Split Compilation for Security of Quantum Circuits

Abdullah Ash Saki
Dept. of Electrical Engineering
Pennsylvania State University
University Park, PA
axs1251@psu.edu

Aakarshitha Suresh
Dept. of Electrical Engineering
Pennsylvania State University
University Park, PA
ams9647@psu.edu

Rasit Onur Topaloglu
IBM
Poughkeepsie, NY
rasit@us.ibm.com

Swaroop Ghosh
Dept. of Electrical Engineering
Pennsylvania State University
University Park, PA
szg212@psu.edu

2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 978-1-6654-4507-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCAD51958.2021.9643478

***Abstract*—An efficient quantum circuit (program) compiler aims to minimize the gate-count - through efficient instruction translation, routing, gate, and cancellation - to improve run-time and noise. Therefore, a high-efficiency compiler is paramount to enable the game-changing promises of quantum computers. To date, the quantum computing hardware providers are offering a software stack supporting their hardware. However, several third-party software toolchains, including compilers, are emerging. They support hardware from different vendors and potentially offer better efficiency. As the quantum computing ecosystem becomes more popular and practical, it is only prudent to assume that more companies will start offering software-as-a-service for quantum computers, including high-performance compilers. With the emergence of third-party compilers, the security and privacy issues of quantum intellectual properties (IPs) will follow. A quantum circuit can include sensitive information such as critical financial analysis and proprietary algorithms. Therefore, submitting quantum circuits to untrusted compilers creates opportunities for adversaries to steal IPs. In this paper, we present a *split compilation* methodology to secure IPs from untrusted compilers while taking advantage of their optimizations. In this methodology, a quantum circuit is split into multiple parts that are sent to a single compiler at different times or to multiple compilers. In this way, the adversary has access to partial information. With analysis of over 152 circuits on three IBM hardware architectures, we demonstrate the split compilation methodology can completely secure IPs (when multiple compilers are used) or can introduce factorial time reconstruction complexity while incurring a modest overhead ($\approx 3\%$ to $\approx 6\%$ on average).**

***Index Terms*—Quantum Computing, Transpilation, Split, Obfuscation, Overhead, Coupling Map, IP theft, Compilation**

I. Introduction

The compilation is an important step to convert the quantum programs written in high-level gate sets to the low-level (native) gate sets tailored for the underlying hardware. The compilation is also key to the success of the program in the actual hardware. A well-optimized program with fewer gates and less depth can provide meaningful results of the problem at hand from the real hardware whereas a poorly optimized program (for identical functionality) can produce random outcomes from the hardware.

The hardware vendors of quantum computers offer a compiler for their hardware such as Qiskit compiler from IBM [1], QuilC compiler [2] from Rigetti, etc. Besides, some 3^{rd} party software tools, including compilers like Orquestra [3] and tKet [4], are also appearing which supports hardware from multiple vendors. As the quantum computing ecosystem evolves, more 3^{rd} party compilers will emerge offering potentially higher performance. This will entice users to utilize these services. However, the trustworthiness of 3^{rd} party compilers can become a security issue [5].

A quantum circuit can include sensitive intellectual properties (IPs) such as, the problem being solved, financial analysis, and proprietary algorithms which must be protected from untrusted third parties. As pointed out in [5], reliance on untrusted third-party compilers can expose the quantum circuit to threats such as information theft.

We thank Mahabubul Alam (Penn State) for helpful discussions. This work is supported by National Science Foundation (NSF) (OIA-2040667 and DGE-2113839) and seed grants from Penn State Institute for Computational and Data Sciences (ICDS) and Penn State Huck Institute of the Life Sciences.

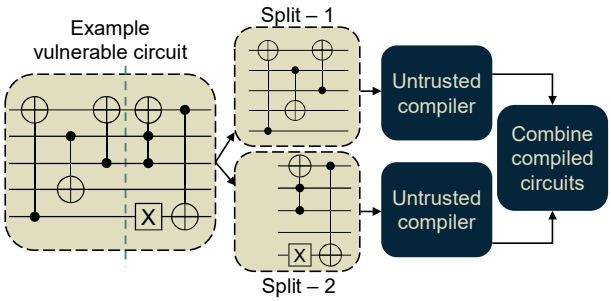

Fig. 1: The proposed split compilation. The sub-circuits could be sent to the same/different compilers.

We propose an approach to obfuscate the quantum circuit that relies on the following observations: (i) the existing compilation techniques heavily depend on local optimization using windowing technique i.e., the gates present in the nearest few layers of the circuit are optimized together [6]–[8]. This is primarily due to the poor scalability of global optimization methods. Therefore, larger circuits can be split into smaller sub-circuits without compromising the optimization quality significantly, and (ii) quantum circuits can offer significant corruptibility of outputs with few omitted layers of gates.

Proposed Idea: Based on the observations presented above, we propose splitting the quantum circuit pre-compilation into two or more sub-circuits. The splitting could be even (e.g., 50%-50%) or uneven (e.g., 20%-80%). We expect an overhead since the optimization opportunities around the splitting points will be lost (exploits observation (i)). However, the overhead may not be significant due to the local optimization used by the compilers. Each sub-circuit omitted from the original circuit will heavily corrupt the functionality. More splitting will be beneficial since it will present more combinations to the adversary to brute force. Finally, the sub-circuits could be sent to the same untrusted compiler in randomized order (e.g., the order of the sub-circuits) to impose a high reverse engineering effort. Note, the adversary lacks the oracle model so he will not be able to validate his brute-force guess. The sub-circuits will be stitched by the designer post-compilation. To further increase the adversarial effort, we also propose another approach where the sub-circuits are sent to multiple untrusted compilers. Fig. 1 exemplifies this idea further.

Contributions: In this paper we (a) propose split compilation as a secure circuit obfuscation technique, (b) perform a complete overhead analysis for multiple splits to be compiled on a single compiler or distributed among multiple compilers, (c) propose a new attack mode, where the adversary can use the coupling map of the splits to guess

978-1-6654-4507-8/21/$31.00 ©2021 IEEE

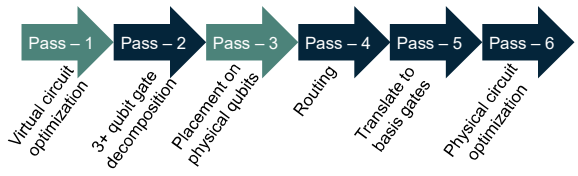

Fig. 2: Compilation (transpilation) steps from qiskit compiler. The dark-colored passes add/remove news gates in the circuit.

the correct order, (d) propose a robust swap router layer to map physical layout of splits to layouts of each other, (e) coherently justify the experimental results and conclude.

Paper Organization: In the remaining of the paper, Section II covers the background for the proposed idea. Section III describes the threat model and the motivation. Section IV presents the proposed idea and experimental procedure in detail. Section V contains the experimental results and corresponding analysis. Section VII concludes the paper.

II. Background

A. Compilation of quantum circuits

1) Gate translation: Before a quantum circuit can be executed on hardware, it undergoes several steps. The steps, collectively, are named *compilation* of quantum circuits (in IBM terminology it is called *transpilation*). Fig. 2 shows the steps used in IBM's qiskit compiler. Quantum programs are usually written with high-level gates. Current quantum computers support only a restricted set of gates known as basis gates (e.g., IBM machines support following gates {`'ID'`, `'SX'`, `'X'`, `'CX'`}). Therefore, any high-level instructions are translated to native instructions of the hardware.

2) Coupling constraint and mapping: Besides instruction-set misalignment, present hardware architectures have another challenge known as *coupling constraint*. Fig. 3 demonstrates the constraint. Fig. 3a shows the coupling graph of IBM's Vigo architecture where the nodes represent physical qubits. An edge between 2 nodes signifies that 2-qubit operation (CX gate) between those physical qubits are directly allowed. Fig. 3b shows a sample 3-qubit quantum program that we want to run on the Vigo architecture. To run the program on the hardware, each program (logical) qubit has to be mapped to a separate physical qubit. This is known as the *initial layout* which describes the starting physical-to-virtual (p2v) qubit mapping (note, virtual-to-physical mapping renders the same information). For this example, we assume a p2v mapping of {`Q0` → `L0`, `Q1` → `L1`, `Q2` → `L2`}. The gate `CX L0, L2` from the program cannot be directly executed with this mapping as there is no edge in Vigo's coupling graph between Q0 (L0) and Q2 (L2) (Fig. 3c). This is the coupling constraint. To resolve the constraint, qubits are *routed* using SWAP operation so that logical qubits with 2-qubit operations become nearest neighbors. Fig. 3d shows a SWAP gate between Q1 and Q2 which brings L0 and L2 closer so that CX L0, L2 can be applied. Note that, adding a SWAP gate modifies the p2v mapping. After the SWAP(Q1, Q2) the p2v becomes {`Q0` → `L0`, `Q1` → `L2`, `Q2` → `L1`}. The p2v map after all gates are finished is known as the *final layout*.

B. Circuit optimizations

The SWAP operation is not native to IBM architectures, and it is decomposed using 3 alternating CX gates. Thus, the routing step introduces more gates in the final compiled circuit. The routing algorithms in quantum compilers aim to route efficiently to minimize

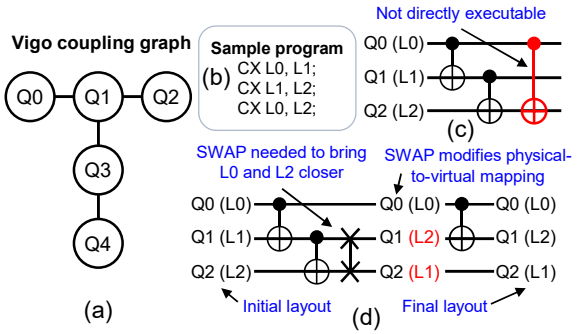

Fig. 3: (a) Coupling graph of ibmq_vigo. (b) A sample quantum program. (c) Example of physical-to-virtual mapping of the program. The CX L0, L2 gate cannot be directly executed as the physical qubits Q0 and Q2 are not coupled. (d) Adding SWAP to resolve coupling constraint. SWAP gate modifies the mapping.

the number of added SWAPs. The efficiency of a compiler depends heavily on the efficiency of the routing process. Besides efficient routing, compilers also perform several optimizations like gate cancellation and/or gate replacement. For example, compilers can cancel two back-to-back CX gates working on the same set of control and target qubits or replace a chain of single-qubit gates with a simpler equivalent gate. The key objective is to reduce the gate count in the final compiled (machine-executable) circuit.

C. Related work

In the classical domain, there is the concept of *split manufacturing* [9]–[12] which protects a classical chip from untrusted foundries. In this approach, the front-end-of-line (FEOL) (transistors) and the backend-of-the-line (interconnects) are fabricated using separate foundries and finally integrated into a trusted facility. The functionality of a chip depends on both the transistors and the interconnection between them. As no single untrusted foundry has information about both, it protects the chip from IP theft, over-production, etc.

In the quantum domain, security in manufacturing a quantum computer is not the main concern. Quantum gates of a quantum circuit are realized using microwave or laser pulses applied on quantum bits. The underlying quantum computing hardware is the same even for different quantum circuits run on it. However, the quantum circuits themselves need protection. As quantum circuits are applied on generic quantum hardware, the split manufacturing method from the classical domain is not directly applicable in the quantum domain for protecting the quantum circuits.

There are several recent works on the security of quantum computing [5], [13]–[19]. In [14], authors consider an attack model where a rogue element in the quantum cloud can report incorrect device calibration data due to which, a user may run his/her program on an inferior set of qubits. The authors propose inserting test points in the circuit to detect any malicious change of calibration data. In [16], the authors assume a similar attack model where a malicious entity in the cloud can schedule a user circuit to inferior hardware instead of the requested one. They propose quantum PUFs (QuPUFs) to authenticate the requested device. A closely related work [5] obfuscates the functionality of a quantum circuit from untrusted compilers using dummy gates.

The works in [13], [17] address security issues in a multi-programming environment where programs from different users run parallel on the same hardware.

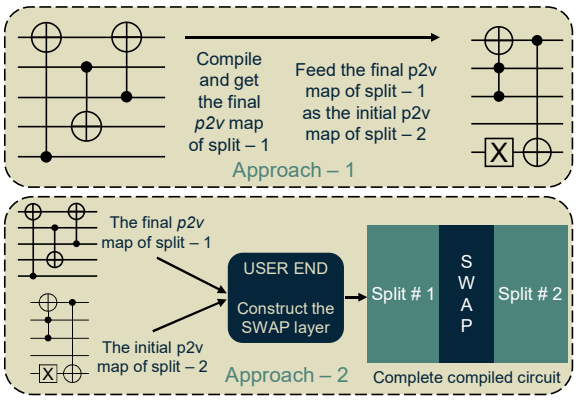

Fig. 4: Approaches to achieve continuity of p2v mappings between split circuits.

Another line of work [15] studies watermarking approaches for quantum circuits, including embedding secret signatures in the decomposition phase of the compilation. This is performed to verify the ownership of the IP. Finally, several works exist on the IP/IC piracy of reversible circuits [18], [19] that aim to protect reversible IPs from untrusted foundries. Although gate-based quantum circuits are based on reversible logic, they are not physically fabricated as mentioned earlier.

III. Threat Model

In this paper, we assume that the compiler package is cloud-based and hosted remotely by an untrusted third party. We also assume that the compilers involved cannot be trusted, or there could even be some rogue adversary who could attack the netlist to retrieve sensitive information from the quantum circuit. As a result, the user that sends the sensitive unprotected circuit to the compiler may be suspect of unwanted theft, counterfeiting, and many such adversarial problems. For example, the adversary can get the sensitive outputs of a quantum circuit by running it, and this can potentially reveal much more information about the algorithm it was designed for or its functionality. Hence, this could prove to be a significant problem, as many such users look towards third-party sources to optimize their work, make it less dense and less complex, and more efficient in nature, both in classical and quantum worlds. This paper, thus, proposes a novel idea in the field of obfuscation or hiding the functionality of the insecure quantum circuit before it is sent to such compilers or other sources. More details of this attack model are also provided in [5].

IV. Proposed Split Compilation Procedure

In this section, we provide a detailed procedure of split compilation. These split partial circuits are sent to the compiler in random order to hide the true original circuit's complete structure from the rogue adversary. These circuits are later stitched or combined post transpilation to recover the full compiled circuit. A basic swap layer algorithm is designed in this paper to synchronize the coupling maps between the two splits appropriately to ensure data integrity.

The idea of split compilation is to split a high-level quantum circuit into two or more parts and send them to a compiler or multiple compilers. If a single compiler is used, then the split circuits are sent to the compiler at different times and in random order. If multiple compilers are available, then the split circuits can be sent without risking security.

The proposal hinges on two key questions: (i) What is the overhead of the splitting? and (ii) How secure is the solution? We address both questions by performing overhead analysis in terms of gate counts for 2 to 8 splits over 152 circuits into three different IBM architectures i.e., 5-qubit ibmq_vigo (vigo), 14-qubit ibmq_melbourne (melbourne)[1], and 20-qubit ibmq_almaden (almaden).

We construct many 5-qubit, 10-qubit, 14-qubit, and 20-qubit random circuits of varied depth (from 10 to 100 at an interval of 5) to perform the overhead analysis of the split compilation. The 5-qubit circuits are compiled using coupling graphs of all three architectures whereas 10 and 14-qubit circuits are compiled for both melbourne and almaden, and 20-qubit circuits are compiled for almaden only. This approach covers a wide range of circuit width (number of qubits), depth, and architectures.

We introduce a metric named *weighted gate count (W)* for the overhead analysis. Not all gates have the same implementation cost and thus trivially adding all types of gates is not a fair representation. In IBM architectures, the native (basis) gates are `'ID'`, `'RZ'`, `'SX'`, `'X'` and `'CX'`. The high-level circuit is compiled into these gates. We use the following equation for W:

$$\begin{aligned} W = & \quad w_{cx}.n(cx) + w_x.n(x) + w_{sx}.n(sx) \\ = & \quad 1.0.n(cx) + 0.2.n(x) + 0.1.n(sx) \end{aligned}$$

The weight of 2-qubit CX gate is set to 1.0 whereas the weight of 1-qubit SX gate is set to 0.1. This is primarily because 2-qubit gates typically have an order of magnitude higher gate error and gate time than 1-qubit counterparts (e.g., the average 2-qubit CX error in the Melbourne architecture is 3.45×10^{-2} where average 1-qubit SX gate error is 1.65×10^{-3}).

An SX gate contains one $X_{\pi/2}$ pulse whereas an X gate has two $X_{\pi/2}$ pulses. Therefore, the X gate takes 2X longer to run and typically incurs 2X more error than the SX gate. Therefore, we set w_x to 0.2. Now, RZ is a virtual gate [20] which takes zero physical time and incurs no error. Therefore, it is omitted from the weighted gate count equation ($w_{rz} = 0$).

We compute weighted gate counts for both the original circuit and the split combined circuit. Then, we compare and report the percentage change in weighted gate counts between two implementations. Between splits, the continuity of p2v qubit mapping (layout) needs to be maintained for data integrity. There can be two approaches to achieve this (Fig. 4).

Approach–1: In this approach, the user needs to feed the final layout of the previous split as the initial layout of the next split. In that way, the continuity of maps and integrity of computation is preserved.

Approach–2: In this approach, each split starts with its initial layout. After compilation, the user collects the compiled circuits and adds a SWAP layer in between compiled splits. The SWAP layer converts the final layout of the previous split to the initial layout of the next split, and thus, ensures continuity and integrity.

We propose a greedy algorithm for the SWAP layer. The SWAP layer will introduce more gates in the combined circuit, potentially increasing the overhead. We perform an overhead analysis with the added SWAP layer. We discuss the security implications of approach–1 and approach–2 in Section VI.

Example 4.1: Consider the coupling graph of the vigo architecture (Fig. 5). The final layout (source layout, S) of the split–1 is {`Q0` → `L3`, `Q1` → `L4`, `Q2` → `L1`, `Q3` → `L3`, `Q4` → `L0`}. The initial layout (target layout, T) of the split–2

[1]Although the last version of the melbourne architecture had 15 qubits, we used an older version with 14 qubits.

Algorithm 1: Create SWAP layer

Input: hardware coupling graph (G), source layout (S), target layout (T)
Output: SWAP layer (SL)

```
interim_layout ← S;
remaining_nodes ← all out of position nodes in S;
num_swaps ← 0;
while interim_layout ≠ T do
    min_distance ← ∞;
    for node ∈ remaining_nodes do
        src ← node position in interim layout;
        dest ← node position in target layout;
        dist ← shortest distance(G, src, dest);
        h_dist ← heuristic_distance(G, src, dest);
        total_dist ← dist + h_dist;
        if total_dist < min_distance then
            min_distance = dist;
            move_from = src;
            move_to = dest;
        end
    end
    simple_path ← shortest_simple_path(G, move_from, move_to);
    Add SWAPs in SL;
    distance ← edges in the simple_path;
    num_swaps += (2 × distance - 1);
    update interim_layout;
    update remaining_nodes;
end
```

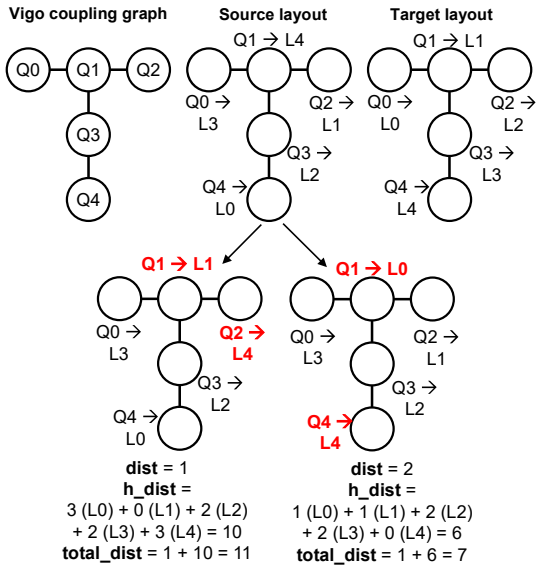

Fig. 5: An example of creating a SWAP layer using the Algorithm 1.

is {Q0 → L0, Q1 → L1, Q2 → L2, Q3 → L3, Q4 → L4}. The objective is to go from the source layout to the target layout using as few number of SWAPs as possible.

All logical qubits in S are out of position from T, and thus, remaining_nodes = {L0, L1, L2, L3, L4}.

The algorithm iterates over each qubit in the *remaining nodes* and computes the *shortest distance* (dist) from its current position to the target position. Besides the shortest distance, the algorithm also

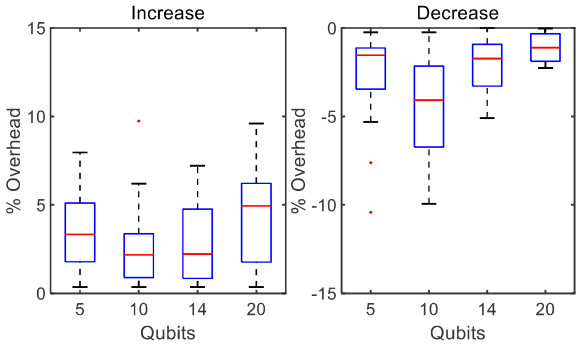

Fig. 6: Trend in increase and decrease overhead (%) for approach–1 with respect to the different sized circuits (like 5q, 10q, 14q and 20q).

computes an approximate heuristic distance (*h_dist*). When one qubit is moved, it changes the position of other qubits as well. Thus moving a qubit closer to its target position may push other qubits farther from their respective targets. The heuristic distance computes the aggregate distance of other qubits from their targets for an intended SWAP operation. The total distance is computed by adding the shortest distance (dist) and the heuristic distance (h_dist). After iterating over all remaining nodes, the algorithm picks the node with the lowest total distance for SWAP insertion and updates the interim layout.

Suppose, the algorithm selects L1 first. Its current position is Q2 and target position is Q1. Therefore, the shortest distance (dist) between the nodes is 1. Now, swapping Q1 (L4) and Q2 (L1) will move L1 to its target position (Q1) but in the process will move L4 farther from its target position (Q4). This is captured in the heuristic distance (h_dist). Therefore, the h_dist is 10, and total distance for moving L1 is $1 + 10 = 11$.

Consider, L4 (Q1) is selected in the next iteration. Moving it to the target location (Q4) will need swapping it with L0. *dist* for this swap is 3. The heuristic distance after this movement is 6. Thus, the total distance is $1+6 = 7$. As it is lower than previous min_dist (11), min_dist is updated to 7, candidate is updated to L4 (new move_from = Q1 and move_to = Q4).

The *for* loop will iterate over all the remaining nodes, and the node with the smallest total_dist will be the selected movement. From this exercise, L4 from the source layout is selected to move from Q1 to Q4. The simplest movement path for this is Q1 → Q3 → Q4. It requires 2 swaps to move L4: SWAP(Q1, Q3) and SWAP(Q3, Q4), and 1 additional SWAP(Q3, Q1) to move to L0 to Q1 (i.e., to L4's initial position). These SWAPs are added to the swap layer (SL), and the num_swaps is updated.

After this interim layout is updated to {Q0 → L3, Q1 → L0, Q2 → L1, Q3 → L2, Q4 → L4} and L4 is removed from the remaining_nodes. As the interim layout is not the same as T yet, the *while* loop will continue with the updated interim layout and remaining nodes.

V. Results and Discussions

This section gives an elaborate description of the experimental results and the consequent analysis with respect to overhead.

A. Simulation Setup

We use the open-source quantum software development kit from IBM (Qiskit) [21][2] for our simulations. The software runs on Ubuntu

[2]We do not claim the Qiskit, tKet, QuilC compilers to be untrusted. Qiskit compiler is used for demonstration and analysis purposes only.

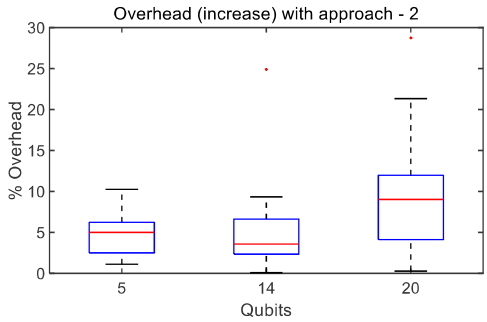

Fig. 7: Trend in overhead increase when SWAP layer is added between splits for p2v mapping.

20.04 virtual machine with 8 GB RAM on an Intel Core i7-9700K (3.60GHz) host (Windows 10). We use over 152 circuits containing different gate operations and different sizes of qubits- 5, 10, 14, and 20 qubits. We use the ibmq_vigo, ibmq_melbourne, and ibmq_almaden architectures for all the experiments performed [21].

B. Overhead analysis

1) Approach–1: We present the overhead of split compilation in terms of % difference of weighted gate counts between the original circuits and split combined circuits.

$$\%Overhead = \frac{W(split\ combined) - W(original\ circuit)}{W(original\ circuit)} \quad (1)$$

Fig. 6 shows distribution of these % overhead values for various circuit widths (5, 10, 14, and 20) and for 2 splits. We observe negative overhead i.e., weighted gate counts can drop in the split combined circuit for $\approx 47\%$ test circuits.

The remaining circuits show small increases in gate count. From the box-plots, we can observe that the majority of the circuits show a percentage overhead of within 5% with an average overhead of 3% across different circuit sizes. The highest overhead is $\approx 10\%$.

We also plot the overhead trend for a various number of splits ranging from 2 to 8 (Fig. 8). The trend shows the average overhead remains stable across a various number of splits.

2) Approach–2: For approach–2, we plot results from 5-, 14-, and 20-qubit circuits to force possible worst-case overheads on the 3 IBM architectures (Fig. 7). The SWAP layer will have more SWAP gates if, (i) there are many displaced qubits between the final layout and the initial layout, and (ii) connectivity between the physical qubits is sparse. These are manifested in the boxplot for 20-qubit circuits as they show the highest mean overhead (9.5%) and spread ($\sigma = 7.36$) (max $\approx 28\%$). Between using same final and initial layout (low overhead) and using completely independent layouts, a mixed approach can be adopted where a certain number of qubits in the final layout will be kept the same and the remaining qubits will be shuffled to get the initial layout for the next split. We re-run the compilation for 20-qubit circuits with the mixed approach where 10-qubits are shuffled and other 10-qubits are kept the same in the final layout. The results show that the mean, the standard deviation, and the max value of the overheads are 4.93%, 3.13%, and 12.7% respectively. This is a significant drop from the previous values. We show that even with 10 shuffled qubits it can lead to high reconstruction effort on the adversary's end.

3) Asymmetric split: We also perform overhead analysis for asymmetric splits. Equally split parts may give some clue to the adversary about which parts are related. Therefore, the original circuit can be split unequally where each part will have a different number of layers. The Table I shows statistics for several asymmetric splits (60%-40%, 40%-60%, 80%-20%, and 20%-80%). We present the result for the increase in % overhead as the decreased overhead is favorable. Asymmetric splits also show low overheads with average overhead ranging from $\approx 3\%$ to $\approx 4.5\%$ with a tight spread (standard deviation $\approx 2.6\%$ to $\approx 3.6\%$).

Splits	Mean	Std	Max	% ckts
60-40	3.7	3.6	13.22	41.77
40-60	3.4	3.02	12.72	43.42
80-20	3.2	2.67	12.52	44.74
20-80	4.54	3.37	17.48	44.77

TABLE I: Asymmetric splits statistics. The results show the asymmetric split incurs a low overhead.

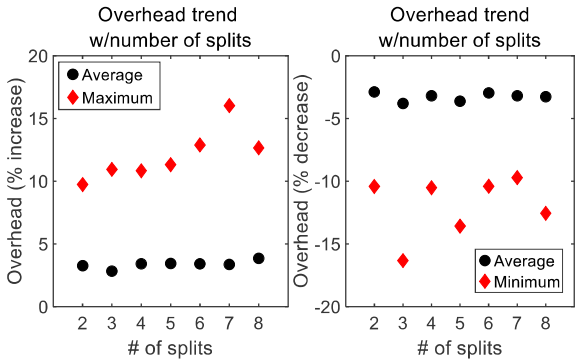

Fig. 8: Overhead trend with various number of splits. The average overhead remains stable over different number of splits.

C. On positive and negative overhead

From the overhead analysis, we observe that roughly 50% circuits show an increase in weighted gate count while approximately 50% circuits show a decrease. We identify that pass–4 (routing using SWAP) is the primary factor behind the increase or the decrease in gate counts accounting for more than 90% of the additional/removed gates.

Theoretically, considering the whole circuit should give the best (least) number of SWAPs. However, globally-optimized routing using exact methods like (ILP) [7] and satisfiability modulo theorem (SMT) solvers [22] scales poorly as the problem is NP-complete [23], [24]. Therefore, more scalable approaches resorted to heuristics methods [6], [8].

Heuristic algorithms also include a *look-ahead* capability which enables them to consider a circuit as a whole. However, the look-ahead capability in heuristic algorithms is approximate (to accelerate the search). Therefore, increasing the look-ahead window up to the whole circuit does not always guarantee the best heuristic condition. For example, Fig. 9 shows the number of SWAPs with varied look-ahead window sizes for a circuit. The results clearly show that increasing the look-ahead window size does not provide a monotonous trend and the window size of 1 resulted in the lowest overhead.

The above discussion establishes that with scalable heuristic routing algorithms, considering the whole circuit does not necessarily have an advantage over partial circuits. Besides, the heuristic nature of the routing algorithm also contributes to the bipolar overhead values.

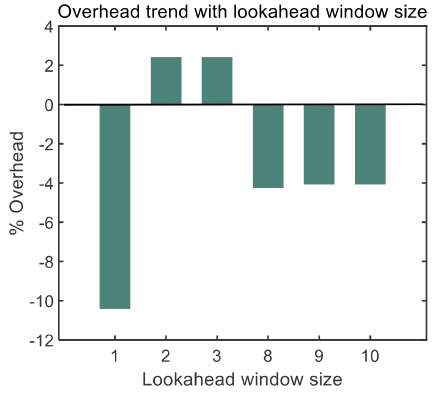

Fig. 9: Look-ahead trend in the SABRE [6] routing algorithm. An increased lookahead window size does not guarantee a lower number of SWAPs as it computes an approximate cost for better scalability. Window sizes 4 to 7 are omitted as they do not show any overhead.

D. Design knobs for reducing overhead

In the previous section, we presented overheads for a collection of circuits for certain compilation conditions. We explore few parameters to reduce the overhead for the worst corner cases e.g., routing window size, asymmetric split, and the number of shuffled qubits (for approach–2). The user can split-compile a circuit with few combinations of these parameters and select the combination that provides the least weighted gate counts. However, checking many options will increase the compilation cost for the user. Therefore, we prescribe the following sequence to reduce the overhead: change the number of shuffled qubits $\rightarrow$ increase window size at a step of 2 $\rightarrow$ check the asymmetric split starting with $20\% - 80\%$ (then, $40\% - 60\%$, $60\% - 40\%$, and $80\% - 20\%$).

VI. Security Analysis

For security analysis, we consider a single untrusted compiler and multiple untrusted compilers.

A. Scenario–1: Single untrusted compiler

In this case, the user sends the split circuits to the compilers at different times. If all the splits are sent to the untrusted compiler together, an adversary can try to correlate the parts and join them to reconstruct the full circuit. Suppose, approach–1 is followed for the split compilation i.e., the split–2 is sent for compilation with an initial layout same as the final layout of the split–1. In such a case, an adversary can trace back his/her compilation queue and find which circuit matches the layout. After finding the match, an adversary can join the circuits to get the full version. This search will take linear time, $O(k)$ where k is the number of n-qubit circuits in the compilation queue in a time window.

In approach–2, the split–1 final layout and the split–2 initial layouts are independent of each other. Therefore, any n-qubit circuit becomes a potential candidate. By adopting the approach–2, the complexity of circuit reconstruction can be increased. Besides, for an n-qubit circuit, there can be $n!$ permutations for the initial layout of split–2. Therefore, the number of options an adversary needs to check to reconstruct will be $k \times n!$. If there are more than 2 splits the options expands even more, $k \times \prod_{s=1}^{s-1} n!$ (s = number of splits). We present several numerical values for various choices of s, k and n in Table II:

# of Splits (s)	Queue size (k)	# of qubits (n)	$k \times \prod_{s=1}^{s-1} n!$
2	1	5	120
3	1	5	$14,400$
4	1	5	$1,728,000$
2	1	10	$3,628,800$
3	1	10	$\approx 1.3 \times 10^{13}$
2	1	14	$\approx 8.7 \times 10^{10}$
3	1	14	$\approx 7.6 \times 10^{21}$

TABLE II: The possible number of options an adversary needs to reconstruct a full circuit.

The numerical analysis shows that the number of possible options explodes for a higher number of qubits in circuits. For circuits with a smaller number of qubits such as, 5-qubit, splitting the circuit into more than 2 parts can increase the reconstruction effort for an adversary.

B. Scenario–2: Multiple compilers

Usage of multiple compilers makes the split compilation inherently secure since a single compiler only has partial information about the circuit at all times. An adversary can try reconstructing the missing layers by selecting gates from a design gate-set and placing them on different combinations of qubits. For that, the adversary needs three pieces of information: (i) a design gate-set (G) which the user used to design his/her circuit, (ii) number of missing layers (L) in the circuit and (iii) number of qubits in the circuit. Except for item–(iii), it is unlikely that information about missing layers and the specific design gate-set will be available to the adversary. Therefore, the split compilation with multiple compilers becomes inherently secure. However, we take a pessimistic stand (from the user perspective, optimistic for adversary) and assume the adversary has all 3 pieces of information. Using numerical examples we show that even for such a pessimistic case it introduces a prohibitively large effort on the adversary's end.

Example 6.1: Consider an adversary aims to reconstruct a 5-qubit circuit with a design gate set containing only 2-qubit {CX}. Besides, consider there can be only 1 gate in a layer. Therefore, there can be ${}^nP_r = {}^5P_2 = 20$ possibilities (n = number of qubits in the circuit, r = qubits in the selected gate) to create a single layer. If there are L missing layers, the number becomes $({}^nP_r)^L = 20^L$. For example, with $L = 5$, this value becomes 3.2×10^6. However, it is unlikely that the design gate set will contain only 1 gate. In theory, any unitary operation can be a quantum gate, and therefore, the size of the design gate set can be large. For an extended gate set G the number of choices becomes $(\sum_G {}^nP_r)^L$. Considering an optimistic case for the adversary with only 2 types of gates in the set {CX, CCX}, a 5-qubit circuit, and 5 missing layers, the number options becomes $\approx 7.7 \times 10^8$. It will become even worse with a typical larger design gate-set.

VII. Conclusion

Sensitive IPs contained in the quantum circuit such as algorithms, data, financial or other proprietary information, can be exposed to adversarial threats such as stealing, counterfeiting, or reverse engineering during compilation at an untrusted third party. This paper aims to secure the quantum circuits by splitting them and sending the subcircuits to single or multiple compilers. We analyzed 152 circuits on three IBM hardware architectures to evaluate the proposed idea. Our results indicated a modest overhead of 3%-6% on average.

REFERENCES

[1] H. Abraham *et al.*, "Qiskit: An open-source framework for quantum computing," 2019.

[2] R. S. Smith, E. C. Peterson, M. G. Skilbeck, and E. J. Davis, "An open-source, industrial-strength optimizing compiler for quantum programs," *Quantum Science and Technology*, vol. 5, no. 4, p. 044001, 2020.

[3] Z. Computing, "Orquestra," May 2021. [Online]. Available: https://www.zapatacomputing.com/orquestra/

[4] C. Q. Computing, "pytket," May 2021. [Online]. Available: https://cqcl.github.io/pytket/build/html/index.html

[5] A. Suresh, A. A. Saki, M. Alam, R. O. Topaloglu, and S. Ghosh, "A quantum circuit obfuscation methodology for security and privacy," *arXiv preprint arXiv:2104.05943*, 2021.

[6] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. Association for Computing Machinery, 2019, p. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[7] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh, "Muqut: Multi-constraint quantum circuit mapping on nisq computers: Invited paper," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[8] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[9] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1259–1264.

[10] M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, and M. Fritze, "Split-fabrication obfuscation: Metrics and techniques," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 7–12.

[11] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi, "Efficient and secure intellectual property (ip) design with split fabrication," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 13–18.

[12] Y. Wang, P. Chen, J. Hu, G. Li, and J. Rajendran, "The cat and mouse in split manufacturing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 805–817, 2018.

[13] A. Ash-Saki, M. Alam, and S. Ghosh, "Analysis of crosstalk in nisq devices and security implications in multi-programming regime," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: https://doi.org/10.1145/3370748.3406570

[14] N. Acharya and S. M. Saeed, "A lightweight approach to detect malicious/unexpected changes in the error rates of nisq computers," *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020.

[15] V. Saravanan and S. M. Saeed, "Decomposition-based watermarking of quantum circuits," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 73–78.

[16] K. Phalak, A. Ash-Saki, , M. Alam, R. O. Topaloglu, and S. Ghosh, "Quantum puf for security and trust in quantum computing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2021.

[17] A. A. Saki and S. Ghosh, "Qubit sensing: A new attack model for multi-programming quantum computing," 2021.

[18] S. M. Saeed, A. Zulehner, R. Wille, R. Drechsler, and R. Karri, "Reversible circuits: Ic/ip piracy attacks and countermeasures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2523–2535, 2019.

[19] N. Limaye, M. Yasin, and O. Sinanoglu, "Revisiting logic locking for reversible computing," *IEEE European Test Symposium (ETS)*, 2019.

[20] D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, and J. M. Gambetta, "Efficient z gates for quantum computing," *Phys. Rev. A*, vol. 96, p. 022330, Aug 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.96.022330

[21] A. Cross, "The ibm q experience and qiskit open-source quantum computing software," in *APS Meeting Abstracts*, 2018.

[22] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. Association for Computing Machinery, 2019, p. 1015–1029. [Online]. Available: https://doi.org/10.1145/3297858.3304075

[23] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, ser. CGO 2018. Association for Computing Machinery, 2018, p. 113–125. [Online]. Available: https://doi.org/10.1145/3168822

[24] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. a. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3360546