The algorithm works as follows: to compute $F(z)$, first, choose an integer $r$ uniformly at random from 0 to $n - 1$. Then, output $T(r) + T((z - r) \mod n) \mod m$, where $T$ is the "corrupted" version of $F$ that the adversary has tampered with.

The output of this algorithm will be correct with probability at least $3/5$. Observe that so long as neither $r$ nor $r - z \mod n$ are tampered with, then the output is $F(r) + F((z - r) \mod n) \mod m$, which from the given property of $F$ equals $F(r + z - r \mod n) = F(z)$.

Next, observe that at most $1/5$ of the table entries are tampered with. Since $r$ is chosen uniformly at random, that means that there is at most a $1/5$ chance that $T(r) \neq F(r)$, and likewise at most a $1/5$ chance that $T(r - z \mod n) \neq F(r - z \mod n)$. Even if these two events are made by the adversary to be completely disjoint, that would mean at most a $2/5$ chance that the output is not equal to $F(z)$. Thus, the algorithm is correct with probability greater than $1/2$ (since $3/5$ is more than $1/2$).

If my algorithm can be run three times, then I would then always select whichever output was returned at least twice, which against a perfect adversary there would always be (as the tampered values could be set so that on a specific input, all the tampered values cause the same wrong output). This would thus only increase the lower bound to the odds of succeeding in at least two out of three $3/5$ 'flips', which is $54/125 + 27/125 = 81/125$, or .648. This is admittedly not much better than just running it once, but the odds of a correct result would increase exponentially as the number of trials did.
Suppose, for the sake of contradiction, that there could be more than \( \frac{n(n-1)}{2} \) distinct minimum cut sets. If that were so, then clearly no algorithm could possibly exist that could choose each one of the sets with probability greater than \( \frac{2}{n(n-1)} \), because (since the algorithm obviously can only select one cut at a time) this would cause the total probability to exceed 1. Yet the analysis of the algorithm, and the lower bound on its probability of correctness, does not depend on the uniqueness of the cut—it holds true for any arbitrary min cut set. As it is impossible for more than \( \frac{n(n-1)}{2} \) cut sets to each have a probability greater than \( \frac{2}{n(n-1)} \) of being chosen by the algorithm, it follows that the initial assumption is incorrect, and by contradiction it is impossible for there to be more than \( \frac{n(n-1)}{2} \) distinct min cut sets in any graph.
Claim: At every point, after \( n \) items have been seen, all items seen have \( 1/n \) probability of being the "stored" item.

Proof by induction.

Base case: the first item has \( 1/1 \) chance of being the stored item.

Inductive step: Suppose there have been \( n \) items seen, and all items seen thus far have probability \( 1/n \) of being the "stored" item. We must show that after seeing another item, all items seen have \( 1/(n+1) \) probability of being the "stored" item. Upon seeing the next item, the algorithm sets it as the stored item with probability \( 1/(n+1) \). Thus, the new item has the correct probability of being the stored item. Additionally, any given previous item had a \( 1/n \) chance of being the stored item before this new item, and there is a \( n/(n+1) \) chance that the new item was NOT stored. Since the outcome of this step of the algorithm is independent of all previous steps, the chance of any particular previous item now being the stored item is \( 1 \cdot n/(n+1) = 1/(n+1) \). Thus all items have probability \( 1/(n+1) \) of being the currently stored item. This completes the inductive step, and thus by induction the claim is true.
To find the expected number of swaps, we must count the total number of swaps across all permutations, and divide by $n!$. Across all $n!$ possible permutations, observe that the number of swaps required for BubbleSort in each permutation can be determined by the number of swaps required for the same permutation without the largest element present, plus the number of swaps needed to put the largest element at the end of the list. Therefore, the total number of permutations needed for a size $n$ list is equal to the total number of possible permutations of size $n - 1$, plus the total number of swaps introduced by the largest element. Since the largest element has an equal number of permutations where it needs to be swapped each of $0$, $1$, ..., $n - 1$ times, each of these occurs for every possible permutation of $n - 1$ elements, and there are $n$ possible positions for the largest element to be in, the expected number of swaps for size $n$ permutations can be expressed with the following recursion:

$$T_n = \frac{n(n-1)(n-1)! + nT_{n-1}}{n!}$$

$$= \frac{n-1}{2} + \frac{T_{n-1}}{(n-1)!}$$

$$= \frac{n-1}{2} + \frac{n-2}{2} + \frac{T_{n-2}}{(n-2)!}$$

$$= \sum_{i=1}^{n} \frac{n-i}{2}$$

$$= \frac{n(n-1)}{4}$$