Poster: Secure Computations of Trigonometric and Inverse Trigonometric Functions

Fattaneh Bayatbabolghani*, Marina Blanton†, Mehrdad Aliasgari‡ and Michael Goodrich§
*Computer Science and Engineering, University of Notre Dame, Email: fbayatba@nd.edu
†Computer Science and Engineering, University at Buffalo (SUNY), Email: mblanton@buffalo.edu
‡Computer Engineering and Computer Science, California State University, Long Beach, Email: mehrdad.aliasgari@csulb.edu
§Computer Science, University of California, Irvine, Email: goodrich@uci.edu

***Abstract*—Secure computation has been receiving a lot of attention in recent years and has been used in a variety of applications where protecting privacy of data throughout computation is essential. Simple operations form fundamental building blocks of secure computation and therefore proposing efficient and novel protocols for such operations will benefit a wide variety of applications such as secure biometric computation and mobile computing. For that reason, we design new and efficient secure protocols for trigonometric and inverse trigonometric functions using fixed-point arithmetic, which are applicable to both two-party and multi-party settings.**

***Index Terms*—Secure sine, cosine, arctangent, garbled circuit evaluation, secret sharing.**

I. Introduction

The motivation of this work comes from the need to compute trigonometric functions on private data in a number of applications such as, for example, secure fingerprint recognition. Because performance of secure computation techniques is of grand significance (as protecting secrecy of data throughout the computation often incurs substantial computational costs), besides security, efficient performance of secure trigonometric building blocks is crucial. Therefore, the focus of this work is on building secure and cost-efficient constructions for trigonometric and inverse trigonometric functions such as sine, cosine, and arctangent. Because of the similarities between the computation of sine and cosine functions, we primarily focus on sine, and secure computation for the cosine function can be implemented similarly.

II. Secure Setup

To build secure protocols, two secure computation frameworks in the presence of semi-honest parties are of interest to us: In the two-party setting, we build our solutions based on garbled circuit (GC) evaluation techniques, while in the multi-party setting, we employ linear secret sharing (SS) techniques. Due to a lack of space, additional information about these underlying techniques is provided in [1].

We use notation $[x]$ to denote that the value of x is protected. The computation takes place on fixed-point values which are represented using ℓ bits, k of which are stored after the radix point (and thus $\ell - k$ are used for the integer part). Each function is evaluated on private input and no information is to be revealed throughout protocol execution.

III. Secure Protocols

In this section, we build secure protocols for trigonometric and inverse trigonometric functions with efficiency of their execution in mind in the chosen two-party and multi-party settings. Our optimizations focus on the specific costs of the underlying techniques for secure arithmetic and ensure that we achieve efficiency together with precision and provable security guarantees.

We denote secure protocols as $[b] \leftarrow \mathsf{Sin}([a])$ and $[b] \leftarrow \mathsf{Arctan}([a])$, where a and b are fixed-point values. The input a to sine and cosine is represented in degrees and additional optimizations are possible if it is integer. The complexity of the designed protocols can be found in Table I.

A. Sine protocol

There are a variety of different approximation algorithms that can be used for trigonometric functions, many of which take the form of polynomial evaluation. Upon examining the options, we chose to proceed with the polynomials described in [2, Chapter 6], as they achieve good precision using only small degree polynomials. The polynomials used in [2] for trigonometric functions take the form $P(x^2)$ or $xP(x^2)$ for some polynomial P over variable x (i.e., use only odd powers for sine), which requires one to compute only half as many multiplications as in a polynomial with all powers present.

The approach used in [2] offers two types of polynomial approximations. The first type uses a regular polynomial $P(x)$ of degree N to approximate the desired function. The second type uses a rational function of the form $P(x)/Q(x)$, where P and Q are polynomials of degree N and M, respectively. For the same desired precision, the second option will yield lower degrees N and M and thus fewer multiplications to approximate the function. This option, however, is undesirable when the division operation is much costlier than the multiplication operation. In our setting, the rational form is preferred in the case of GC evaluation (at least for higher precisions) where multiplication and division have similar costs. However, with SS, the cost of division is much higher than that of multiplication, and we use the first option with regular polynomial evaluation.

It is assumed in [2] that the input to sine/cosine is in the range $[0, \pi/2]$. This means that input a given in degrees

Prot.	Secret sharing		Garbled circuits	
	Rounds	Interactive operations	XOR gates	Non-XOR gates
Sin	$2\log N + 16$	$2Nk + 8\ell + 2N + 6k + 4$	$(\max(N,M) + N + M + 2) \times (4\ell^2 - 4\ell) + 7\ell^2 + 4\ell(N+M) + 31\ell$	$(\max(N,M) + N + M + 2) \times (2\ell^2 - \ell) + 3\ell^2 + \ell(N+M) + 11\ell$
Arctan	$2\log N + 3\log \ell + 2\log(\frac{\ell}{3.5}) + 22$	$1.5\ell\log\ell + 2\ell\log(\frac{\ell}{3.5}) + 2Nk + 18.5\ell + 2N + 4\log(\frac{\ell}{3.5}) + 6$	$8N\ell^2 + 3\ell^2 - 4N\ell + 43\ell$	$4N\ell^2 + \ell^2 - N\ell + 15\ell$

TABLE I
PERFORMANCE OF PROPOSED SECURE BUILDING BLOCKS FOR FIXED-POINT VALUES.

needs to be reduced to the range $[0, 90]$ and normalized to the algorithm's expectations. Note that it is straightforward to extend the result of the computation to cover the entire range $[0, 2\pi]$ given the output of this function and the original input. Furthermore, because evaluating trigonometric functions on a smaller range of inputs offers higher precision, it is possible to apply further range reduction and evaluate the function on even a smaller range of inputs, after which the range is expanded using an appropriate formula or segmented evaluation. We refer the reader to [2] for additional detail.

Based on the desired precision, one can retrieve the minimum necessary polynomial degree used in the approximation to achieve the desired precision, then look up the polynomial coefficients and evaluate the polynomial(s) on the input. As mentioned before, sine is approximated as $xP(x^2)$ or $xP(x^2)/Q(x^2)$ and in the more general second case we obtain the following secure protocol that takes input in degrees in the range [0, 360):

$[b] \leftarrow \mathsf{Sin}([a])$

1) Compute $[s] = \mathsf{LT}(180, [a])$.
2) If $([s])$ then $[a] = [a] - 180$.
3) If $(\mathsf{LT}(90, [a]))$ then $[a] = 180 - [a]$.
4) Compute $[x] = \frac{1}{90}[a]$ and then $[w] = [x]^2$.
5) Lookup the minimum polynomial degrees N and M for which precision of the approximation is at least k bits. Then, lookup polynomial coefficients $p_0, \ldots, p_N$ and $q_0, \ldots, q_M$ for sine approximation.
6) Compute $([z_1], \ldots, [z_{\max(N,M)}]) \leftarrow \mathsf{PreMul}([w], \max(N, M))$.
7) Set $[y_P] = p_0 + \sum_{i=1}^{N} p_i[z_i]$ and $[y_Q] = q_0 + \sum_{i=1}^{M} q_i[z_i]$.
8) Compute $[y] \leftarrow \mathsf{Div}([y_P], [y_Q])$.
9) If $([s])$ then $[b] = 0 - [x]$ else $[b] = [x]$.
10) Compute and return $[b] = [b] \cdot [y]$.

Cosine is implemented similarly and can be found in [1]. Recall that we recommend using a rational function in the form of $P(x)/Q(x)$ as in the above protocol for GCs-based implementation of high precision. With SS, we modify the protocol to evaluate only a single polynomial P of (a different) degree N and skip the division operation in step 8.

B. Arctangent protocol

In the case of inverse trigonometric functions and arctangent in particular, the function input domain is the entire $(-\infty, \infty)$ and the range of output is $(-\pi/2, \pi/2)$. We recall that arctangent is an odd function (i.e., $\arctan(-x) = -\arctan(x)$) and for all positive inputs we have $\arctan(x) + \arctan(\frac{1}{x}) = \pi/2$. It is, therefore, sufficient to approximate arctangent over the interval of $[0, 1]$. To this end, we use the technique introduced by Medina in [3]. In particular, [3] defines a sequence of polynomials over the input domain of $[0, 1]$, denoted as $h_N(x)$, with the property that $|h_N(x) - \arctan(x)| \leq \left(\frac{1}{4^{5/8}}\right)^{\deg(h_N)+1}$. We use this formula to determine the degree N for any k-bit precision. This degree N is logarithmic with respect to the desired precision. Afterwards, the coefficients of $h_N(x)$ are computed from the recursive definitions in [3]. We choose this approach over other alternatives such as [2] for its efficiency.

Another candidate for arctangent evaluation is the well-known Taylor series of arctangent. However, it provides a possibly reasonable precision if the input is bound to be very close to one point from the input domain, which we cannot assume. In addition, arctangent Taylor series converges very slowly and we do not pursue this option.

Our secure protocol to approximate arctangent based on the approach from [3] is given next:

$[b] \leftarrow \mathsf{Arctan}([a])$

1) Compute $[s] \leftarrow \mathsf{LT}([a], 0)$.
2) If $([s])$ then $[x] = 0 - [a]$ else $[x] = [a]$.
3) Compute $[c] \leftarrow \mathsf{LT}(1, [x])$.
4) If $([c])$ then $[d] = \pi/2$, $[y] \leftarrow \mathsf{Div}(1, [x])$; else $[d] = 0$, $[y] = [x]$.
5) Lookup the minimum polynomial degree N for which precision of the approximation is at least k bits. Then, lookup polynomial coefficients $p_0, \ldots, p_N$ for arctangent approximation from [3].
6) Compute $([z_1], \ldots, [z_N]) \leftarrow \mathsf{PreMul}([y], N)$.
7) Set $[z] = p_0 + \sum_{i=1}^{N} p_i[z_i]$.
8) If $([c])$ then $[z] = [d] - [z]$ else $[z] = [d] + [z]$.
9) If $([s])$ then $[b] = 0 - [z]$ else $[b] = [z]$.
10) Return $[b]$.

ACKNOWLEDGMENTS

This work was supported in part by grants 1223699, 1228639, 1319090, and 1526631 from the National Science Foundation and FA9550-13-1-0066 from the Air Force Office of Scientific Research, as well as DARPA agreement no. AFRL FA8750-15-2-0092. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. Department of Defense.

REFERENCES

[1] F. Bayatbabolghani, M. Blanton, M. Aliasgari, and M. Goodrich. Secure fingerprint alignment and matching protocols. *arXiv Report 1702.03379*, 2017.

[2] J. Hart, E. Cheney, C. Lawson, H. Maehly, C. Mesztenyi, J. Rice, H. Thacher, and C. Witzgall. *Computer approximations*. John Wiley & Sons, Inc., 1968.

[3] H. Medina. A sequence of polynomials for approximating arctangent. *The American Mathematical Monthly*, 113(2):156–161, 2006.