Secure MPC Protocol for (Sample) Standard Deviation

November 11, 2020

1 Introduction

Recall that for a database of size n with data points x_i indexed by $i \in 1, \ldots, n$, the sample variance σ^2 is defined to be:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

where $\bar{x}$ is the sample mean:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The sample standard deviation of a database, σ, is the square root of the sample variance.

In the setting of MPC, rather than considering a single database of entries, we consider the case in which there is a set of parties where each party P_i holds entry x_i. The parties wish to compute the standard deviation of their inputs, but do not wish to reveal their inputs to anyone. The inputs are allowed to be negative or positive, and they may not be whole numbers.

MPC-friendly formulation However, there exists a well-known alternative formulation of the equation above, that show that σ^2 can be written as:

$$\begin{aligned}\sigma^2 &= \mathrm{E}[X^2] - (\mathrm{E}[X])^2 \\ &= \frac{\sum X^2}{n} - \frac{(\sum X)^2}{n^2} \\ &= \frac{n \sum X^2 - (\sum X)^2}{n^2}\end{aligned}$$

2 Secure Building Blocks

The following building blocks that are implemented in JIFF are useful to us here:

- share(input): secret-shares input between all parties
- share1.sadd(share2): secure addition of shares, which returns a share of results.
- share1.ssub(share2): secure subtraction of shares, which returns a share of results.
- share1.smult(share2, <op_id>, <truncate>): secure multiplication of shares, which returns a share of results.

  The fixed-point extension supports the optional <truncate> parameter, which determines whether the result of the multiplication is truncated to within the precision or not.

  It is unsafe to use a non-truncated value in a subsequent multiplication or division without truncating it, as well as in an addition or subtraction where the two operands do not have the same truncation/precision.
- share.cmult(n, <op_id>, <truncate>): secure multiplication of share by a constant, with a similar truncate optional parameter to the above.

3 Basic Implementation

We use the alternative MPC-friendly formulation from above. This alternative formulation is friendlier for MPC for the following reasons:

1. It has only a single division by a public constant, furthermore the division is the very last operation. Since division by a public constant is reversible, we can have our MPC computation output the numerator only, and perform the division after the MPC concludes locally at every party. This avoids performing an MPC division which is one of the most expensive operations under MPC.
2. The formulation contains $2+n$ multiplications: (1) n of them for squaring independent inputs, which can be performed locally by every party prior to sharing their input. (2) a multiplication by a public constant n which is efficient to perform under MPC. (3) Squaring the average: this is the only secret multiplication that needs to be performed.
3. Since we allow decimals as inputs, we must use the fixed-point extension of JIFF (as well as the negative number extension for negative inputs). This changes the performance behavior of the underlying primitives: namely, every multiplication by a secret requires an internal division to shift the decimal point to the left and truncate

the portion of the result that is beyond the specified accuracy. However, we are in luck: the only such multiplication here is the one for squaring the average. The result of that multiplication is not used for any division (the division is in the clear!) or multiplication. So we can instruct JIFF to not shift the decimal point so that the expensive division is skipped. In order for the following subtraction operation to be meaningful, the decimal point on both operands need to be at the same location, we can shift the decimal point to the right for the minuend, which is a communication-free operation, making both operands consistent, and then truncate and shift to the left locally after the output of the MPC was revealed to the parties.

Hence, our entire protocol operates as follows:

Local stage Every party i shares both X_i and X_i^2

MPC The parties together compute and reveal the following quantity:

$$\Delta = (n \otimes \sum X_i^2) - (\sum X_i \otimes \sum X_i)$$

Where $\otimes$ is the un-truncated fixed-point multiplication operator:

$$x \otimes y = x \times y \times 10^{\text{precision}}$$

Local stage Every party receives revealed Δ and computes:

$$\sigma = \sqrt{\frac{\Delta}{n \times 10^{\text{precision}}}}$$

3.1 Decimal Errors

All the operations used through out the protocol, both under MPC and locally, except for $\otimes$ are fixed point operations. They are guaranteed to satsify the following relation:

$$x \bigoplus y = \text{truncate}(x \oplus y, \text{precision})$$

Where truncate floors to the nearest fixed-point number towards $-\infty$.

In other words, they satisfy the following error bound:

$$(x \bigoplus y) - (x \oplus y) < 10^{\text{precision}}$$

Finally, the non-truncated multiplication is a perfectly accurate operation and introduces no error in-of-itself. Instead, it introduces an implicit error since it requires a division by an additional $10^{\text{precision}}$ afterwards. Note that this additional division is the only operation added to our MPC protocol compared to the fixed-point non-MPC version, which does not perform that division and uses truncated fixed point multiplication instead.

This means that the overall error introduced by our protocol is smaller or equal to that introduced by the non-MPC fixed point version of it.

3.2 Security

The only thing learned by any semi-honest party after the execution of our protocol is the standard deviation. The intermediate value revealed by our MPC computation is deducible from the standard deviation knowing the number of parties, which is public information. Nothing else is revealed.