Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits

Ivan Oleynikov

May 3, 2021

1 Background

Secure multi-party computation (MPC) techniques allow multiple parties to collectively evaluate a function, ensuring that nobody learns anything except what they are supposed to learn. We will be looking at *actively secure* MPC protocols, such protocols stay secure even if an adversary corrupts some of the parties and changes the way they behave.

A typical MPC protocol allows n parties engaged in it to repeatedly do one of the following actions.

- *Input.* If a partiy i knows the value x, it can secret share it between all n parties. (Of course, this does not reveal x to the other parties.) We will use notation $[x]$ for values secret-shared this way.
- *Evaluate.* If values $[x]$ and $[y]$ are secret-shared between the parties, the parties can add, subtract or multiply them in order to obain $[x+y]$, $[x-y]$ or $[xy]$ secret-shared between them in the same way. The parties can also do these operations on shares and public values, e.g. they can obtain $[xc]$ by given $[x]$ and c publicly known to each of them.

  Most of the common MPC protocols use additive sharing: adding shares corresponds to adding the shared values. This means that addition and multiplication by a constant in such MPC protocols are essentially for free: they require no communication between the parties and very little local computation.
- *Reveal.* If $[x]$ is secret-shared, the parties can open their shares to a party i and reveal x to it.

The domain of the values on which an MPC protocol operates (like x, y, z above) is usually $\mathbb{Z}_M$ with M being a prime number $M = p$, a power of two $M = 2^m$, or just two $M = 2$. The latter case is special: with $M = 2$ the MPC protocol works with bits and arithmetic operations there correspond to Boolean operations; this case is called binary domain, and the other two are arithmetic domains.

Binary domain is especially good for expressing various non-arithmetic operations. If we represent a number as its bit decomposition, then we can do comparison, truncation, logical shifts using Boolean operations faster than in

the arithmetic domain. At the same time, additon and multiplication are a lot cheaper in the arithmetic domain: we can do them in a single operation while in binary domain they are expressed by big Boolean circuits.

Given that some applications (e.g. machine learning) require mixing operations of the two kinds, one might want to run two MPC protocols at the same time—one over arithmetic domain, and the other over binary domain—and find some way to convert values between the two.

Recently Rotaru et. al. [1] proposed *doubly-authenticated Bits* (daBits), an efficient technique to convert between arithmetic and binary domains. Let us denote sharing in the arithmetic domain with $[\cdot]_M$ and sharing in the binary domain with $[\cdot]_2$. A daBit is a random bit b which is simultaneously secret-shared in both domains: as $[b]_M$ and $[b]_2$.

If we have m daBits $b_0 \dots b_{m-1}$, we can convert an m-bit value $[x]_M$ into the binary domain:

1. Compute the number represented by the daBits $b_0 \dots b_{m-1}$ in the arithmetic domain, $[r]_M = \sum_{i=0}^{m-1} 2^i [r_i]_M$.
2. Compute and reveal $[x']_M = [x - r]_M$ to all the parties.
3. Input the bits of x' into the binary domain.
4. Evaluate the Boolean circuit that adds x' and r (r is represented by the daBits) modulo M in the binary domain.

They also provide a relatively efficient way to pre-generate daBits.

2 Contribution

An *extended daBit* (edaBit) is a tuple of random bits $(b_0, \dots b_{m-1})$ such that $r = \sum_{i=0}^{m-1} 2^i b_i$ is shared in arithmetic domain and the bits $b_0 \dots b_{m-1}$ are shared in the binary domain. The conversion trick from the previous section actually uses a edaBit, which it constructs from m daBits.

The authors of this paper suggest a novel, more efficient *cut-and-choose* technique to generate edaBits directly, without resorting to daBits. They observe that some of the binary MPC protocols are based on pre-generated Beaver triples, which are often generated using a round of cut-and-choose, and edaBits can also be generated with cut-and-choose. They join the two rounds of cut-and-choose into one and show that it can be made faster if we allow a small fraction of the Beaver triples to be incorrect and use special fault-tolerant Boolean circuits.

3 Importance

Converting between arithmetic and binary domains is necessary for many natural problems. For example, this can be useful for proximity testing where you compute Euclidean distances between your users and compare them against a threshold, or for machine learning where one often needs to do multiple alternations between arithmetic and threshold operations.

It is one of the asymptotically fastest ways to convert between arithmetic and binary MPC domains. It gets faster than daBits already for $10^3 \dots 10^4$ conversions.

4 Strong sides

The work is very practical and uses an interesting technique. Pre-generated edaBits can also be applied to do some non-arithmetic operations directly, in a more efficient way than using Boolean circuits.

5 What can be improved

The way this technique uses MPC protocols is not completely black-box, it assumes that the binary MPC is based on additive secret-sharing and Beaver triples, and furthermore that Beaver triples are generated using cut-and-choose technique.

This work generates edaBits in big batches. Although it is very efficient (in the amortized sense) if you generate 10^4 or more edaBits, it is not as good (both in computation and communication cost) if you want 10 edaBits.

References

[1] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Report 2019/207, 2019. `https://eprint.iacr.org/2019/207`.