Summary of the current state of local mixing

Nicholas Ho
phantom.zone

Indistinguishable obfuscation has been a topic of discussion in cryptography ever since Barak et al in 2001 first introduced the concept. Simply put, indistinguishable obfuscation is a form of obfuscation where different circuits of equivalent functionality will have their obfuscated circuits be indistinguishable to each other. In notation terms, given C_1 and C_2, we have that $O(C_1)$ is indistinguishable from $O(C_2)$. This seemingly simple definition actually gives us many cryptographic primitives. Namely, we can get something as simple as public-key cryptography, trapdoor permutations, general secure multiparty computation, etc. When used with lossy or rerandomizable encryption, we can even get things like fully homomorphic encryption and more.

Constructing indistinguishability obfuscators, though, is quite hard. We already know some constructions, however, are complex or have high overhead. Some of these tools include evasive LWE, multilinear maps, and learning parity with noise. These tools are highly structured and so we get the construction with the high complexity. The high overhead then means that we can't practically use iO in any meaningful way.

We then get to the main question: "Can we achieve iO through the use of only first principles". In other words, can we achieve iO through only simple methods? This is the motivation of local mixing. In fact, local mixing provides something stronger than iO, but we will not go into deep detail here.

Let us take an example. Suppose we have a circuit with input X and output C(X). Within the circuit, we can select a random "piece" and make a small replacement. We can of course do this many times, until we seemingly have replaced the entire circuit. After we have replaced enough, we would have a new circuit with the same functionality but has been randomized with all of these pieces.

More intuitively, the basic premise of local mixing is as follows: first we select a random subcircuit, then we select a random subcircuit with equal functionality, and then we make the replacement. Of course, there are many variables to be mindful of as we make these selections, but more on this later. In order to maintain our ideal of "simplicity", we actually want these changes to be small, random, but equal in functionality to what it is replacing. Thus, local mixing is just making many small perturbations of equal functionality in the hopes that after many iterations, a global effect would have occurred.

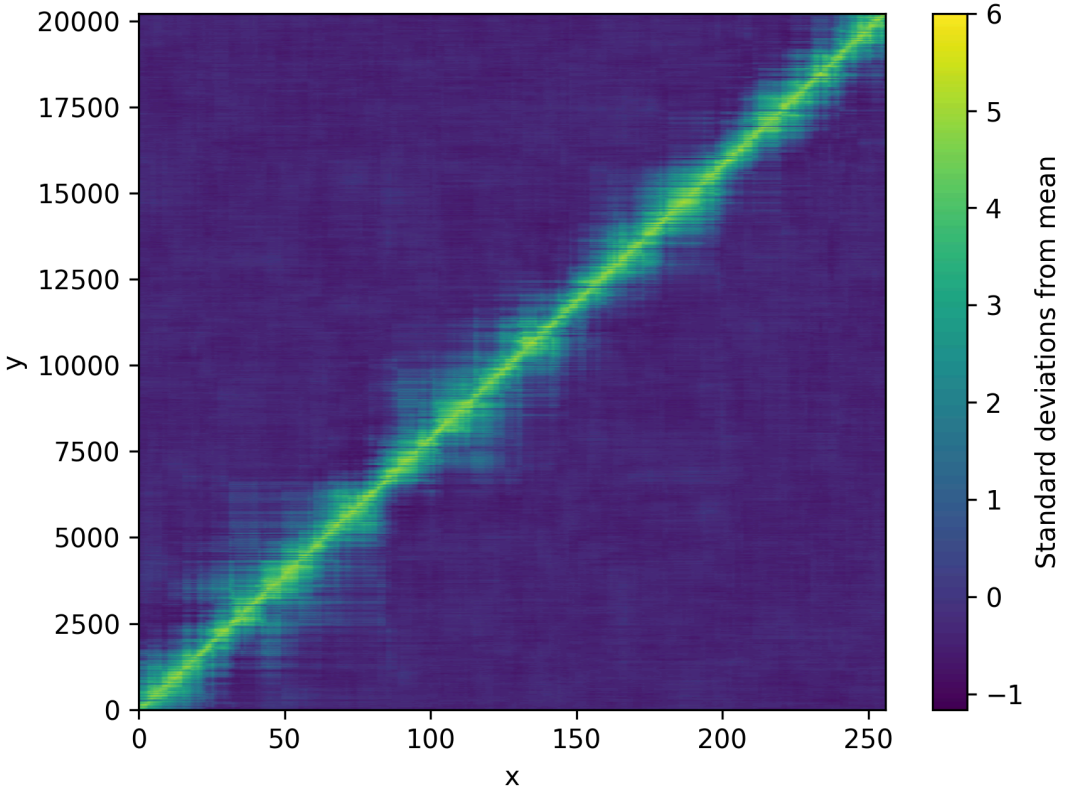
Before we talk about how we sample circuits, it would be helpful to take a small look at our circuits themselves. We are working with reversible circuits. This does not pose a problem, thankfully, as any general circuit can be represented by a reversible circuit. Thus, if we can do local mixing on reversible circuits, then we can do local mixing on the general circuit. In addition, we only use gate r57. This may seem strange as well, since using many other gates can provide additional freedom to our circuits, however, we find that this simplifies our circuits vastly, which is important to our considerations of local mixing.

One thing that will be important to consider, is how to sample subcircuits. Of course, there are two easy ways to do this. First, we can sample contiguous subcircuits and then non-contiguous subcircuits. However, we must be careful with how we do replacements on non-contiguous subcircuits. This is where skeleton graphs of a circuit become useful. Suppose we have some gates for circuit A. Then we can create a skeleton graph for circuit A as follows: For each gate, let there be a node. For each collision between gates, we get a directed wire from gate i to gate j with the additional condition that $i < j$. This gives us a skeleton graph of circuit A. We can then take a convex subgraph of the skeleton graph, which then gives us a convex subcircuit that we can use to make replacements. Since this is a convex subcircuit, then we know that we can reorder the non-contiguous subcircuit in

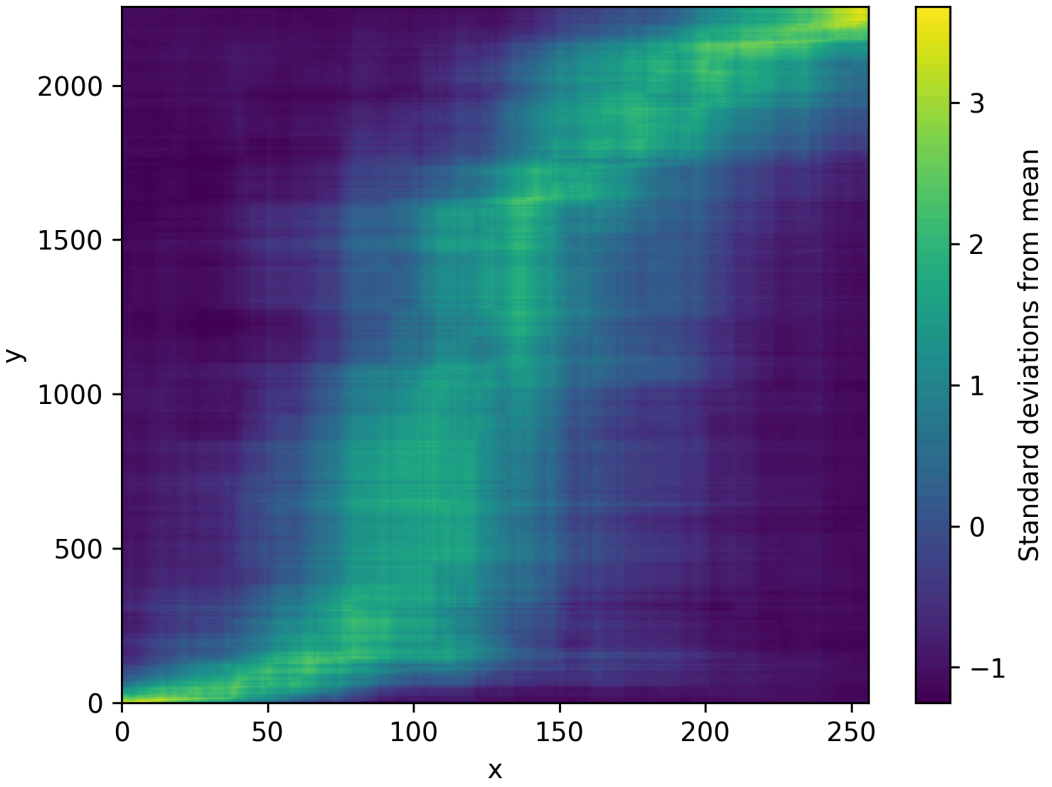
the larger circuit in such a way that the non-contiguous subcircuit becomes contiguous. We also then see that gates on a shared topological level of the skeleton graph do not collide.

So we have our circuit and a seemingly randomized version of it. A natural question that will arise is how we can analyze such a circuit. One simple way to do it is to look at the heatmap of the circuit. Let us have circuit A and circuit O(A). If we are curious about how random O(A) actually looks from the original circuit A, then we can take the heatmap between them. The heatmap will simply compute the hamming distance between the two states. Which states? Well we just take the state of both circuits after each gate. For instance, if we have circuit A and circuit B, each with 10 gates. Then we would compare the hamming distance of circuit A after 1 gate with circuit B after 1 gate, circuit A after 2 gates with circuit B after 1 gate, …, circuit A after 10 gates and circuit B after 10 gates. This will give us a 2D heatmap of circuit A and circuit B. If circuits are identical, then we would expect a diagonal along y = x to appear as the hamming distance would be zero. On the other hand, for completely random circuits of the same functionality, we would expect only the bottom left and top right corners to show low hamming distance, with everything else just being an average amount of hamming distance.

Let us take a look at our first method. This method can be called the two phase method. Simply, this method first has an inflationary phase, where convex subcircuits are sampled and replaced with “wider” (more gates) circuits. In order to spread these local perturbations globally, we then move onto the kneading stage, where we do the same, but the replacements are now made with equal width circuits. In this method, we find that the heatmap gets blurred more and more as we do more and more kneading stages.



This was one of our early heatmaps on a ~250 gate circuit on 64 wires, inflated to ~20,000 gates and then with ~1,000 rounds of kneading. We see that along the y = x, there is clear correlation between the circuits. In other words, kneading did very little.



If we up our number of rounds to the millions, we see that the line “blurs” quite a bit more. Of course, we can still tell that there is a line of sorts. Ideally, we would see high correlation in the bottom left corner and the top right corners. So while this is better than before, it is not the most ideal.

However, we felt that this may be good enough in a practical sense. It is with this motivation, that a bounty was released. The original circuit for the bounty was 64 wires and 1014 gates. This was then inflated, and kneaded to 12111 gates. Of course, a sanity check was made to show that indeed, the heatmap did look random. However, as many know, this bounty did not last long.

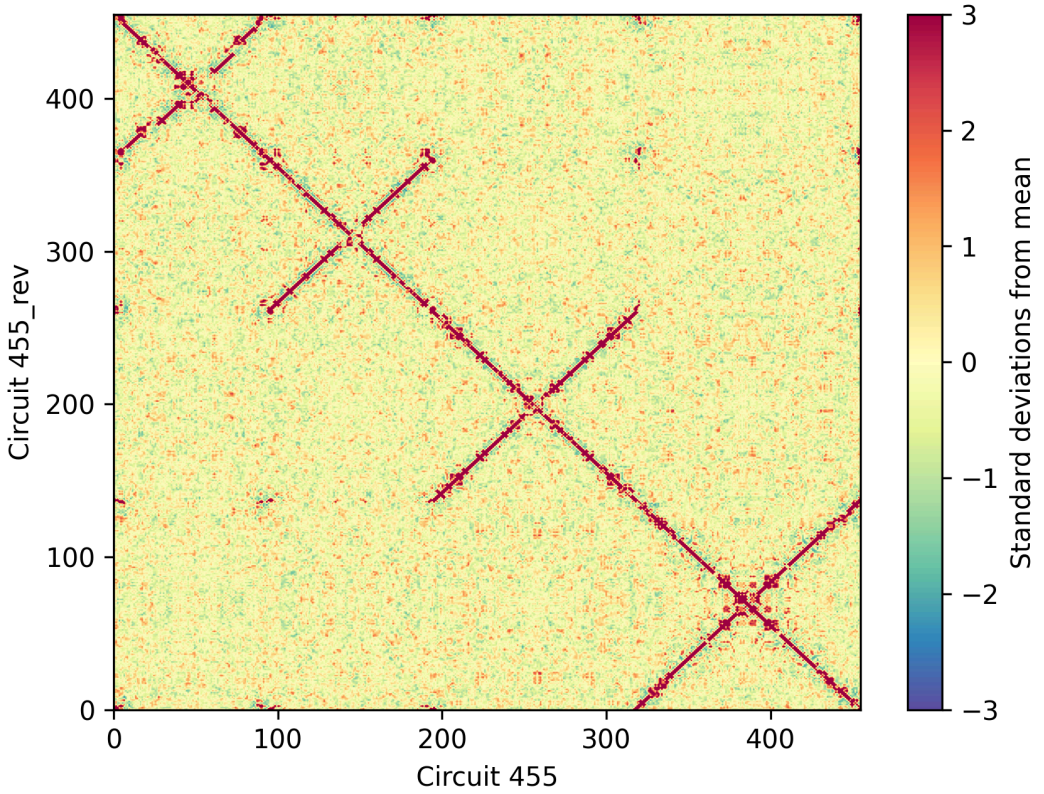
The way that the bounty was broken was via compression. We know that we can make a replacement by increasing the number of gates. So there is nothing stopping anyone from merely using the same method, but instead attempting only to decrease the total number of circuits. We will return to this shortly.

How can we sample subcircuit replacements? One easy way, if we are working on a small number of wires, is to just keep generating random circuits until the functionality is the same. This is what was used in the method above. However, instead of sampling randomly, what if we don’t want replacements with little meaning? This means we need a sense of a circuit canonicalization. Another way to make replacements is via a rainbow table, where we preprocess every single possible replacement and store it in memory. However, there are a lot of circuits… For just 5 wires and 5 gates, we already have over 770,000,000 different circuits. So for even more reasons, we need some type of circuit equality/isomorphism. One way to do this is to merely use the lexicographic ordering of its non-colliding gates. However, we can also try to canonicalize the permutation that a circuit computes. This can be done by taking the smallest lexicographic ordering with respect to all possible bit shuffling. By considering these things, as well as not caring about circuits with adjacent gates (since

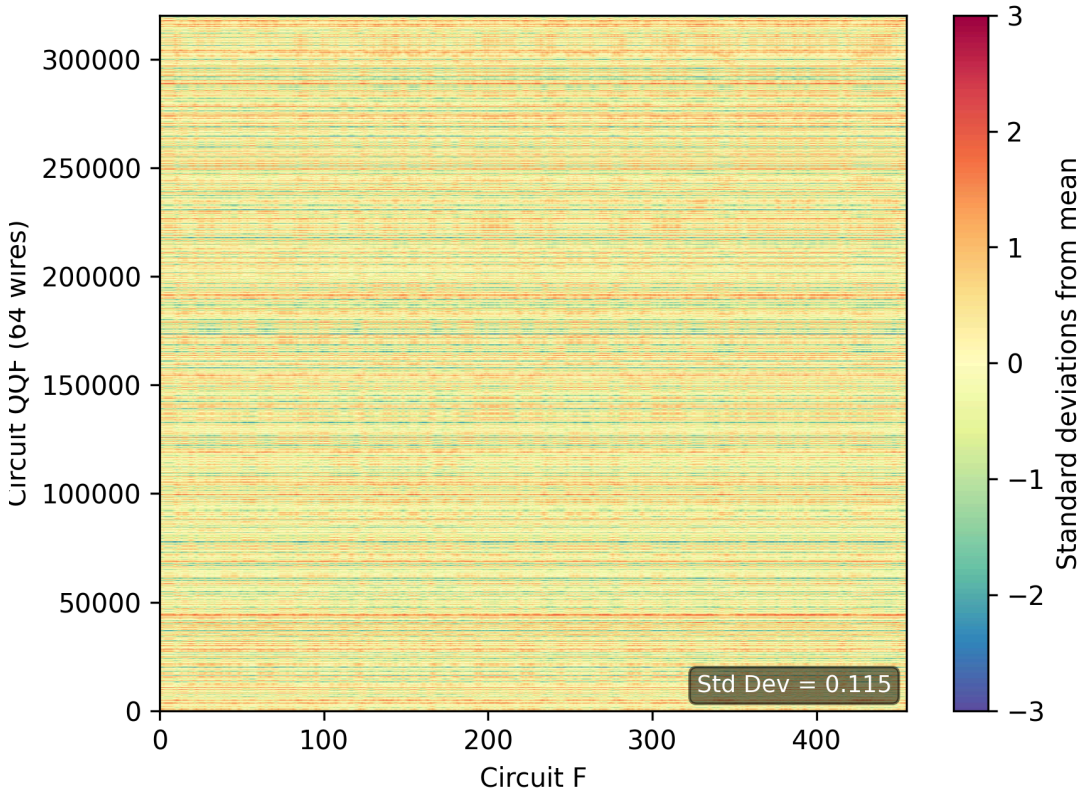
this yields an identity) and inverses (since we can merely reverse a circuit to get a reverse permutation) then our 770,000,000 falls down to below 3,000,000.

We can then use our rainbow table for compression. Instead of having two phases, an initial inflation followed by kneading (which doesn't change the size of the circuit), we will inflate and compress repeatedly. In other words, we will inflate, then compress, then inflate, then compress… for some number of rounds, whereas our original method would inflate to a certain size and then stay in the kneading phase.

This brings us to our next method: the asymmetric butterfly method. Simply, this is just a method where we insert in between each gate an identity, and then separate everything into blocks of the form $R^*_{i-1}\, g_i R_i$ and then compress them all. We then merge the blocks together and continue compression. The hope is that these random blocks will compress enough so that when we merge, we don't just recombine R*R to get a stupid identity. However, heatmaps showed that there was little change. One belief of a reason for this, is because each block is completely random. In addition, we know that compression is hard to work on on completely random blocks, but could be easy on identities. Of course, this raises the problem of generating large incompressible identities, which is actually a very important problem for us, but we will not go into greater detail.

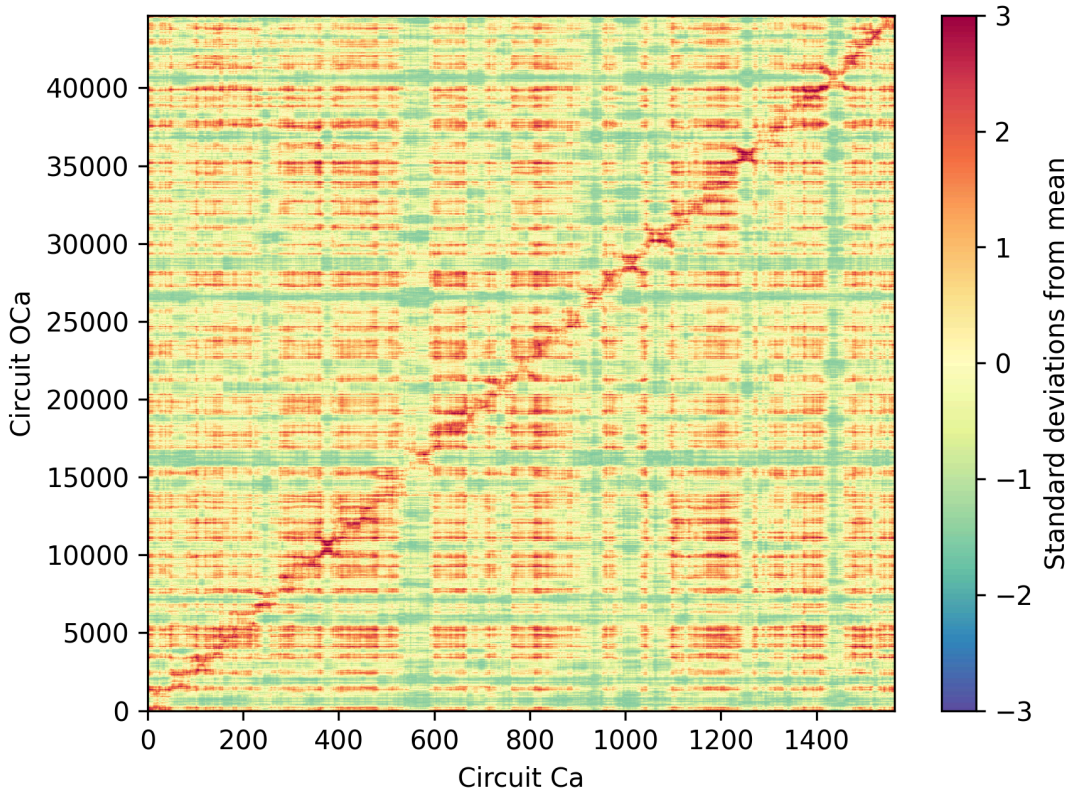


We shall start with the 455 gate identity on 5 wires above. The y-axis is flipped, but of little importance to our observations here. We will only take note that this identity has four "X" shapes.



There are two main concerns here. First, we can still see the four “X”s. Of course, we only did this on two rounds of asymmetric butterfly. So what if we did many more rounds? Recall that we were able to do millions of rounds of kneading since we never had to grow the circuit in any way. Well the problem that arises here is that the butterfly method causes much greater blow up in the number of gates. In just two rounds of asymmetric butterfly, we have almost a 1000x blowup in the number of gates. This greatly limits our ability to do many rounds.

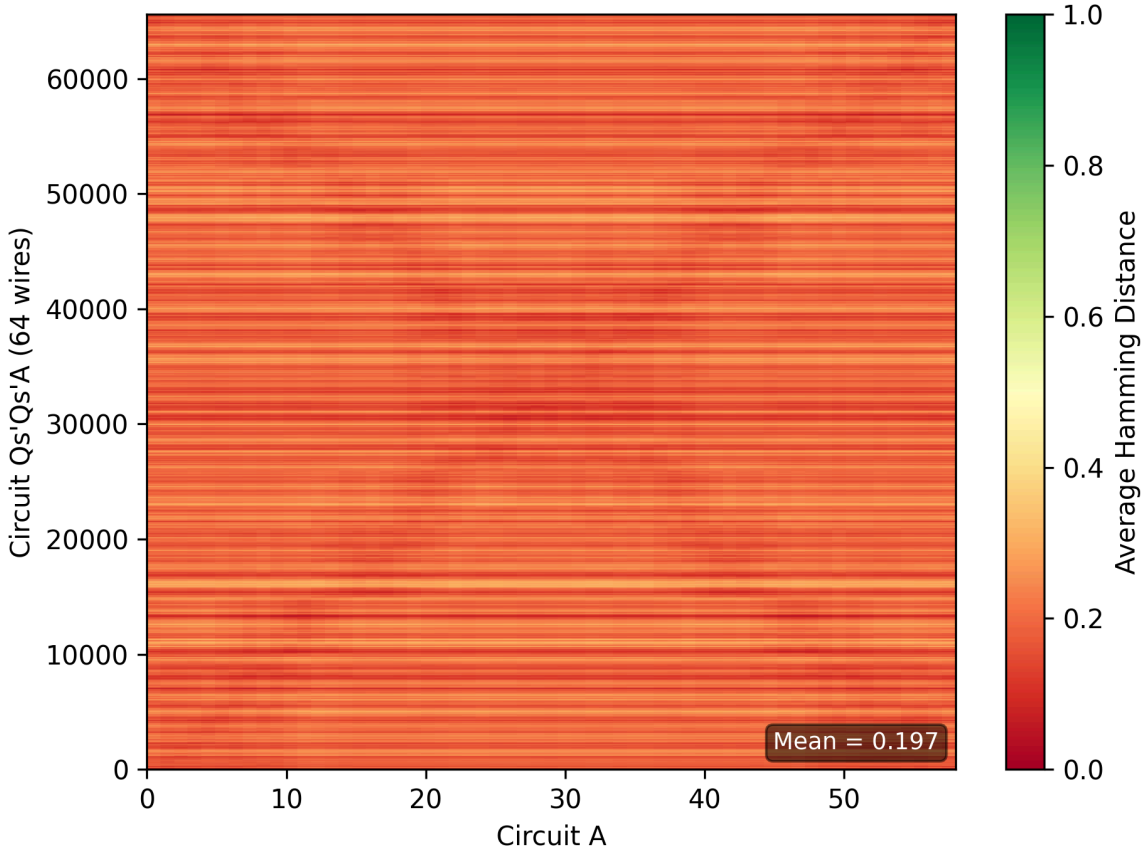
So what if we want to do the same as above, but want our blocks to be more compressible? That is where the last method we will talk about here comes in: the butterfly method. This is just the same as above, but the R*R is now the same across all gates. The hope here is that we purposefully inject structure into our blocks to force them to compress more.



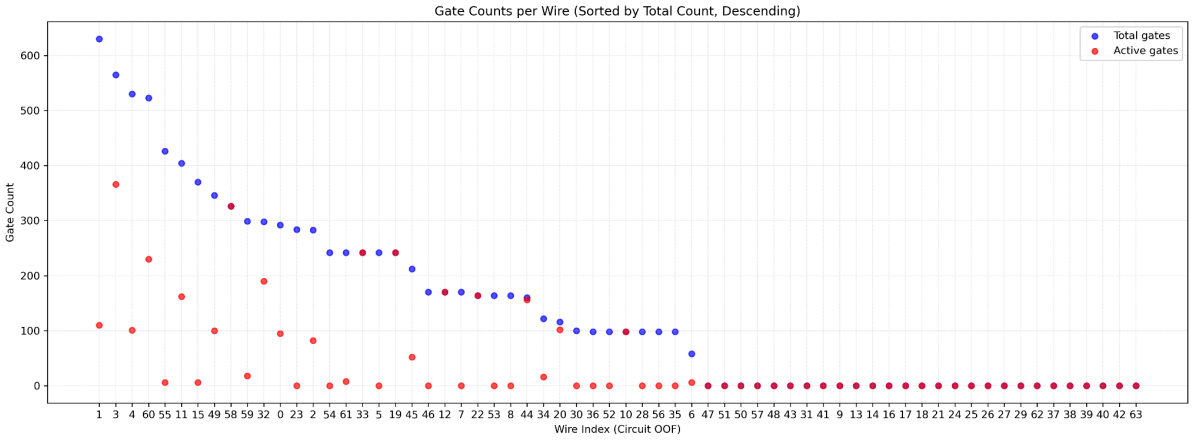
We not that Ca is just one round of symmetric butterfly on a simple, 22 gate identity. Thus, OCa is two rounds of butterfly on that same identity. We can of course take note on the blowup in the number of gates, but maybe more interesting is the heatmap itself. We see the red diagonal, which shows that the gates didn't move around that much, but we see a lot of red lines, horizontal and vertical. This means that our heatmaps can detect that we are only inserting identities. As another way to think of this, this means that our blocks still aren't compressing enough to hide the the identities!

How did compression do on these two methods? On a weak compressor, we find that the circuits don't mix that much and that the number of gates blows up. In method 1, we could run kneading millions of times. However, for these versions of butterflies, it becomes hard to keep them from blowing up in the number of gates.

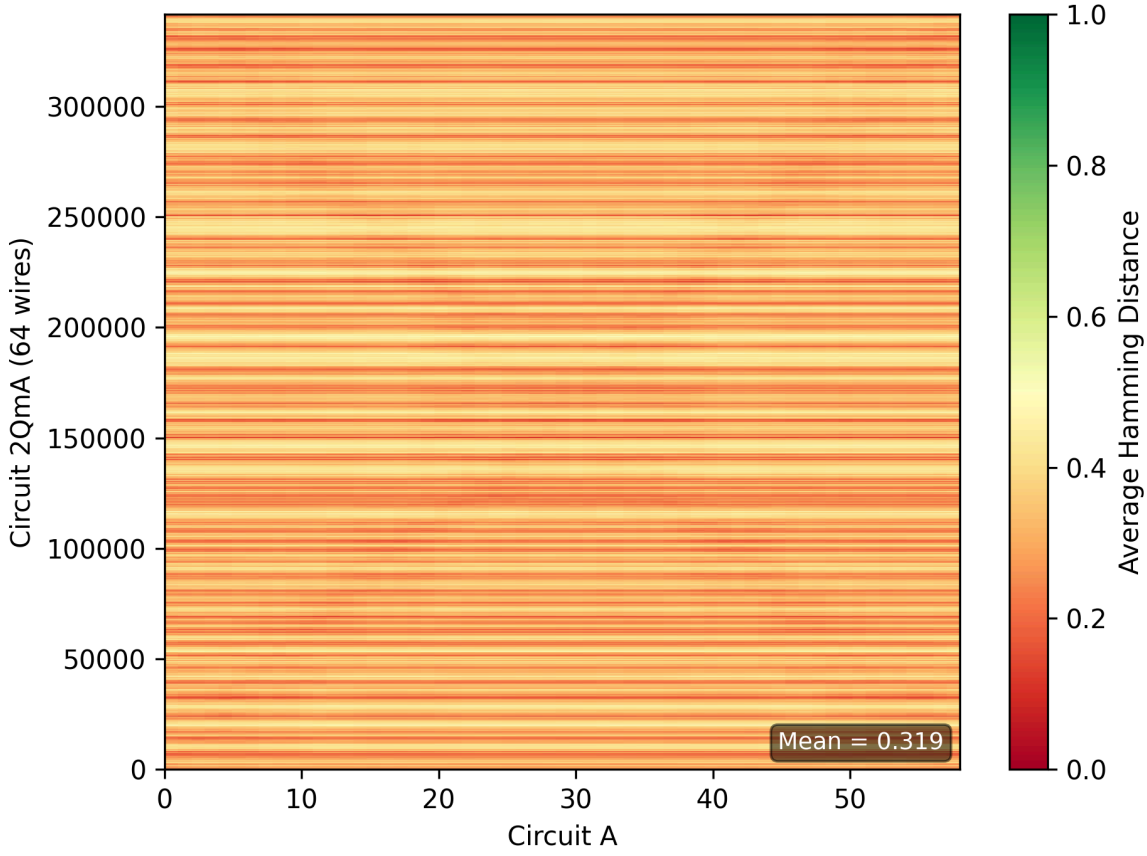
Another question is then "How long do the wings of each butterfly in each block need to be". Well, to answer this question, we took heatmaps of varying obfuscations with varying butterfly wing lengths. For the most random, ~150 and ~250 gates is quite large enough. However, this also comes with a cost: the number of gates explodes astronomically from 58 gates to 1,000,000 gates. Let's take a look at this a little more closely.

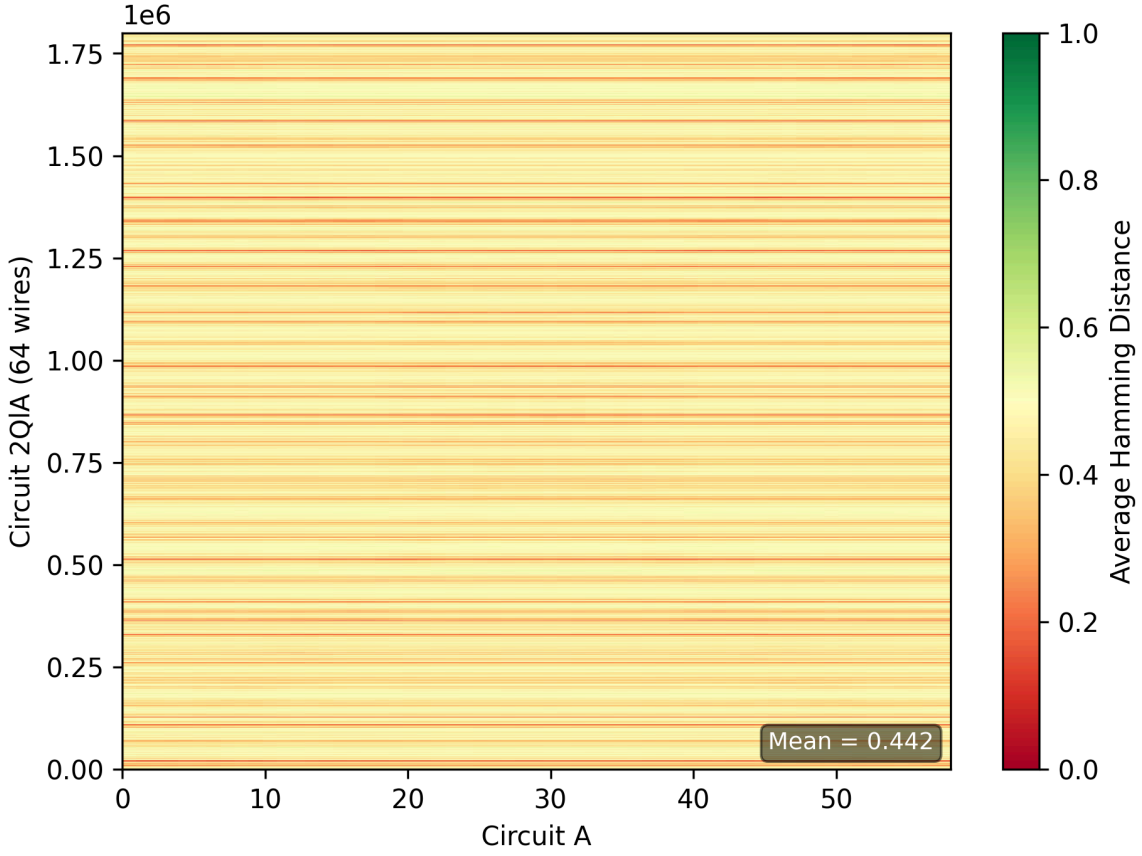


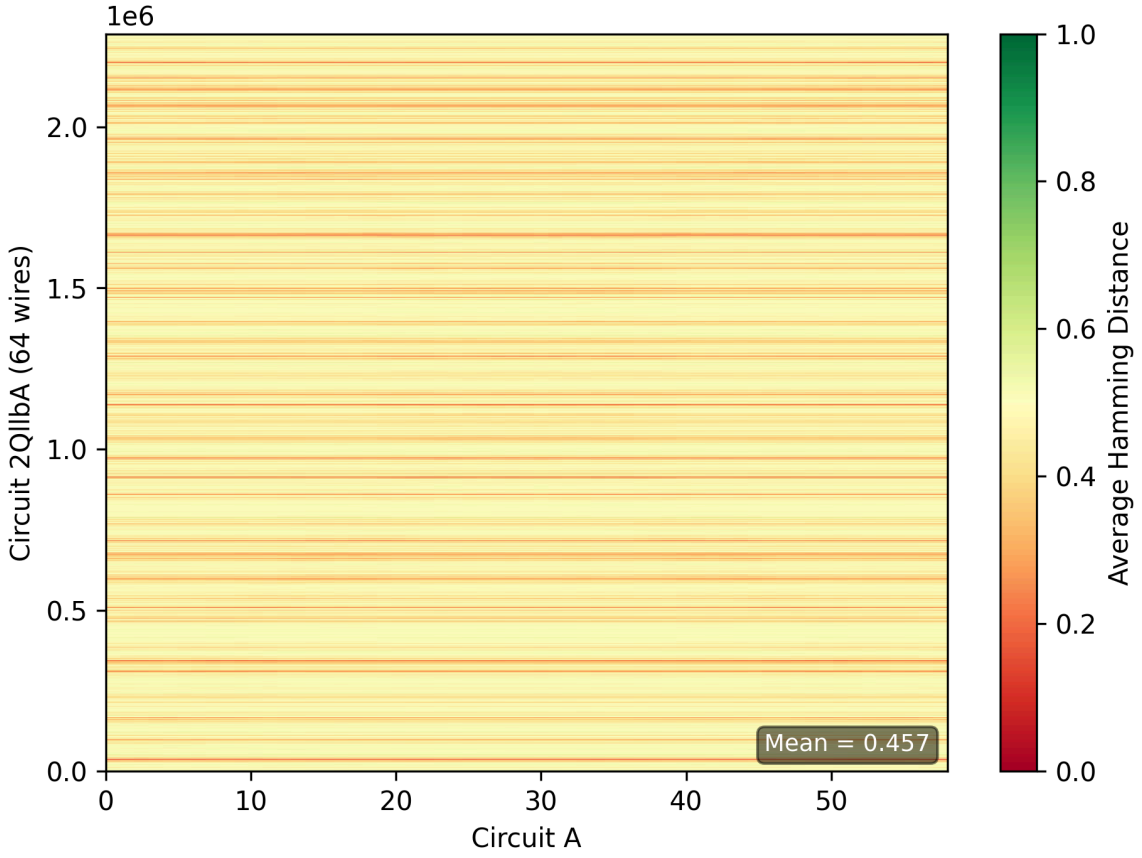
This is a test run with two rounds of asymmetric butterfly with wings of length about 10-15 gates long. We see that the circuit, while there is still a blowup in the number gates, we see that there is not much change at all. So we will want to make our wings longer. Before moving to longer wings, we can actually see why the heatmap looks like this.



From this, we see that the gates are still dominated on the first 5 wires, which circuit A completely lies on. On the other hand, despite the butterfly rounds, we have many wires which remain untouched. Ideally, this graph should be uniform and without “clumping” (vaguely, no wire should have most of its associated gates clumped together).


Circuit 2QmA (64 wires)
300000
250000
200000
150000
100000
50000
0
0
10
20
30
40
50
Circuit A
Mean = 0.319
1.0
0.8
0.6
0.4
0.2
0.0
Average Hamming Distance

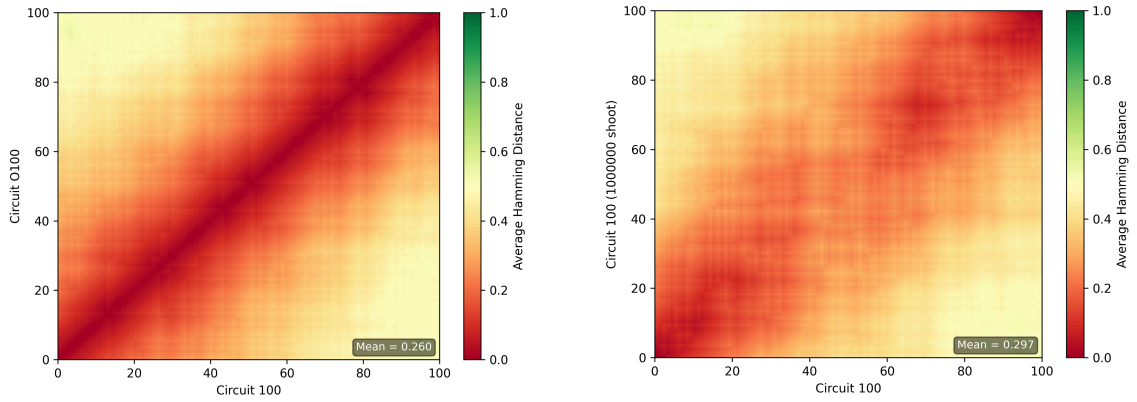

1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Circuit 2QIA (64 wires)
Mean = 0.442
0
10
20
30
40
50
Circuit A
1.0
0.8
0.6
0.4
0.2
0.0
Average Hamming Distance


1e6
2.0
1.5
1.0
0.5
0.0
Circuit 2QIIbA (64 wires)
Mean = 0.457
0
10
20
30
40
50
Circuit A
1.0
0.8
0.6
0.4
0.2
0.0
Average Hamming Distance

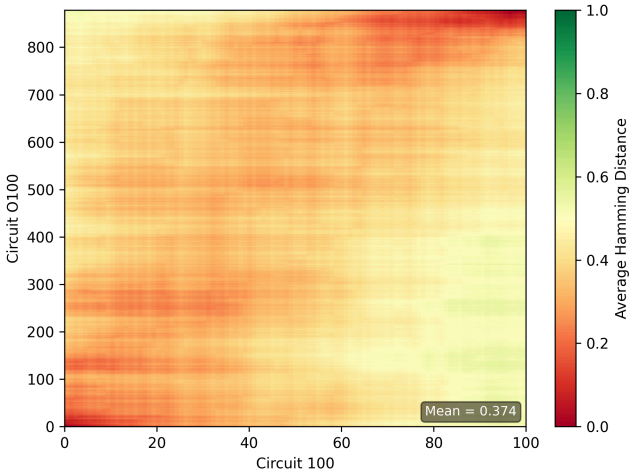
In the above heatmaps, the wings are about 60, 150, and 250 gates, respectively. We note that indeed, we get a lot of randomness from ~150 gate wings. For 250 gate wings, we indeed get more randomness, but there is a large blowup in the number of gates. We keep returning to this problem that we have too many gates, which stops us from doing many rounds. We know that doing many rounds of kneading was necessary to get a blurry heatmap, so we will need to compress more.

Unfortunately, if we just try to compress everything, then indeed, we would have compressed it all down. However, there are times where these butterflies create incompressible circuits.

Most of the results so far have been negative, but the fact that we can create incompressible circuits is quite fulfilling. Our newest idea, which seems to have a lot of promise, is to stack blurring methods. In this method, we will make many replacements of pairs of gates. The idea is that the seam between the two gates that we replace is completely removed. However, we can then also randomly swap non-colliding gates around. This would then blur the heatmaps (if we aren't canonicalizing beforehand). Of course, any adversary can undo this random swapping via circuit canonicalization. Then we can fix the swaps via the butterfly method, which then also makes our proposition incompressible. We can repeat, and then get a new obfuscated circuit obtained by blurring a circuit again and again. To ascertain whether this is really effective or not, it would be good to see how well shooting of gates actually does.



Indeed, we see that the shooting of gates indeed contributes some blurring to the heatmap. For now, we have not yet tested intensively how well we can "lock in" this blurring via the butterfly, nor do we have a great understanding of when to stop compression, but these are promising paths for us to follow. However, we can show one result.



This heatmap is the result of one symmetric butterfly on the 100 gate random circuit on 64 wires. We note that after the butterfly phase, and before the compression part, this became around 30,000 gates. In addition, it will eventually completely compress (we have already seen the heatmap for the circuit 100 on itself). Thus, despite being so close to completely compressing down, we still see a great deal of blurring that has not yet turned into the y=x line we see in the heatmap of circuit 100 vs itself. Thus, we do see that the butterfly method is effective in "locking" in this blurring. The only thing we need to ensure is that the circuit becomes incompressible.

We end this summary of recent results with some other considerations. For the methods we have above, this includes thinking about how much we need to compress. It is clear that we can't compress too little or else our gates blow up too much, and that we can't just compress them too much, because then the rounds of butterflies don't do anything. There may be many more methods as well, as well as more attacks. Our hope is to follow our optimistic method above and then release a bounty. This achieves two goals. The first goal is that it can show us new possible attacks on the method. At the moment, we can only think of heatmap and compression attacks. The second goal is to hope that that bounty never gets beat. While this wouldn't necessarily mean we have created practical iO, this would give us a huge step forward in the local mixing approach, with the hope that new people will explore this new path towards practical iO.