Journal of Information Security and Applications 000 (2017) 1–11

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Secure numerical and logical multi party operations

Johannes Schneider[a,*], Bin Lu[b]

[a] University of Liechtenstein, Liechtenstein
[b] Ecole Polytechnique Federale de Lausanne, Switzerland

ARTICLE INFO

Article history:
Available online xxx

Keywords:
Big data
Secure numerical computations
Secure comparisons
Client-server computation
Secure cloud computing
Secure multi-party computation
Privacy preserving data mining

ABSTRACT

We derive algorithms for efficient secure numerical and logical operations in the semi-honest model ensuring statistical or perfect security for secure multi-party computation (MPC). To derive our algorithms for trigonometric functions, we use basic mathematical laws in combination with properties of the additive encryption scheme, ie. linear secret sharing, in a novel way for the JOS scheme [23]. For division and logarithm, we use a new approach to compute a Taylor series at a fixed point for all numbers. Our empirical evaluation yields speed-ups for local computation of more than a factor of 100 for some operations compared to the state-of-the-art.

1. Introduction

Consider the following tasks: i) Identify people on a picture without looking at it; ii) Outsource computations giving away encrypted data but keeping keys private. Both tasks come with the challenge that there is no access to the non-encrypted data. It seems impossible to work on encrypted data only. Surprisingly, computing on encrypted data is indeed doable. A rather mature technique is secure multi-party computation (MPC) relying on non-collusion of a network of parties. To this date, MPC suffers heavily from its performance overhead. Whereas a lot of emphasis has been put on optimizing the computation of Boolean circuits, only limited effort has been made to secure numerical operations efficiently. For example, prior work did not deal with trigonometric functions such as sine or cosine needed in many applications, such as signal processing in an industrial context. In fact, aside from basic operations (such as addition and multiplication) no complex mathematical operation can be carried out efficiently and accurately. Prior work uses either (slow) iterative schemes or approximations of the function to compute. We address this gap using a recent scheme called JOS [23] that explicitly separates between keys and encrypted values. It supports various variants of linear secret sharing, eg. additive blinding with and without modulo as well as using XOR. The distinction between keys and encrypted values together with the simple encryption schemes lend itself well to make use of basic mathematical equations that relate the ciphertext, the plaintext and the key. In essence, to compute some functions we can use the same implementation (plus a few additional operations) used for plaintexts on the ciphertexts or keys as we show for trigonometric functions. This makes it possible to benefit from the long history of optimizations of implementations and algorithms for non-encrypted data. For illustration, our empirical evaluation yields that the amount of local computation per party to compute a sine function is only about a factor 2 more than for computation on non-encrypted data. At times, we also employ the idea of using multiple encryptions of the same plaintext to derive a system of equations to leverage operations on non-encrypted data. This helps to deal with a reduced key space caused by the inability to evaluate certain functions designed for non-encrypted data on arbitrary keys. Additionally, we discuss a method for computing Taylor series based on scaling the secret value. The scaling makes it possible to develop the series at a fixed number (for the entire value range of a secret). This approach yields fast conversion for a broad range of functions as we demonstrate for division and logarithm.

For logical operations, the key ingredient is an efficient comparison protocol for equality (with zero) and for checking if a value is less than zero. This is done by using algorithms for conversions between encryption schemes and using large Fan-In gates. Our ideas might prove valuable in other settings or using other schemes aside from JOS well.

1.1. Contributions

- Presenting the first algorithms for efficient computation of trigonometric functions, ie. sine, cosine and tangent. They provide statistical security using only five rounds, local computation proportional to computation without encryption and com-

* Corresponding author.
E-mail address: johannes.schneider@uni.li (J. Schneider).

http://dx.doi.org/10.1016/j.jisa.2017.05.001
2214-2126/

munication of $O(k)$ bits where k is the security parameter. They improve on series-based techniques [14] by more than a factor of 10 in local computation and communication.
- Stating an algorithm for calculating Taylor series efficiently for a wide range of functions, demonstrated for division and logarithm. More concretely, we improve the round complexity of the state-of-the-art [1,7] for division and logarithm for computation on 32-bit floats and 64-bit double values by more than 10 rounds.
- Presenting an algorithm for division of a confidential number by a public divisor requiring only one round without the need to perform comparisons, which take significantly more than one round [6,27].
- Introducing a number of efficient operations using the JOS scheme [23] for comparison(equality, less than), conversion between different forms of encryptions and large fan-in gates that achieve comparable or better performance to prior work using different schemes. They require a constant number or almost constant rounds, ie. $O(\log^* l)$ and $O(\log_b l)$, where l is the number of bits of the encrypted value and b a parameter. In terms of local computation our equality protocol is more than a factor 100 faster than the state-of-the-art.

1.2. Outline

After some preliminaries (Section 1.3) focusing on summarizing the JOS scheme and presenting some notation and conventions, we introduce algorithms for three areas: conversions between encryption schemes (Section 2), logical operations (Section 3) and numerical operations (Section 4). There are some interdependencies between algorithms from different sections, eg. some conversion algorithms between encryption schemes are used by some algorithms for logical operations. Finally, we give a short empirical evaluation (Section 5) and we discuss related work (Section 6).

1.3. Preliminaries and notation

We briefly recapitulate notation and concepts from the JOS scheme [23]. For a secret value $a \in [0, 2^l - 1]$ with l bits and a key K with $b \geq l$ bits we consider three kinds of linear encryptions:

- $ENC_K(a) = a + K$: (Purely) additive encryption
- $ENC_K(a) = (a + K) \mod 2^l$: Additive (modulo) encryption
- $ENC_K(a) = a \oplus K$: XOR encryption

Given an encryption $ENC_K(a)$ we denote the effective maximum number of bits by the key or the encrypted value by l_E. For $ENC_K(a) := (a + K) \mod 2^l$, we have $l_E = l$. For additive encryption of a key $K \in [0, 2^b - 1]$ with b bits, we have $l_E = b + 1$ for $a + K \geq 2^b$ and $l_E = b$, otherwise. Denote by subscript i the i^{th} bit of a number in little Endian notation, eg. for $a = 10$: $a_0 = 0$ and $a_1 = 1$. In particular, E_i, a_i and K_i denote the i^{th} bit of $ENC_K(a)$, a and K. The JOS scheme [23] uses three parties, namely: a key holder (KH), an encrypted value holder (EVH) and a helper (HE). The KH holds keys only (most of the time), the EVH keeps encrypted values (most of the time) and the helper can have either of them, but it is not allowed to have an encrypted value and the matching key. For additive encryption $a + K$, we define the carry bit c_i to be the "carry over" bit that is added to a_{i+1} aside from k_{i+1} during encryption. Thus, by definition $c_0 := 0$ and for $i > 0$ we have $c_i := 1$ iff $c_{i-1} + a_i + k_i > 1$, otherwise $c_i := 0$. Frequently, we encode 'TRUE' as one and 'FALSE' as zero. Our algorithms process as inputs encrypted values from the EVH and keys from the KH. They ensure the encrypted value of the result is held by the EVH and its key by the KH. For inputs and outputs we write an encrypted value and key as pair $(ENC_K(a), K)$. Thus, we write for a function f operating on an encryption of value a returning an encrypted value rE and key rK the following $(rE, rK) := f(ENC_K(a), K)$. In particular, we use the multiplication protocol *MUL* [23] and the protocol for bitwise AND, ie. *AND*. Both typically take two confidential numbers as input. For *AND* we sometimes use larger Fan-Ins as described in [23]. We also assume a protocol for computing the power a^i for a non-confidential integer $i > 0$, ie. power $POW((ENC_K(a), K), i)$. It can be implemented using the multiplication protocol *MUL* using $O(\log i)$ multiplications. To reduce the number of bits needed we also use scaled power computation $SCALEDPOW((ENC_K(a), K), i, s)$, ie. for a non-confidential scaling factor s we compute a^i / s^{i-1}. We enumerate keys either by using primes, eg. K', K'', K''' or using numbers K^0, K^1, K^2. The terms E', E'' and E''' denote that an encrypted value with K', K'' and, respectively, K'''.

2. Conversions between encryptions

We show how to convert between all three encryption schemes, ie. XOR, additive with and without modulo.

2.1. Additive ↔ XOR encryption

Algorithm AddToXOR computes an XOR encryption of a secret from an additive encryption(with or without modulo). It uses the carry bits algorithm in Section 3.5[1] and it exploits the definition of the carry bit c_i to get XOR encryptions of the bits.

Theorem 1. *Algorithm 1 converts correctly and securely from additive to XOR encryption.*

Algorithm 1 AddToXOR(encrypted value $ENC_K(a)$, key K).

1: $(ENC_{K_i''}(c_i), K_i'') := CarryBits(ENC_K(a), K)$ with $i \in [0, l-1]$
2: $K_i' := K_i \oplus K_i''$ {by KH}
3: $ENC_{K_i'}(a_i) := e_i \oplus ENC_{K_i''}(c_i)$ {by EVH}
4: return $(ENC_{K_i'}(a_i), K_i')$

Proof. For bit e_i of $ENC_K(a) = a + K$ (with or without $\mod 2^l$) we have using the definition of the carry bit:

$$e_i = (a_i + k_i + c_i) \mod 2 = a_i \oplus k_i \oplus c_i \quad (1)$$

This can also easily be verified by listing all 8 options for each bit a_i, k_i and $c_i \in \{0, 1\}$. In Algorithm AddToXOR we compute $e_i' := e_i \oplus e_i''$. We show that this definition ofo e_i' is equivalent to claimed return value, ie. an XOR encryption of a_i with key k_i', ie. $a_i \oplus k_i'$ with $k_i' := k_i \oplus k_i''$. We have:

$$\begin{aligned}
& a_i \oplus k_i' \\
&= a_i \oplus k_i \oplus k_i'' \oplus 0 \quad \text{(by definition of } k_i' \text{ and since } x \oplus 0 = x) \\
&= a_i \oplus k_i \oplus k_i'' \oplus (c_i \oplus c_i) \quad \text{(since } x \oplus x =) \\
&= (a_i \oplus k_i \oplus c_i) \oplus k_i'' \oplus c_i \quad \text{(due to commutativity of } \oplus) \\
&= ((a_i + k_i + c_i) \mod 2) \oplus e_i'' \\
&\qquad \text{(by definition of } e_i'' \text{ and Equation (1))} \\
&= e_i \oplus e_i'' \quad \text{(by Theorem 8)} \\
&=: e_i'
\end{aligned}$$

Security follows from the security of Algorithm CarryBits (Theorem 8) and the fact that Algorithm AddToXOR does not

[1] Due to the dependencies among algorithms, it is not possible to avoid referencing later sections of the paper without removing a consequent structuring of the paper into three main sections (encryption conversions, logical and numerical operations).

Please cite this article as: J. Schneider, B. Lu, Secure numerical and logical multi party operations, Journal of Information Security and Applications (2017), http://dx.doi.org/10.1016/j.jisa.2017.05.001