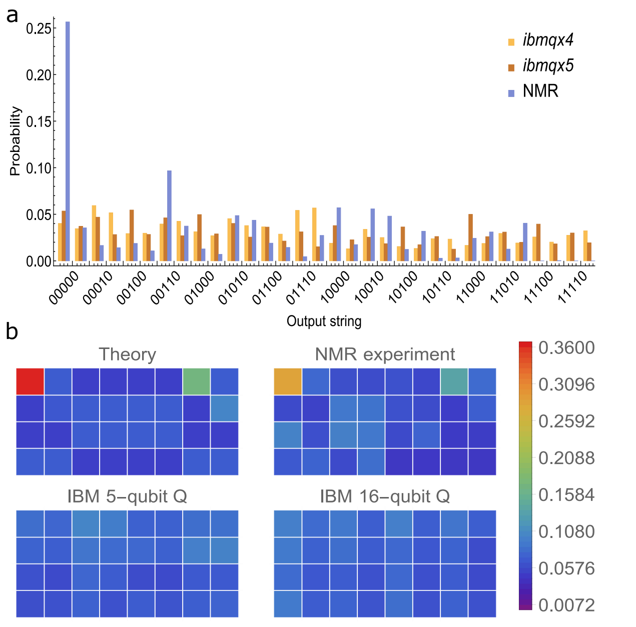


FIG. 4. **(a)** Probability distributions from IBM quantum processors and the NMR processor. *ibmqx4* is the 5-qubit processor and *ibmqx5* is the 16-qubit one. **(b)** The probabilities are put into grids and the colors indicate their values according to the color scale on the right.

Starting from the thermal state ρ_{eq}, the NMR system is initially prepared in a PPS $\rho_i = \frac{1-\varepsilon}{32}\mathbf{I}_{32} + \varepsilon\,|00000\rangle\langle 00000|$ by the line-selective method [39]. Here $\mathbf{I}_{32}$ represents the 32×32 identity operator and $\varepsilon \approx 10^{-5}$ is the polarization. Note that the identity operator is invariant under the unitary transformation, neither does it affect the measurement step. The state ρ_i evolves the same way as a true pure state $|00000\rangle\langle 00000|$ and generates the same signal up to a proportionality factor ε, so we can simply regard the ρ_i as $|00000\rangle\langle 00000|$.

To implement the IQP circuit, i.e. the U_{IQP} described by Eq. (4), we packed it into one shaped pulse optimized by the gradient ascent pulse engineering (GRAPE) method [40], with the length being 37.5 ms and the number of segments being 1500. The shaped pulse has the theoretical fidelity of 99.4% and is designed to be robust against the inhomogeneity of the pulse amplitude.

To obtain the probability bias in the experiment, we need five readout pulses (i.e. a Y pulse on each qubit $\exp\left(-i\pi Y_j/4\right)$) to reconstruct the diagonal elements of the density matrix of the final state [41], which are the probabilities in the computational basis. For details, we refer to the Supplemental Materials.

From each readout pulse, we can obtain 16 peak intensities, and each peak intensity is a linear combination of the 32 probabilities. So we have 80 linear equations of the form: $\sum_x c_x(l)\,\text{Pr}(x) = \alpha_l$ $(1 \leq l \leq 80)$, where α_l's are the peak intensities read out by our device. We present 10 of these peak intensities in Fig. 3 (a), and figure with all peak intensities can be found in Supplemental Materials. The blue lines in

Fig. 3 (a) are from the experiment on our NMR processor, and the red ones are from theoretical simulation without considering the noise effect. After solving those 80 linear equations from NMR processor together with the normalization condition $\sum_x \Pr(x) = 1$ through the least square method, we obtain the corresponding probability distribution, as depicted in Fig. 4 (a) (the blue histogram). The probability bias from this raw distribution is 0.755.

The probability bias is connected to the n-point correlation function $\langle Z^{s_1} Z^{s_2} \cdots Z^{s_n} \rangle$ through Eq. (2). If there is single-qubit dephasing noise on every qubits at each step, the n-point correlation function will decay by a factor $(1-2\epsilon)^{|s|}$ [34, 35]. Thus the ratio of the experimental n-point correlation (which is from the raw distribution) to the theoretical value is $(1-2\epsilon)^{|s|}$. Fig. (3) (b) shows this ratio in log scale, versus Hamming weights of all possible s. The slope of the linear fit is $\log(1-2\epsilon)$, from which we obtain an effective noise rate $\epsilon = 6.79\%$.

We note that the whole duration of the dynamic evolution is 37.5 ms whereas the decoherence time is about 50 ms. Hence, the decay caused by the decoherence is not negligible. To compensate the effects of decoherence, we experimentally estimate the attenuation factor, that is, the ratio of peak intensities with decoherence to peak intensities without decoherence. Concretely, we design a shaped pulse of the 32×32 identity operator with length 37.5 ms and use the decay in the peak intensities of this identity evolution to estimate the attenuation factor. To compensate the effects of decoherence, peak intensities from actual experiment are divided by the attenuation factor (see Supplemental Material for details) and the resulted peak intensities are shown as the green lines in Fig. 3. Then with a similar method by solving linear equations, we derive the probability bias compensated by noise: 0.866 ± 0.016 (comparable with the theoretical value: 0.854). Details of the error bar 0.016 can be found in Supplemental Material.

Verification with IBM cloud.— As for experiments on IBM devices, we run the same circuits as in Fig. 2. However, due to the connectivity constraints [6], we have to include several additional SWAPs to complete the circuit, which makes the whole circuit depth be about $40 \sim 50$; for example, CNOT gates can only be applied to a certain pairs of qubits. In our demonstration, we applied the verification algorithm for *ibmqx4*, a 5-qubit superconducting processor, and *ibmqx5*, a 16-qubit one, which are accessed via a software called QISKit. We collect specification of IBM devices in Supplemental Material, including the connectivity and coherence time. These data can also be found in Ref. [6].

In Fig. 4 (a), the histogram in orange is the probability distribution from the experiment on *ibmqx4*, while the brown one is from *ibmqx5*. The probability biases from these two distributions are respectively 0.488 and 0.492, which are far from the expected value of 0.854. In fact, from a completely-mixed state, we can obtain a probability bias of 0.5. So the values of bias from these two quantum cloud services by IBM indicate that their final states are highly corrupted by decoherence. Furthermore, to see whether the IBM cloud would have a better performance if we reduce the depth, we implement a quantum circuit only corresponding to the main part of matrix (3) on *ibmqx4*. However, the obtained bias is 0.512, which is still very close to that from a completely mixed state (see Supplemental Material for details). Therefore, we concluded that the IBM cloud was too noisy to pass our test.

Fig. 4 (b) shows the comparison of the distributions. Each grid has 32 elements, corresponding to 32 probabilities. The color in each element indicates the concrete value, according to the color scale on the right. The distribution from the experiment run on NMR device is close to the theoretical prediction while the last two, which are distributions from IBM devices, are not.

Summary.— In conclusion, we have performed a proof-of-principle demonstration of a cryptographic verification scheme, using an NMR quantum processor and the IBM quantum cloud. The experimental results show that the fidelity of the quantum cloud service has to be significantly improved, in order to be testable with the verification method. In particular, the connectivity between the qubits imposes an extra overhead in the implementation of the scheme. For a large-scale implementation, it is the also important to determine numerically the size of IQP circuit that can no longer be simulable by classical computers, which is currently an open question.

Acknowledgement.— We acknowledge use of the IBM Q for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team. XP, XC, ZL and XN are supported by the National Key Research and Development Program of China (Grant No. 2018YFA0306600), the National Science Fund for Distinguished Young Scholars (Grant No. 11425523), the National Natural Science Foundation of China (Grants No. 11575173), Projects of International Cooperation and Exchanges NSFC (Grant No. 11661161018), and Anhui Initiative in Quantum Information Technologies (Grant No. AHY050000). NY is supported in part by the Australian Research Council (Grant No. DE180100156). MHY is supported by the National Natural Science Foundation of China (11875160), the Guangdong Innovative and Entrepreneurial Research Team Program (2016ZT06D348), Natural Science Foundation of Guangdong Province (2017B030308003), and Science, Technology and Innovation Commission of Shenzhen Municipality (ZDSYS20170303165926217, JCYJ20170412152620376, JCYJ20170817105046702).

[*] These two authors contributed equally
[†] nengkunyu@gmail.com
[‡] yung@sustech.edu.cn
[§] xhpeng@ustc.edu.cn

[1] P. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE Comput. Soc. Press).
[2] R. P. Feynman, Int. J. Theor. Phys. **21**, 467 (1982).
[3] S. Lloyd, Science **273**, 1073 (1996).

[4] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).

[5] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," (2013), arXiv:1307.0411.

[6] IBM QX team, "IBM Quantum Experience," https://github.com/QISKit/qiskit-backend-information, *ibmqx4* V1.1.0 accessed Dec 2017; *ibmqx5* V1.1.0 accessed May 2018.

[7] Rigetti Computing, devices specification http://docs.rigetti.com/en/latest/qpu.html.

[8] "Quantum in the Cloud," University of Bristol, http://www.bristol.ac.uk/physics/research/quantum/engagement/qcloud/.

[9] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Lokhov, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'Malley, D. Oyen, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. Swart, M. Vuffray, J. Wendelberger, B. Yoon, R. Zamora, and W. Zhu, (2018), arXiv:1804.03719.

[10] J. Preskill, "Quantum computing and the entanglement frontier," (2012), arXiv:1203.5813.

[11] A. P. Lund, M. J. Bremner, and T. C. Ralph, npj Quantum Inf. **3**, 15 (2017).

[12] B. M. Terhal, Nat. Phys. **14**, 530 (2018).

[13] M.-H. Yung, Natl. Sci. Rev. , nwy072 (2018).

[14] At first sight, this seems a simple question. One may ask the quantum cloud to run a classical intractable task which is feasible for a quantum computer. This idea is not practical as it is equivalent to separating BQP (bounded-error quantum polynomial time) and P (polynomial time), one of the most important open problem in quantum complexity theory. See https://www.scottaaronson.com/blog/?p=284 for more detail..

[15] J. Preskill, "Quantum Computing in the NISQ era and beyond," (2018), arXiv:1801.00862.

[16] A. Broadbent, J. Fitzsimons, and E. Kashefi, in *2009 50th Annual IEEE Symposium on Foundations of Computer Science* (2009) pp. 517–526.

[17] A. Broadbent, J. Fitzsimons, and E. Kashefi, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6154 LNCS**, 43 (2010).

[18] D. Aharonov, M. Ben-Or, and E. Eban, (2008), arXiv:0810.5375.

[19] D. Aharonov, M. Ben-Or, E. Eban, and U. Mahadev, (2008), arXiv:1704.04487.

[20] J. F. Fitzsimons and E. Kashefi, Phys. Rev. A **96**, 012303 (2017).

[21] J. F. Fitzsimons, M. Hajdušek, and T. Morimae, Phys. Rev. Lett. **120**, 040501 (2018).

[22] D. Mills, A. Pappa, T. Kapourniotis, and E. Kashefi, Electronic Proceedings in Theoretical Computer Science **266**, 209 (2018).

[23] B. W. Reichardt, F. Unger, and U. Vazirani, Nature **496**, 456 (2013).

[24] H.-L. Huang, Q. Zhao, X. Ma, C. Liu, Z.-E. Su, X.-L. Wang, L. Li, N.-L. Liu, B. C. Sanders, C.-Y. Lu, and J.-W. Pan, Phys. Rev. Lett. **119**, 050503 (2017).

[25] S. Aaronson, A. Cojocaru, A. Gheorghiu, and E. Kashefi, (2017), arXiv:1704.08482.

[26] U. Mahadev, (2017), arXiv:1708.02130.

[27] U. Mahadev, (2018), arXiv:1804.01082.

[28] O. Regev, in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05 (ACM, 2005) pp. 84–93.

[29] D. Shepherd and M. J. Bremner, Proc. R. Soc. A **465**, 1413 (2009).

[30] M. J. Bremner, R. Jozsa, and D. J. Shepherd, Proc. R. Soc. A **467**, 459 (2011).

[31] M. J. Bremner, A. Montanaro, and D. J. Shepherd, Phys. Rev. Lett. **117**, 080501 (2016).

[32] S. Aaronson and A. Arkhipov, Theory Comput. **9**, 143 (2013).

[33] Codes are available on https://gitlab.com/BinCheng/IQP_Experiment.

[34] M. J. Bremner, A. Montanaro, and D. J. Shepherd, Quantum **1**, 8 (2017).

[35] M.-H. Yung and X. Gao, (2017), arXiv:1706.08913.

[36] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes* (Elsevier, 1977).

[37] D. Gottesman, (1998), arXiv:quant-ph/9807006.

[38] R. Shankar, S. S. Hegde, and T. Mahesh, Phys. Lett. A **378**, 10 (2014).

[39] X. Peng, X. Zhu, X. Fang, M. Feng, K. Gao, X. Yang, and M. Liu, Chem. Phys. Lett. **340**, 509 (2001).

[40] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, J. Magn. Reson. **172**, 296 (2005).

[41] J.-S. Lee, Phys. Lett. A **305**, 349 (2002).

SUPPLEMENTAL MATERIAL

Quadratic-Residue-Code Construction

We briefly review the quadratic-residue-code (QRC) construction. For detailed proofs, we refer readers to Ref. [29] and Ref. [36]. Below, all linear algebraic objects (vectors, matrices, linear spaces, etc.) are over $\mathbb{F}_2$.

Related Classical Coding Theories

Definition 1. *(code and codeword)* A *code* C is a linear subspace of $\mathbb{F}_2^n$, and the elements of a code is called *codeword*. Denote "c is a codeword of C" as $c \in C$.

Given a matrix $\mathcal{X}$, the linear combination of its rows generate a code C, and such a matrix is called *generating matrix* of C. Generating matrices for a code are not unique.

Definition 2. *(quadratic residue)* An integer j is called a quadratic residue modulo q if there exists an integer x, *s.t.* $x^2 \equiv j \pmod{q}$.

Suppose q is a prime such that 8 divides $q+1$. Consider a binary vector of length q, the j-th components of which is 1 if and only if j is a quadratic residue modulo q. The smallest example is $q = 7$, in which case such vector is $(1, 1, 0, 1, 0, 0, 0)$. Rotate it, and we get a class of vectors, such as $(0, 1, 1, 0, 1, 0, 0)$, $(0, 0, 1, 1, 0, 1, 0)$, etc. This class of vectors generates a linear subspace C_{QRC}, called *quadratic residue code* (QRC). For $q = 7$, the matrix below can be a generating matrix:

$$\mathcal{X}_M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} \tag{S1}$$

Here, the last 4 rows form a basis, and the first row is just a linear combination of the basis, thereby leaving the code invariant. More explicitly, any vectors in the code space generated by $\mathcal{X}_M$ can be written as $\sum_{i=1}^{4} c_i r_i$, where $c_i \in \{0, 1\}$.

Definition 3. *(Hamming weight)* The *Hamming weight* of a codeword is the number of 1's that it has, denoted as $|c|$.

If the Hamming wieght is even, then we say the codeword is *even*.

Definition 4. *(doubly even)* A code is *doubly even* if all the codewords have Hamming weight a multiple of 4.

Definition 5. *(dual code and self-dual)* For a code $C \subseteq \mathbb{F}_2^n$, the *dual code* $C^\perp$ is defined as $C^\perp = \{x \in \mathbb{F}_2^n \mid x \cdot c = 0 \text{ for all } c \sim C\}$. A code is *self-dual* if its dual code is itself.

Here, the inner product is over $\mathbb{F}_2$. It is clear that $\dim C + \dim C^\perp = n$. The dimension of a self-dual code is $n/2$ with n even. We collect some results from classical coding theory in Lemma 1.

Lemma 1. *The dimension of QRC with respect to q is $(q + 1)/2$. Append a single bit to every codeword of QRC (these bits are not necessarily the same for all codewords), to make them all even. The extended QRC is self-dual and doubly even.*

The Quantum Value of Bias

We append columns whose 1-st component is 0 to $\mathcal{X}_M$, to make a larger matrix $\mathcal{X}$. Obviously, columns in $\mathcal{X}_M$ are not orthogonal to $s = (1, 0, 0, 0, 0)$, but the appending columns are. We interpret $\mathcal{X}$ as an X-program, run it and calculate the bias in the direction of s, then we have the following theorem [29]:

Theorem 1. *Denote the code generated by $\mathcal{X}_M$ as C_{QRC}. Then the bias in the direction of s is*

$$\mathcal{P}_{s\perp} = \frac{1}{2^d} \sum_{c \in C_{\text{QRC}}} \cos^2[\theta(q - 2|c|)] \, , \tag{S2}$$

where $d = (q + 1)/2$ is the dimension of QRC and 2^d is the number of elements in QRC.

From the above formula, we know that the bias depends on the QRC $\mathcal{C}_{\text{QRC}}$ instead of the matrix $\mathcal{X}_M$ or $\mathcal{X}$. This means that we can append $\mathcal{X}_M$ with arbitrarily many rows orthogonal to s, without changing the the bias. Also, we can do row manipulation to $\mathcal{X}$, and the bias remains unchanged as long as we change s correspondingly s.t. the appending rows are still orthogonal to s. This can scramble the circuit and the secret vector in order to hide it. We write a code for generating the initial QRC matrix and scrambling, and put it at the end of this section as well as in Ref. [33].

From Lemma 1, we know that odd codewords in the original QRC have weight -1 modulo 4, and those of even parity have weight 0 modulo 4, since the extended QRC is doubly even. For a code, the number of even and odd codewords are equal. Thus $q - 2|c|$ in the summation is half the time 1 modulo 8, and half the time -1 modulo 8. So the bias (for $\theta = \pi/8$) is

$$\mathcal{P}_{s\perp} = \frac{1}{2}\cos^2\frac{\pi}{8} + \frac{1}{2}\cos^2\left(-\frac{\pi}{8}\right) = \cos^2\frac{\pi}{8} = 0.854\ .$$

This is the quantum value of bias.

The Optimal Classical Value of Bias

In [29], a classical efficient algorithm was proposed to approximate the bias. The algorithm works as follows:

1. Randomly pick 2 n-bit vectors d and e. (Recall that n is the number of qubits as well as the length of columns in $\mathcal{X}$.)

2. Delete columns in $\mathcal{X}$ that are orthogonal to d, and denote the remaining rows as a matrix $\mathcal{X}_d$. Do the same to get $\mathcal{X}_e$.

3. Denote the sum of rows in $\mathcal{X}_d \cap \mathcal{X}_e$ as y. Then y is the approximate result of x.

By some simple calculation, we can show that $\Pr(y \mid y \cdot s = 0) = \Pr(c_1, c_2 \in \mathcal{C}_{\text{QRC}} \mid c_1 \cdot c_2 = 0)$ [29]. From Lemma 1, we know that the extended QRC is self-dual, which implies the even codewords in the extended QRC is orthogonal to all codewords. Since the extended QRC is obtained by appending a single bit to codewords in QRC to make them all even, the bit appended to even codewords in QRC is 0. So the even codewords in QRC is orthogonal to all codewords in QRC. If the inner product between two codewords is not zero, then they are all odd, which occurs with probability $1/4$. Thus

$$\Pr(y \mid y \cdot s = 0) = \Pr(c_1, c_2 \in \mathcal{C}_{\text{QRC}} \mid c_1 \cdot c_2 = 0) = 1 - \frac{1}{4} = \frac{3}{4}\ .$$

This value is conjected to be the best that a classical computer can approximate *efficiently*.

Python Code for QRC Construction

```
import numpy as np

# set parameters, which can be changed by readers
q = 7 # q must satisfy 8 divides (q+1)
n = int((q+3)/2) # number of qubits
r = 3 # number of redundant rows

def ColAdd(A, i, j):
    '''
    add the j-th column of A to the i-th column
    '''
    if len(A.shape) == 1:
        A_i = A[i]
        A_j = A[j]
        A_i = (A_i + A_j)%2
       A[i] = A_i
    else:
        A_i = A[:,i]
        A_j = A[:,j]
        A_i = (A_i + A_j)%2
       A[:,i] = A_i
    return A

def Redund(n):
    '''
    generate a redundant row
    '''
    a = np.random.choice(2, n)
    a[0] = 0
    return a

def QuadResidue(q):
    '''
    return quadratic residues modulo q
    '''
    qr = []
    for m in range(q):
        qr.append(m**2%q)
    qr.pop(0)
    return list(set(qr))

def Init(n, q, r):
    '''
    generate the matrix
    '''
    result = {}
    P_s = np.zeros([q,n], dtype = int)
    P_s[:,0] = np.ones(q, dtype = int)
    qr = QuadResidue(q)
    for m in range(n-1):
        for m1 in qr:
            P_s[(m1-1+m)%q , m+1] = 1

    P = P_s
    while r > 0:
        r -= 1
        row = Redund(n)
        if (row == np.zeros(n, dtype = int)).all()
:
            r += 1
            continue
        P = np.append(P, [row], axis = 0)
    result["matrix"] = np.unique(P, axis = 0)   #
    delete redundant rows

    s = np.zeros(n, dtype = int)
    s[0] = 1
    result["secret"] = s
    return result

def Scramble(result, times):
    '''
    scramble P and s
    '''
    P = result["matrix"]
    s = result["secret"]
    for m in range(times):
        l = list(range(n))
        i = np.random.choice(l)
        l.pop(i)
        j = np.random.choice(l)
        P = ColAdd(P, i, j)
        s = ColAdd(s, j, i)
    result["matrix"] = P
    result["secret"] = s
    return result

result = Init(n, q, r) # generate the initial
    matrix
print(Scramble(result, 50)) # scramble 50 times,
# and print the matrix
# as well as the secret vector
```

Probability Bias and n-point Correlation Functions

In this section, we first show the relation between bias, Fourier components and n-point correlation functions. Then exploiting this relation, we show why the redundant part of $\mathcal{X}$ has no effect on the value of probability bias. Originally, in Ref. [29], this is proven through Theorem 1. Here, we provide a more straightforward and intuitive way to visualize this fact.

For a probability distribution $\{p(x)\}$, its Fourier coefficient is

$$\begin{aligned}\hat{p}(s) &\equiv \frac{1}{2^n}\sum_x p(x)(-1)^{x\cdot s} \\ &= \frac{1}{2^n}\left[\sum_{x\cdot s=0} p(x) - \sum_{x\cdot s=1} p(x)\right] \\ &= \frac{1}{2^n}(2\mathcal{P}_{s\perp} - 1)\,, \end{aligned} \tag{S3}$$

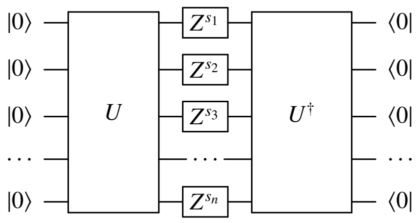


FIG. S1. Quantum circuit representation of $\langle Z^{s_1} \otimes Z^{s_2} \otimes \cdots \otimes Z^{s_n} \rangle$. Here, U can be any quantum circuits. At the same time, this diagram can also be viewed as $\sum_x p(x)(-1)^{s\cdot x}$.

from the normalization of $p(x)$. Denote $\mathcal{Z}_s \equiv Z^{s_1} Z^{s_2} \cdots Z^{s_n}$, then the n-point correlation function for the final state ρ is

$$\langle \mathcal{Z}_s \rangle \equiv \mathrm{Tr}(\mathcal{Z}_s \rho) = \langle 0^n | U^\dagger (Z^{s_1} Z^{s_2} \cdots Z^{s_n}) U | 0^n \rangle \ ,$$

if ρ is a pure state. Fig. (S1) shows the quantum circuit representation of $\langle \mathcal{Z}_s \rangle$. From this representation, we can see that it is actually the Fourier coefficient of s (up to a normalization factor $1/2^n$):

$$\begin{aligned} |0^n\rangle &\xrightarrow{U} \sum_x c_x |x\rangle \\ &\xrightarrow{\mathcal{Z}_s} \sum_x c_x (-1)^{s\cdot x} |x\rangle \\ &\xrightarrow{\langle 0^n | U^\dagger} \sum_x p(x)(-1)^{s\cdot x} \ , \end{aligned}$$

where $c_x = \langle x | U | 0^n \rangle$ is the transition amplitude. Thus $\hat{p}(s) = \langle \mathcal{Z}_s \rangle / 2^n$, and

$$\langle \mathcal{Z}_s \rangle = 2\mathcal{P}_{s\perp} - 1 \ . \tag{S4}$$

It should be noted that this relation is general and not restricted to IQP circuits.

An X-program circuit can be represented by a matrix $\mathcal{X}$ of binary values; its columns represent the gates, as in the Main Text. There are two parts in $\mathcal{X}$, the main part $\mathcal{X}_M$ and the redundant part $\mathcal{X}_R$. We also split the Hamiltonaian read from $\mathcal{X}$ into two part $H = H_M + H_R$, where H_M is translated from $\mathcal{X}_M$ and H_R is from $\mathcal{X}_R$. Then $U_{\mathrm{IQP}} = \exp(i\theta H) = e^{i\theta H_M} e^{i\theta H_R}$, since H_M commutes with H_R. Columns in the redundant part of $\mathcal{X}$ are orthogonal to s, which implies that H_R commutes with $\mathcal{Z}_s$ and so does $\exp(i\theta H_R)$. For example, $(1,1,0,0,0)$ is orthogonal to $s = (1,1,1,1,0)$, and $\left[e^{i\theta X_1 X_2}, \mathcal{Z}_s\right] = 0$. As for H_M, it anticommutes with $\mathcal{Z}_s$, so $e^{i\theta H_M} \mathcal{Z}_s = \mathcal{Z}_s e^{-i\theta H_M}$. Thus

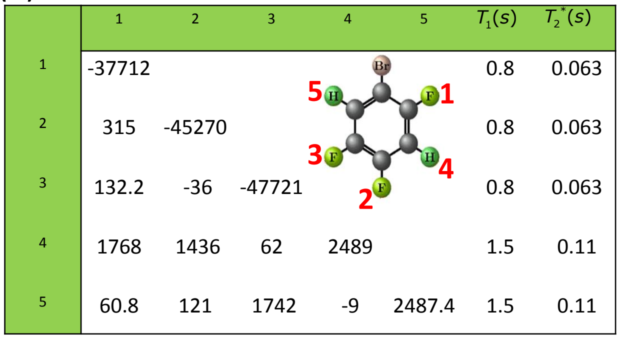
$$\begin{aligned} \langle 0^n | U_{\mathrm{IQP}}^\dagger \mathcal{Z}_s U_{\mathrm{IQP}} | 0^n \rangle &= \langle 0^n | e^{-i\theta(H_M + H_R)} \mathcal{Z}_s e^{i\theta(H_M + H_R)} | 0^n \rangle \\ &= \langle 0^n | e^{-i\theta H_M} \mathcal{Z}_s e^{i\theta H_M} | 0^n \rangle \\ &= \langle 0^n | e^{i 2\theta H_M} | 0^n \rangle \ , \end{aligned}$$

which has no dependence on the redundant part. Together with Eq. (S4), we can see that the value of probability bias $\mathcal{P}_{s\perp}$ does not depend on the redundant part.

Noise effect on probability bias

Consider a simple noise model: single-qubit bit-flip channels $\mathcal{E}(\rho) = (1-\epsilon)\rho + \epsilon X \rho X$ applied to every qubits at the end of the circuits. This model is equivalent to the noise model described in the main text: dephasing channels on every qubit at each time step. Those dephasing channels in between commute with each other as well as the diagonal gates, so they accumulate and can be regarded as an effective single-qubit dephasing channel on every qubit. After commuting with the final layers of Hadamard gates, they become bit-flip channels. Below, we follow Ref. [35] to show that under this noise, $\langle \mathcal{Z}_s \rangle \to (1-2\epsilon)^{|s|} \langle \mathcal{Z}_s \rangle$, and hence from Eq. (S4), $\mathcal{P}_{s\perp} \to \frac{1}{2}\left(1 + (1-2\epsilon)^{|s|} \langle \mathcal{Z}_s \rangle\right)$.

(a)



	1	2	3	4	5	$T_1(s)$	$T_2^*(s)$
1	-37712					0.8	0.063
2	315	-45270				0.8	0.063
3	132.2	-36	-47721			0.8	0.063
4	1768	1436	62	2489		1.5	0.11
5	60.8	121	1742	-9	2487.4	1.5	0.11

Br
5 H
F 1
3 F
H 4
2 F

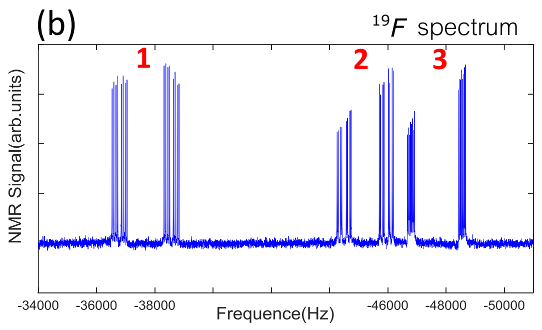


FIG. S2. (a) Molecular structure and parameter of 1-bromo-2,4,5-trifluorobenzene. The diagonal and off-diagonal elements represent the chemical shifts and effective coupling constants in units of Hz, respectively. (b) Equilibrium spectra of ^{19}F nuclear spins. The numbers above the peaks are the indices of qubits.

First, the effect of the measurement operator is $\mathcal{M}(\rho) = \sum_x p(x)\Pi(x)$, where $\Pi(x) = |x\rangle\langle x|$. Note that $\mathcal{E}$ commutes with $\mathcal{M}$. To prove it, we shall first suppose ρ is a single-qubit state. Then

$$\begin{aligned}\mathcal{M}(\mathcal{E}(\rho)) &= (1-\epsilon)\mathcal{M}(\rho) + \epsilon\mathcal{M}(X\rho X)\\ &= (1-\epsilon)\mathcal{M}(\rho) + \epsilon\left[p(0)\Pi(1) + p(1)\Pi(0)\right]\\ &= (1-\epsilon)\mathcal{M}(\rho) + \epsilon X\mathcal{M}(\rho)X\\ &= \mathcal{E}(\mathcal{M}(\rho))\ .\end{aligned}$$

Thus the effect of $\mathcal{E}$ can be viewed as flipping the measurement outcome with probability ϵ. Note that in the expansion of $\mathcal{M}(\rho)$, $\Pi(x)$ is like $|x\rangle$, and they also form a linear space. In this linear space, we introduce a 'Hadamard' operator $\mathcal{H}$: $\mathcal{H}\Pi(i) \equiv \frac{1}{\sqrt{2}}\left[\Pi(0) + (-1)^i\Pi(1)\right]$ for $i = 0, 1$. Then

$$\mathcal{H}^{\otimes n}\Pi(x) = \frac{1}{\sqrt{2^n}}\sum_s (-1)^{x\cdot s}\Pi(s)\ .$$

From the definition of $\mathcal{H}$, we can easily see that $\mathcal{H}^2 = I$. The original expansion of $\mathcal{M}(\rho)$ has probabilities as its coefficients. To explore the effect of noise on Fourier coefficiets, we might want to change basis from $\Pi(x)$ to $\mathcal{H}^{\otimes n}\Pi(s)$:

$$\begin{aligned}\mathcal{H}^{\otimes n}\mathcal{M}(\rho) &= \sum_x p(x)\mathcal{H}^{\otimes n}\Pi(x)\\ &= \frac{1}{\sqrt{2^n}}\sum_x\sum_s p(x)(-1)^{x\cdot s}\Pi(s)\\ &= \sqrt{2^n}\sum_s \hat{p}(s)\Pi(s)\ ,\\ \mathcal{M}(\rho) &= \mathcal{H}^{\otimes n}\mathcal{H}^{\otimes n}\mathcal{M}(\rho)\\ &= \sqrt{2^n}\sum_s \hat{p}(s)\mathcal{H}^{\otimes n}\Pi(s)\ .\end{aligned}$$

Thus in this new basis, the coefficient is exactly $\hat{p}(s)$ (up to a factor). Now, $\mathcal{E}\left[\mathcal{H}\Pi(s_i)\right] = (1-2\epsilon)^{s_i}\mathcal{H}\Pi(s_i)$, which gives $\mathcal{E}^{\otimes n}\left[\mathcal{H}^{\otimes n}\Pi(s)\right] = (1-2\epsilon)^{|s|}\mathcal{H}^{\otimes n}\Pi(s)$, and hence $\hat{p}(s) \to (1-2\epsilon)^{|s|}\hat{p}(s)$. Since $\hat{p}(s) = \langle\mathcal{Z}_s\rangle/2^n$, we have $\langle\mathcal{Z}_s\rangle \to (1-2\epsilon)^{|s|}\langle\mathcal{Z}_s\rangle$ under this noise model.

Experimental Details

NMR processor

Device. The experiments run on NMR quantum processor were carried on a Bruker AV-400 spectrometer at 303K. The 5-qubit quantum processor consists of two ^1H nuclear spins and three ^{19}F nuclear spins in 1-bromo-2,4,5-trifluorobenzene dissolved in

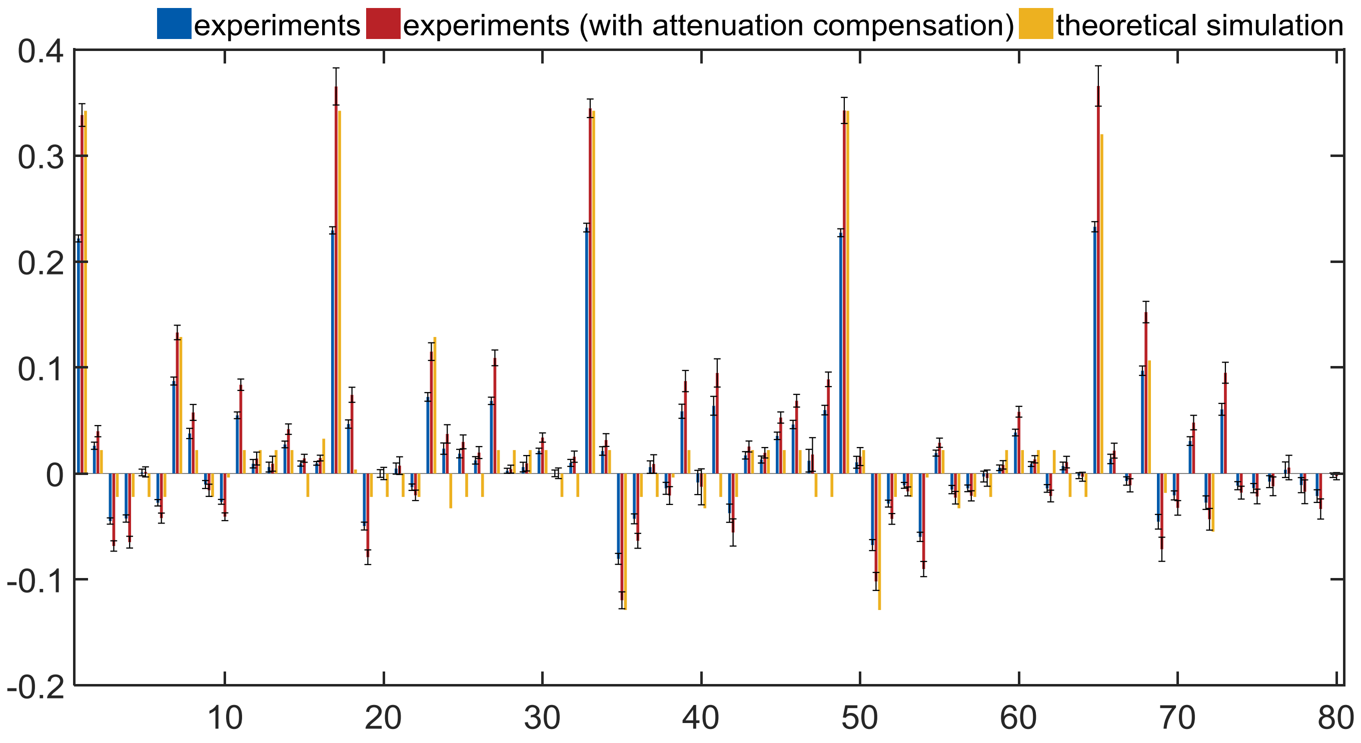


FIG. S3. Peak intensities from five readout pulses. Each pulse creates 16 peak intensities, which are linear combination of probabilities, so there are 80 peak intensities for each kinds of data.

the liquid crystal N-(4-methoxybenzylidene)-4-butylaniline (MBBA) [38]. Figure S2 (a) shows the molecular structure and (b) shows equilibrium spectra of ${}^{19}F$ nuclear spins.

The effective Hamiltonian of the NMR sample in the rotating frame is denoted as:

$$H_{NMR} = \sum_{j=1}^{5} \pi \nu_j Z_j + \sum_{1\leqslant j\leqslant k\leqslant 5} \frac{\pi}{2}\left(J_{jk} + 2D_{jk}\right) Z_j Z_k \, , \tag{S5}$$

where Z_j are the Pauli-Z matrices acting on the j-th qubit. The chemical shifts ν_j and the effective coupling constants $J_{jk} + 2D_{jk}$ are shown in Fig. S2 (a). Here we have employed the secular approximation since that the chemical shift difference in each pair of spins is much higher than the effective coupling strength.

Readout. As we mentioned in the main text, the probability distributions for the IQP circuit is obtained by reconstructing the diagonal elements of the density matrix of the final state. To obtain the information of the j-th qubit, We applied the readout pulse $\exp\left(-i\pi Y_j/4\right)$ and then read the NMR spectrum of it. The first qubit, i.e. F_1, is used as the readout channel in the experiment. The spectrum of F_1 is read directly and the information of other qubits are swapped to F_1 before obtaining the NMR spectra. For each qubit, we can obtain a readout spectrum which contains 16 peaks. Taking the spectrum of third qubit as example, the intensities of the peaks equal to $\Pr(x_1x_20x_4x_5) - \Pr(x_1x_21x_4x_5)$, where $x_1, x_2, x_4, x_5 \in \{0, 1\}$. From the intensities of all the 80 peaks, We have 80 linear equations of the form: $\sum_x c_{lx} \Pr(x) = \alpha_l$ $(1 \leqslant l \leqslant 80)$, where α_l is the intensity of the l-th peak. Therefore, the probability distributions $\Pr(x)$ can be obtained by measuring the intensities of the peaks and then solving these linear equations.

The intensities of all the 80 peaks of the five spins are shown as the blue histogram in Fig. S3. The orange one is the ideal result from the theoretical simulation. In the experiments, the intensities of the NMR signal suffer a significant attenuation due to the overlong lengths of the pulses. To compensate this effect, we numerically measure the attenuation speed of each spin by independent benchmarking experiments. The attenuation factors in the duration of the pulse sequence are estimated as 0.66 ± 0.02, 0.63 ± 0.03, 0.67 ± 0.01, 0.66 ± 0.02 and 0.64 ± 0.03 for the five spins. The attenuation of the decoherence effect then can be compensated by dividing the intensities of the peaks by the corresponding attenuation factors. In Fig. S3, the red histogram shows the intensities of peaks with the attenuation compensation, which has a good agreement with the theoretical expectation.

The probability distributions $\Pr(x)$ can be estimated by solving the overdetermined linear equations $\sum_x c_{lx} \Pr(x) = \alpha_l$ $(1 \leqslant l \leqslant 80)$, on condition that $\sum_x \Pr(x) = 1$ and $\Pr(x) \geqslant 0$. Here α_l represents the compensated intensity of the l-th peak. With the least

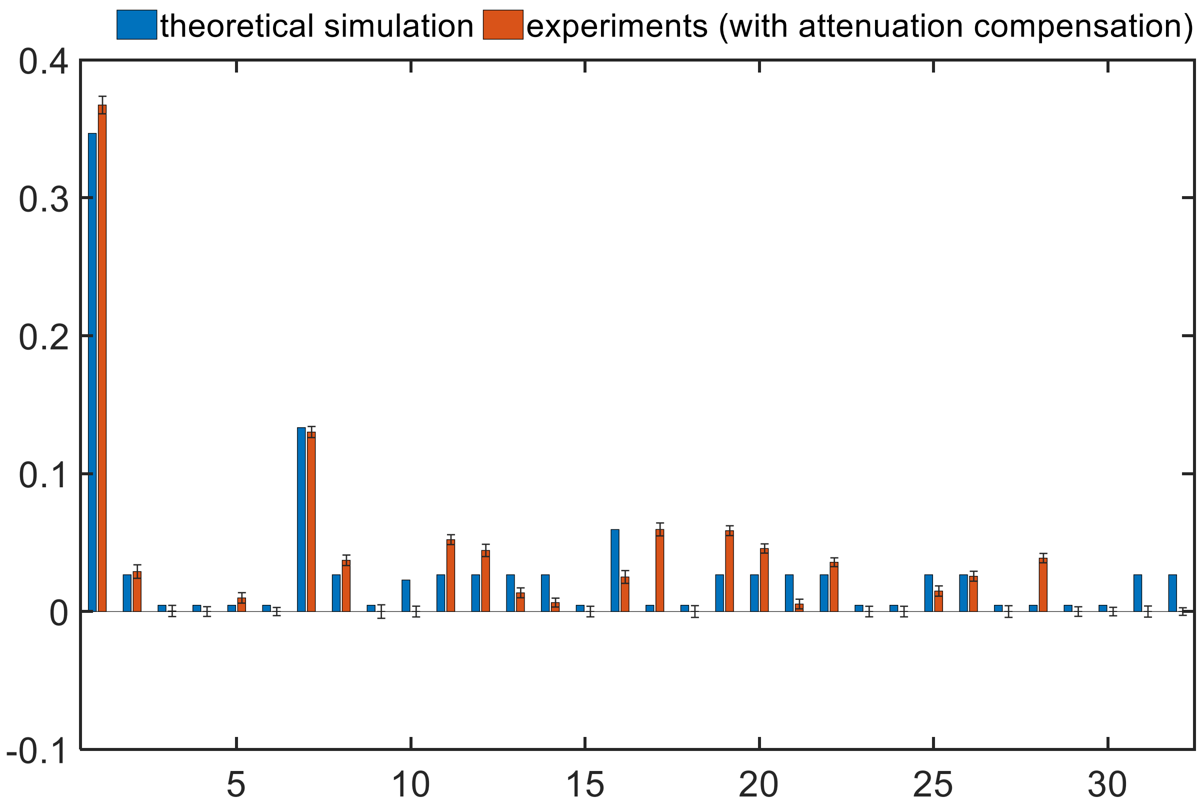


FIG. S4. The blue histogram is from theoretical distribution and the red one is from distribution from the NMR device after noise compensation.

square method, we obtain a probability distribution after compensation of noise (Fig. S4), whose probability bias with respect to the secret string s is 0.866 ± 0.016. The distance between this distribution and the theoretical distribution is 0.23, where the distance between two distributions p_i and q_i is defined by $D = \frac{1}{2}\sum_{i=1}^{32}|p_i - q_i|$.

Estimation of the readout errors. The readout errors of the intensities of the peaks mainly come from two parts: the statistical fluctuation of the NMR spectra and the error in the compensation procedure. In the experiments, the intensities of the peaks are obtained by fitting the NMR spectra to a sum of several Lorentz functions. The results of fitting indicate that these intensities have an average standard error of 0.004, shown as the error bars on the blue histogram in Fig. S3. To compensate the attenuation of the signal due to the decoherence effect, these intensities are divided by an attenuation factor, e.g. 0.66 ± 0.02 for the peaks of the first spin. The error of the attenuation factors also contribute to the compensated results. The readout errors of the compensated intensities of the peaks are shown in Fig. S3, with the average error being 0.007. The estimation of the readout errors in the probability distributions $\Pr(x)$ are shown as the error bars in Fig. S4. The probability bias is estimated as 0.866, with a standard error of 0.016.

The error from the imperfection of experimental implementation. In the experiments, the results are also affected by the imperfection of the implementation, including the imperfection of the initial state and the pulse sequences. We experimentally measured the population of the initial state on the computational basis (Fig. S5), while the ideal initial state is (pseudo) $|00000\rangle$ state. It shows that the population of some quantum state is not zero as expected and thus may affect the final results. The pulse sequence employed to realize the IQP circuit is also not perfect in the experiment. Numerical simulation shows that the fidelity of the shaped pulse used may reduce to 97.5% if we take into consideration the imperfection of the secular approximation in the Hamiltonian of the quantum processor (Eq. (S5)) and the thermal fluctuation of the Hamiltonian parameters. To understand to what extent these imperfections will affect the experimental result, we numerically simulate these effects and show the final state with the presence of these imperfection in Fig. S6, as well as the ideal results without any error. The distance D between ideal final state and the one with error is around 0.08. The probability bias is reduced to 0.832, while the ideal expectation is 0.854. Thus we estimated that these imperfections may lead to a drift of -0.02 on the probability bias.

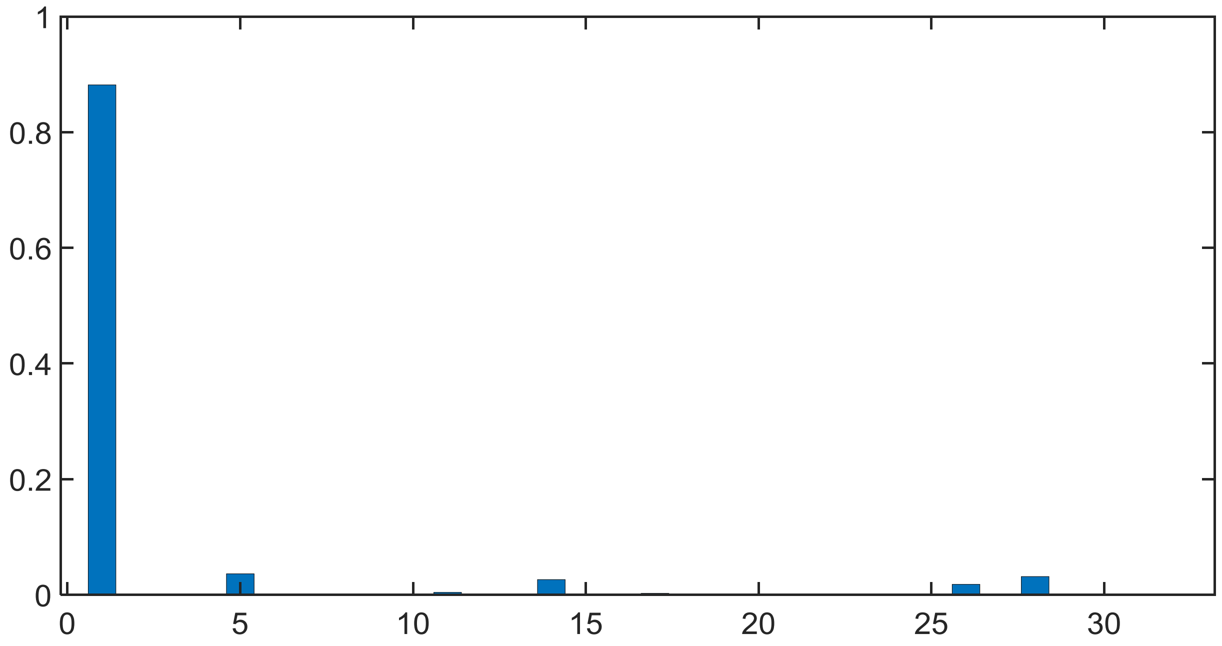


FIG. S5. The experimentally measured population of the initial state on the computational basis, listed in the order from $|00000\rangle$ to $|11111\rangle$.

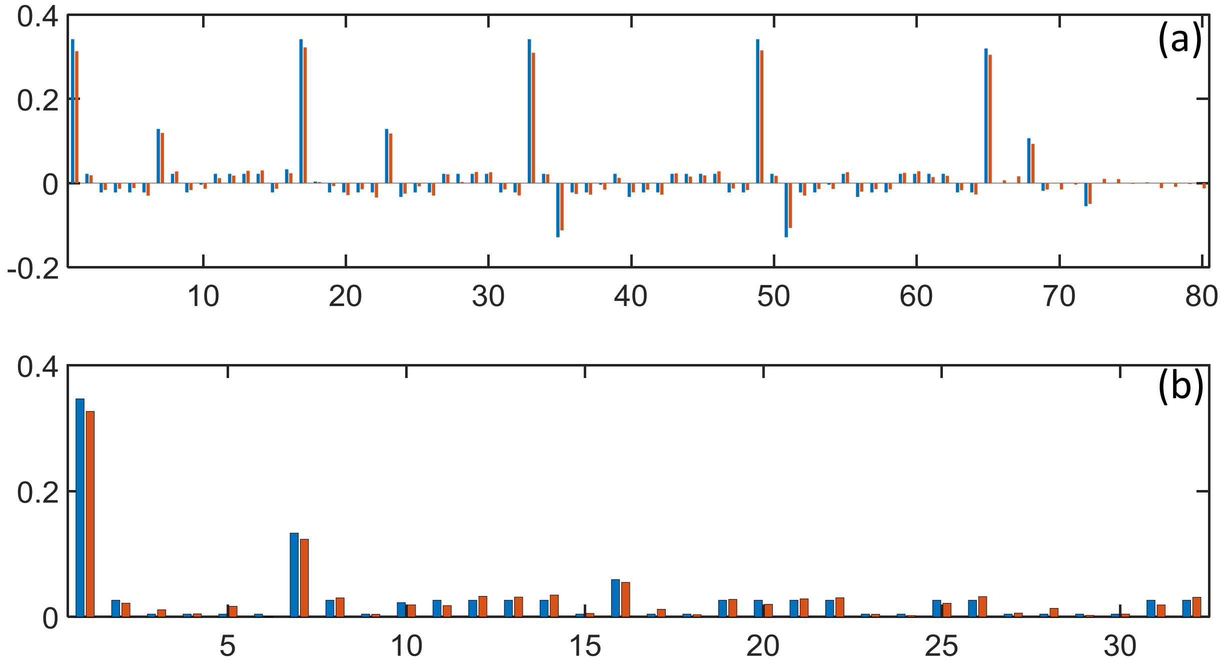


FIG. S6. The numerically simulated results of the final state with (red) and without (blue) the presence of the imperfection of the implementation. (a) The intensities of the peaks of the readout spectra. (b) The probability distributions $\Pr(x)$ of the final state after the IQP circuit.

IBM Devices

We use *ibmqx4* and *ibmqx5* for our experiments. We collect some specification of IBM devices here, and readers can find more details in Ref. [6]. Fig. S7 shows the connectivity of the two IBM devices. There are 16 qubits in *ibmqx5*, and we used the first 5 qubits, indicated as the red circles in Fig. S7 (b). The first table of Table S1 shows the frequency, relaxation time and coherence time of *ibmqx4*, and the second shows those of *ibmqx5*. The IBM devices are calibrated constantly, so the T_1 and T_2

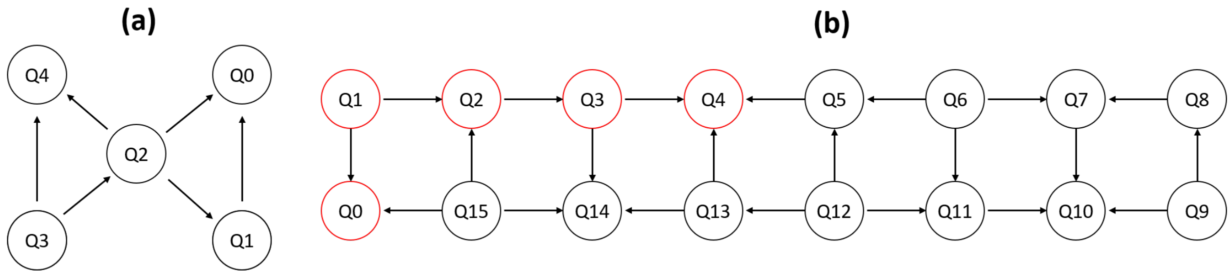


FIG. S7. Connectivity of **(a)** *ibmqx4* and **(b)** *ibmqx5*. Here $a \rightarrow b$ means qubit a controls qubit b. The red circles in **(b)** are the qubits we use in our experiment.

	Q0	Q1	Q2	Q3	Q4
Frequency (GHz)	5.24	5.30	5.35	5.43	5.18
$T_1(\mu s)$	48.81 ± 0.68	50.24 ± 0.66	42.52 ± 0.51	40.09 ± 0.94	55.52 ± 0.96
$T_1(\mu s)$	28.09 ± 0.41	60.24 ± 1.12	34.92 ± 0.51	14.24 ± 0.21	27.09 ± 0.37

	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
Frequency (GHz)	5.26	5.40	5.28	5.08	4.98	5.15	5.31	5.25	5.12	5.16	5.04	5.11	4.95	5.09	4.87	5.11
$T_1(\mu s)$	37 ± 4	35 ± 4	48 ± 6	46 ± 5	49 ± 8	49 ± 4	44 ± 7	37 ± 4	49 ± 7	48 ± 5	28 ± 21	45 ± 7	50 ± 7	48 ± 6	36 ± 3	48 ± 7
$T_2(\mu s)$	31 ± 5	58 ± 10	64 ± 7	70 ± 15	74 ± 24	50 ± 5	74 ± 12	49 ± 7	68 ± 19	88 ± 14	49 ± 36	86 ± 16	33 ± 3	82 ± 12	65 ± 6	89 ± 17

TABLE S1. Specification of IBM devices. The first table collects specification of *ibmqx4*, and the second collects that of *ibmqx5*.

listed in Table S1 are only mean values over an interval, according to the version number in Ref. [6]. The number after the ± sign shows the standard deviation of the mean.

We access *ibmqx4* through the website of IBM quantum experence and can edit the circuit on a graphical interface. Fig. S8 shows the circuit run on *ibmqx4*. This circuit is equivalent to that of Fig. 2 in the main text. but due to the connectivity constraints on the device, we need to add many SWAP gates to implement the protocol on *ibmqx4*. But for *ibmqx5*, we need to access it through QISKit, an open-source software based on python. The code used to construct the circuit is shown in the end of the Supplemental Materials, and the full version can be found in Ref. [33].

On *ibmqx4*, we also implement the quantum circuit corresponding to the main part of the QRC matrix only, that is matrix (S1). Components in Fig. S8 enclosed by blue dashed lines represent the circuit. The output probability distribution is shown in Fig. S9. As we show in the main text, the probability bias of the secret vector $s = (1, 1, 1, 1, 0)$ only depends on the main part of the X-program matrix. From the output distribution, we obtain that the bias is 0.512, which is still very close to bias derived from a completely mixed state. This indicates that even if we significantly reduce the depth of the circuit, the final state is still highly corrupted by noise. Thus, we conclude that the fidelity of IBM cloud needs to be significantly improved in order to pass the test.

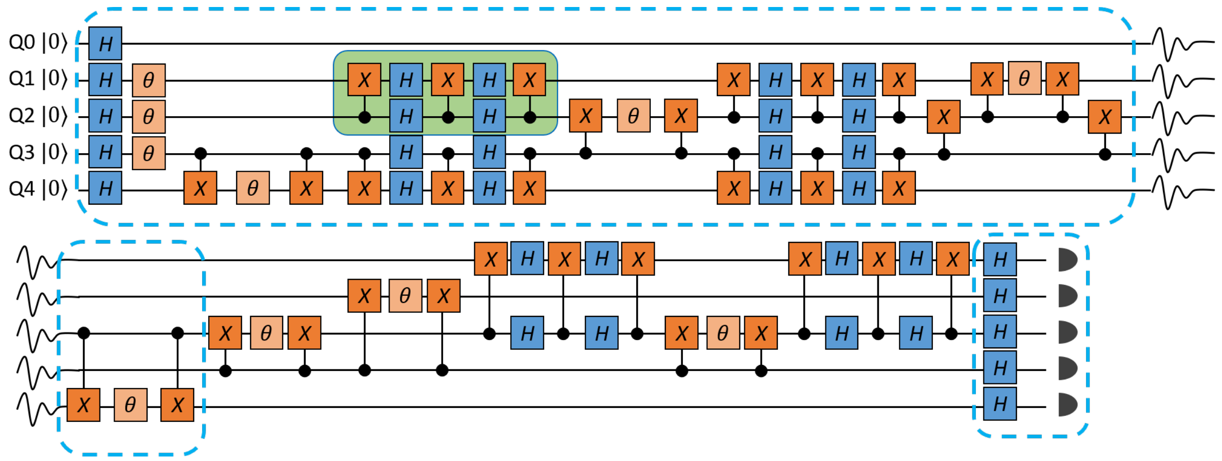


FIG. S8. The *X*-program circuit run on *ibmqx4*. The green block is a SWAP gate. Components enclosed by blue dashed lines form a circuit translated from the main part of the *X*-program matrix in the main text, i.e. matrix (S1).

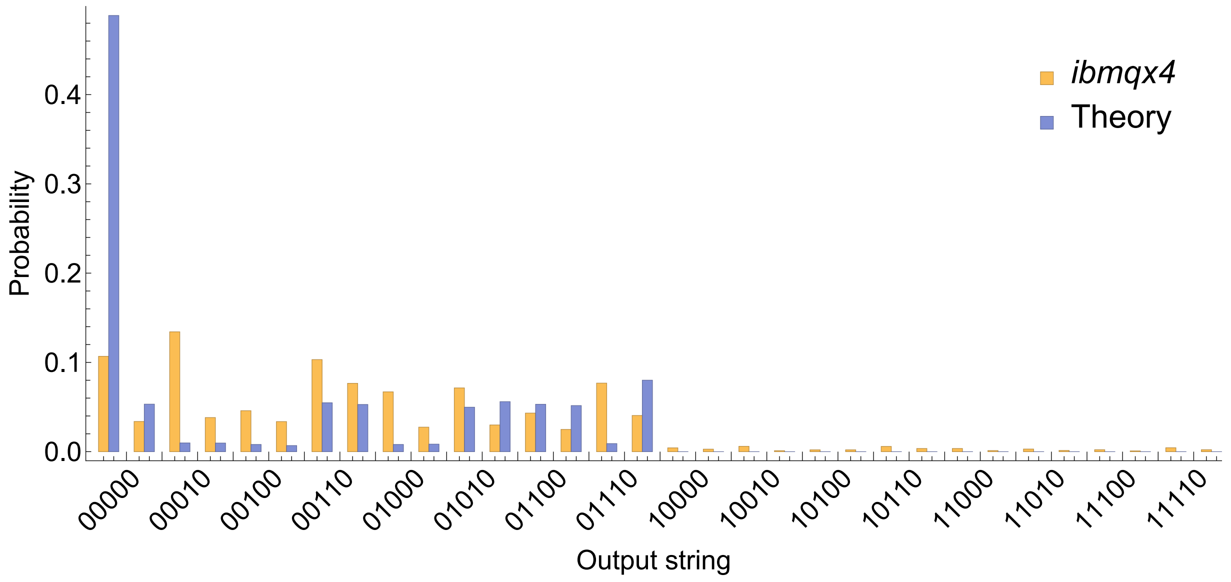


FIG. S9. Probability distribution from the IQP circuit translated from matrix (S1).

Python code to specify the circuit run on *ibmqx5*

```
# circuit

for i in range(5):
    qc.h(qr[i])  # Hadamard gate

for i in [1,2,3]:
    qc.tdg(qr[i])  # T−dagger

# (0,0,0,1,1)
qc.cx(qr[3], qr[4])  # CNOT with Q3 controling Q4
qc.tdg(qr[4])
qc.cx(qr[3], qr[4])

# (0,1,0,0,1)
qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])
for i in [1,2,3,4]:
    qc.h(qr[i])
qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])
for i in [1,2,3,4]:
    qc.h(qr[i])
qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])

qc.cx(qr[2], qr[3])
qc.tdg(qr[3])
qc.cx(qr[2], qr[3])

qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])
for i in [1,2,3,4]:
    qc.h(qr[i])
qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])
for i in [1,2,3,4]:
    qc.h(qr[i])
qc.cx(qr[1], qr[2])
qc.cx(qr[3], qr[4])

# (0,1,1,1,0)
qc.cx(qr[1], qr[2])
qc.cx(qr[2], qr[3])
qc.tdg(qr[3])
qc.cx(qr[2], qr[3])
qc.cx(qr[1], qr[2])

# (0,0,1,0,1)
qc.cx(qr[3], qr[4])
for i in [3,4]:
    qc.h(qr[i])
qc.cx(qr[3], qr[4])
for i in [3,4]:
    qc.h(qr[i])
qc.cx(qr[3], qr[4])

qc.cx(qr[2], qr[3])
qc.tdg(qr[3])
qc.cx(qr[2], qr[3])

qc.cx(qr[3], qr[4])
for i in [3,4]:
    qc.h(qr[i])
qc.cx(qr[3], qr[4])
for i in [3,4]:
    qc.h(qr[i])
qc.cx(qr[3], qr[4])

# (0,0,1,1,0)
qc.cx(qr[2], qr[3])
qc.tdg(qr[3])
qc.cx(qr[2], qr[3])

# (1,0,1,0,0)
qc.cx(qr[1], qr[0])
for i in [0,1]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
for i in [0,1]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])

qc.cx(qr[1], qr[2])
qc.tdg(qr[2])
qc.cx(qr[1], qr[2])

qc.cx(qr[1], qr[0])
for i in [0,1]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
for i in [0,1]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])

# (1,0,0,1,0)
qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])
for i in [0,1,2,3]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])
for i in [0,1,2,3]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])

qc.cx(qr[1], qr[2])
qc.tdg(qr[2])
qc.cx(qr[1], qr[2])

qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])
for i in [0,1,2,3]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])
for i in [0,1,2,3]:
    qc.h(qr[i])
qc.cx(qr[1], qr[0])
qc.cx(qr[2], qr[3])

for i in range(5):
    qc.h(qr[i])

# measure
for j in range(5):
    qc.measure(qr[j], cr[j])
```