Near-Guaranteed Delivery in Garbled Circuits

Mikhail Svetlitskiy

November 2025

Abstract

We propose a near-guaranteed delivery protocol for two-party computation using garbled circuits. The protocol transforms the GC output into SPDZ-sharing with MACs, enabling sequential bitwise reveal with correctness checks. This ensures that even if one party behaves dishonestly, its computational advantage is limited to at most doubling the effort needed to obtain the result. The method works in standard 2PC settings without additional GC rounds and formalizes the steps for secure SPDZ conversion and verification.

Contents

1 Introduction

Modern secure computation protocols allow multiple parties to jointly compute functions over their private data without revealing this data to each other. One of the most well-known approaches is the *garbled circuit* (GC) proposed by Yao [1]. In the two-party computation model (*2-party computation*, 2PC), one party, Alice, acts as the *garbler*, creating the encrypted circuit, while the other party, Bob, is the *evaluator*, computing the output based on the garbled circuit.

Despite the widespread adoption of GC, certain issues arise related to the guarantee of obtaining a correct result. In the classical approach, one party receives the computation result first, which creates a potential for attacks, such as publishing false values or delaying the disclosure of the result. In the context of critical applications (financial computations, distributed voting, protocols with time-sensitive responses), such situations are unacceptable.

Moreover, the classical result by Fischer, Lynch, and Paterson [2] demonstrates that in distributed systems it is impossible to achieve guaranteed delivery in the presence of even a single unreliable process.

This work considers an algorithm for *near-guaranteed delivery* of the GC result, which provides:

- protection against intentional cheating by any party;
- the ability for both parties to correctly obtain the result even under partial malicious behavior;
- limitation of the adversary's computational advantage to at most a twofold speedup compared to the honest party;

The algorithm consists of two logical parts:

1. Modification of the original garbled circuit for subsequent transition to SPDZ-sharing. This allows converting the garbled output into a form suitable for stepwise disclosure and correctness verification.
2. A procedure for alternating disclosure of Alice's and Bob's shares, including MAC checks for verifying the integrity of each bit. This procedure ensures near-guaranteed delivery even under partial malicious behavior.

2 Transition from GC to SPDZ-sharing

Hereafter, we will abbreviate *garbled circuit* as *GC*. We assume that Alice acts as the *garbler*, and Bob as the *evaluator*. We also assume that upon completion, the original GC outputs a *clear* (i.e., unencrypted) result, and Bob receives this result first.

For correct subsequent value disclosure in the SPDZ phase, a series of *MAC checks* is required. We will use MAC checks over bits, i.e., over the field $\mathbb{F}_2$, and apply the method described in [3], with some simplifications in the disclosure procedure due to the fact that we are in a two-party computation (2PC) model rather than general MPC.

How exactly do we want to modify the original GC? First, we want to return the *encrypted* output of the original GC, augmented with several zeros that will later serve as a signaling mechanism to indicate that decryption was performed correctly. In addition, the modified GC should return only Bob's *share* (his part) of the encrypted output, as well as his part of the MAC checks. Alice's shares are known to her in advance. To achieve this, a few additional inputs must be added.

2.1 Structure of inputs and computations within the modified GC

Let us fix a security parameter s. The larger it is, the more reliable the MAC checks are.

Alice's inputs. For each bit (assume there are n) of the original GC output, Alice randomly generates:

- a bit string of length $s + 1$, which will be her output share $[enc(x)]$;
- a bit string of length s, which will be her MAC key share $[a]$;
- a bit string of length $s + 1$, which will be her output-MAC share $[xa]$.

All these values are provided to the modified GC.

Bob's inputs. For each bit of the original output, Bob randomly generates:

- a bit string of length s, which serves as his MAC key share $[a]$;
- a bit string of length s, representing the higher bits of his output share $[x]$ (the lower bits will be computed within the circuit).

The remaining necessary shares for Bob are obtained directly from the execution of the GC.

Shared encryption key. Alice and Bob agree on a symmetric encryption key known to both parties. This key is also provided to the circuit (for simplicity, assume Alice does this).

2.2 Computations within the modified GC.

At the beginning of execution, the modified GC fully evaluates the original GC and obtains its output x. The following steps are then performed:

1. The value x is padded with a specified number of zeros (indicator of correct decryption later) and encrypted using the agreed key, resulting in $enc(x)$.

2. SPDZ-sharing of the value $enc(x)$ is performed. For this, one least significant bit from each of Alice's bit strings for her share [enc(x)] is taken and XORed with $enc(x)$. The resulting bits form the least significant bits of Bob's share of $enc(x)$. Combining them with the bit strings provided by Bob yields for his share [enc(x)] the full share of Bob for $enc(x)$, which is returned as the circuit output.

3. MAC checks are computed. First, Alice's and Bob's MAC key shares provided as input are added:

$$a = a_1 + a_2 \bmod 2^{s+1},$$

   obtaining the shared MAC key a.

4. Then the product is computed:

$$a \cdot enc(x) \bmod 2^{s+1}.$$

5. By subtracting Alice's output strings for her share $[xa]$ (also modulo 2^{s+1}) from $a \cdot enc(x) \bmod 2^{s+1}$, Bob's share of $[xa]$ is obtained, which is also provided as circuit output.

Result of executing the modified GC. After the circuit completes:

- Alice knows her shares of $[enc(x)]$, $[a]$, and $[xa]$, since she generated them and provided them to the GC.

- Bob knows his share of $[a]$ and obtains from the GC his shares of $[enc(x)]$ and $[xa]$.

Thus, the parties transition to an SPDZ-sharing state with respect to the values $[enc(x)]$, $[a]$, and $[xa]$.

An evident advantage of this approach is that Bob does not need to publish anything at this stage. Therefore, he has no opportunity to deceive Alice by providing seemingly plausible but incorrect results.

3 Revealing Alice's and Bob's Shares

At this stage, we are in the SPDZ-sharing model modulo 2 and have n shared bits of the encrypted output. MAC checks for these bits have already been computed. The symmetric key is known to both Alice and Bob, so after fully revealing all bits of the encrypted output, both parties will be able to decrypt it and obtain the original result.

3.1 Algorithm Idea

The key idea of the algorithm is that Alice and Bob reveal bits alternately, checking at each step that neither party is cheating the other.

3.2 Step-by-Step Procedure

The reveal procedure is performed as follows:

1. Alice publishes her share of the i-th bit.
2. Bob publishes his share of the same bit.
3. Correctness is verified using MAC checks: both parties reveal the corresponding $[xa]$ shares and compare them with $[x] \cdot [a]$.
    - If the equality holds, the algorithm proceeds to the next bit.
    - If the equality does not hold, it indicates that someone attempted to cheat, and the interaction is immediately terminated.

Thus, bits are revealed sequentially. If no party deviates from the protocol at any step, both Alice and Bob obtain the correct result at the end of the algorithm.

3.3 Near-Guaranteed Delivery Guarantee

If at any stage one party begins to send incorrect data or stops interacting, in the best case the adversary obtains one bit of the correct output more than the honest party. Thanks to the correct-decryption indicator added earlier, the remaining suffix must be recovered via brute force, which requires twice the computational effort for the cheated party compared to the adversary. This achieves the property of *near-guaranteed delivery.*

Acknowledgements

The author thanks **Lev Soukhanov** and **Yaroslav Rebenko** for their support, insightful discussions, and valuable feedback that contributed to the development of the near-guaranteed delivery protocol.

References

[1] A. C.-C. Yao, *How to Generate and Exchange Secrets*, in Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167, IEEE, doi:10.1109/SFCS.1986.25.

[2] Michael J. Fischer, Nancy A. Lynch, Michael S. Paterson, *Impossibility of distributed consensus with one faulty process*, Journal of the ACM (JACM), 32(2):374–382, 1985.

[3] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, Chaoping Xing, *SPDZ2k: Efficient MPC mod* 2^k *for Dishonest Majority*, Cryptology ePrint Archive, Report 2018/482, 2018.