SECURE COMPUTATIONS ON NON-INTEGER VALUES

M. Franz[*], *B. Deiseroth*[*], *K. Hamacher*[+], *S. Jha*[#], *S. Katzenbeisser*[*], *H. Schröder*[*]

[*] Technische Universität Darmstadt, Security Engineering Group
[+] Technische Universität Darmstadt, Computational Biology Group
[#] University of Wisconsin, Computer Sciences Department

ABSTRACT

In this paper we present for the first time a framework that allows secure two-party computations on approximations of real valued signals. In our solution, we use a quantized logarithmic representation of the signal samples, which enables to represent both very small and very large numbers with bounded relative error. We show that numbers represented in this way can be encrypted using standard homomorphic encryption schemes; furthermore we give protocols that allow to perform all arithmetic operations on such encrypted values. Finally we demonstrate the practicality of our framework by applying it to the problem of filtering encrypted signals.

I. INTRODUCTION

Due to the ever increasing amount of digitally stored data, there is a growing need for technical solutions that protect sensitive personal information. In the past, data privacy was mainly assured through procedures, laws or access control policies. However, these protection mechanisms are ineffective once data is outsourced to partially untrusted servers or processed by third parties. To alleviate this problem, special Privacy Enhancing Technologies have been proposed that allow to keep sensitive data encrypted and compute directly on encrypted values using cryptographic protocols [1], [2]. This strategy allows to control, by design of the protocols, the amount of sensitive data that is leaked to the protocol participants. Commonly, these constructions use techniques from Secure Multiparty Computation [3], [4], which offers various protocols to compute securely with data represented as integers.

However, many computational problems require processing of very small non-integer values, some of them in the range of 10^{-1} to 10^{-300} or even smaller. The most prominent examples are applications that require processing of probabilities over large sets of events, such as dynamic programming algorithms, where a large number of probabilities have to be multiplied. Other problems require computing with values taken from a large interval such as $[2^{-100}, 2^{100}]$, where the relative representation error should be constant. Standard tools from Secure Multiparty Computation provide rather efficient ways to perform computations on integer values; implicitly, these tools also allow to compute on rational numbers of a fixed precision (by representing the numerator and denominator separately) and on integers in fixed point representation (by appropriate scaling and quantization). However, these frameworks usually cannot be used for the above-mentioned problems, as they cannot represent very small values; furthermore, rounding errors accumulate once a large number of operations are performed. We strive for a solution that provides *numerical stability*, where computations can be performed on arbitrarily small or large numbers with finite precision, but yield results as if they were computed with standard floating-point arithmetic.

In this paper we present a computational framework which allows two semi-honest parties to perform secure computations on values which come from a bounded real interval $\mathcal{D}_\ell = [-\ell; \ell]$. While we refrain (for performance reasons) from implementing standard floating point encodings, we do use a representation that shares the most important properties of floating point arithmetic, i.e., values in $\mathcal{D}_\ell$ close to zero are represented with higher accuracy than larger values, yielding a bounded relative error, regardless whether representing very small or very large values. We further demonstrate the practicality of our framework through a series of tests. In particular, we perform measurements of the communication and computation complexity for various parameter sizes; furthermore, we show that for a concrete scenario (namely a digital filter) that our representation of numbers does not degrade the results significantly when compared to a reference implementation.

II. RELATED WORK AND BUILDING BLOCKS

Secure Multiparty Computation (SMC). Several constructions for Secure Multiparty Computation [3], [4] and Secure Function Evaluation [5], [4] are known in the literature. In the latter case, a set of players evaluate a given function on their private inputs; while the result of the function should be available for everyone, the parties do not want to share their inputs with each other. Various approaches for securely evaluating a function have been developed for different function representations, namely combinatorial circuits [5], [6], Ordered Binary Decision Diagrams [7], branching programs [8], or one-dimensional look-up tables [9]. While these methods target computations performed over the integers only, extensions were proposed that can handle rational numbers [10] and real values in fixed point notation [11]. The problem of computing securely with non-integer values in a numerically stable way, as presented in this paper, is still unsolved.

The availability of efficient SMC methods stimulated their use to protect privacy of valuable data. For example, several solutions have been proposed that use SMC to enhance privacy in auctions [12], data clustering [13], [14] or filtering [15]. Recently, there is also an increased interest in combining SMC with methods of signal processing in order to be able to privately analyze signals; examples are the analysis of medical signals [16] or the evaluation of biometrics on encrypted data [2]. All these systems currently rely on a fixed-point representation of the signals.

Paillier cryptosystem. As a central cryptographic tool, we use a semantically secure additively homomorphic public-key encryption scheme introduced by Paillier [17]. Let $n = pq$ of size k, with p, q distinct prime numbers and k in the range of 1000-2048 bits. Also let $g = n + 1$. To encrypt a message $m \in \mathbb{Z}_n$, one selects a random value $r \in \mathbb{Z}_n$ and computes the ciphertext $c = (mn + 1)r^n = g^m r^n \bmod n^2$. We will write the encryption of a message m in the Paillier cryptosystem as $[m]$. Since all encryptions in the proposed protocols will be computed using one fixed public key, we do not specify the key explicitly. It is easy to see that the Paillier encryption scheme is additively homomorphic: an encryption $[a + b]$ can be computed by $[a + b] = [a][b]$, where all operations are performed in the algebra of the message or ciphertext space. Furthermore, messages can be multiplied with constants under encryption, i.e., given an encrypted message $[a]$ and a constant b, it is possible to compute $[ab]$ by $[ab] = [a]^b$. Further details can be found in [17].

Secure multiplications. While additions and multiplications with constants are immediately possible when using homomorphic encryption, multiplications of ciphertexts require execution of a cryptographic protocol. In this work, we often require a multiplication protocol for a two party scenario as follows: Party $\mathcal{A}$ holds a private key for some Paillier public key n. Party $\mathcal{B}$ holds two encrypted values $[a], [b]$ and should obtain an encryption $[ab]$. This can be achieved as follows: party $\mathcal{B}$ generates two random values $r_a, r_b \in_R \mathbb{Z}_n$ and computes encryptions $[a + r_a]$ and $[b + r_b]$. $\mathcal{B}$ sends these values to $\mathcal{A}$, who subsequently decrypts, multiplies them and sends an encryption $[(a + r_a)(b + r_b)]$ back to party $\mathcal{B}$. Party $\mathcal{B}$ now recovers the product by computing

$$[ab] = [(a + r_a)(b + r_b)][b]^{-r_a}[a]^{-r_b}[-r_a r_b].$$

If a and b are binary values from $\{0, 1\}$, then the operations become much more lightweight. In this case the blinding factors r_a, r_b can also be chosen as binary values. Party $\mathcal{A}$ now sends $[a \oplus r_a]$ and $[b \oplus r_b]$, where $\oplus$ denotes XOR, to party $\mathcal{A}$ who subsequently sends back the product $[(a \oplus r_a)(b \oplus r_b)]$. As before, using the values r_a, r_b party $\mathcal{B}$ recovers the result $[ab]$. We denote a secure multiplication by $*$, a secure multiplication of binary values by $\circledast$.

Comparison of encrypted values. Sometimes we require to compare two encrypted values $[a]$ and $[b]$ in order to obtain the encryption of a bit γ indicating whether $a \leq b$ or $a > b$. This problem is a variant of Yao's Millionaire's Problem and has extensively been studied in the literature; for example, an optimized protocol in the above scenario can be found in [2]. Such an operation further allows to compute the minimum of the two values a and b in an oblivious way.

Oblivious Pseudo-Random Function Evaluation (OPRF). An OPRF is a two-party protocol, where one party holds a secret key k and the second party wishes to evaluate a pseudo-random function $F_{\text{PRF}}(k, x)$ on a value x. The evaluation is oblivious in the sense that the party holding the key k does not learn x, while the other party only obtains $F_{\text{PRF}}(k, x)$ and has no knowledge of k [18], [19], [20].

III. PRIVATE FUNCTION EVALUATION

In this section we describe a protocol which will be the basis of our framework for secure computations on non-integer values. In particular, we give a two-party protocol which allows parties $\mathcal{A}$ and $\mathcal{B}$ to evaluate any complex function $f : X \to Y$, with $X, Y \subset \mathbb{N}$ and $X = [x_l; x_u] \subseteq [-2^\kappa + 1; 2^\kappa]$ in a private way: This primitive allows to evaluate f on an encryption $[x]$ of a value $x \in X$, producing an encryption $[f(x)]$, while keeping x and $f(x)$ secret from both parties. We will assume that party $\mathcal{B}$ holds an homomorphic encryption $[x]$, whereas party $\mathcal{A}$ knows the correct decryption key and assists in the computations. We further assume that all values are encrypted in the Paillier encryption scheme, thus messages come from the plaintext space $\mathbb{Z}_n$ for some RSA modulus n (see Section II).

Our approach is based on the oblivious evaluation of a pseudo-random function and consists of two phases: one initialization and one evaluation phase. In the initialization phase, $\mathcal{A}$ generates a prime number p in a way that $n \mid (p-1)$ and an element g of order n in $\mathbb{Z}_p$. $\mathcal{A}$ further selects a random element $k \in \mathbb{Z}_n$. These choices assure that $g_x = g^{\frac{1}{k+x}} \pmod p$ is a pseudo-random function [20]. Let again $X = \{-2^\kappa - 1, \dots, 2^\kappa - 1\}$ be the set of numbers on which f operates on. $\mathcal{A}$ prepares a table $\mathcal{T}$ with two columns, where the first column contains the value $g_z = g^{\frac{1}{k+x}}$ and the second column contains an encryption of $f(x)$ for each $x \in X$. Furthermore $\mathcal{A}$ permutes the table by sorting in ascending order according to the first row. Now $\mathcal{A}$ sends the table $\mathcal{T}$ together with an encryption of k used in the function $g^{\frac{1}{k+x}}$ to party $\mathcal{B}$.

In the evaluation phase, $\mathcal{B}$ obliviously learns an encryption $[f(x)]$ for some value $[x]$ (obliviously to himself and to party $\mathcal{A}$), given $[k]$ and $\mathcal{T}$, by running Protocol 1. $\mathcal{A}$ and $\mathcal{B}$ obliviously evaluate the pseudo-random function $g_x = g^{\frac{1}{k+x}}$ on $[x]$, where the result will be available in the clear only to $\mathcal{B}$ at the end of the protocol. This knowledge allows $\mathcal{B}$ to "look up" the desired result in the table received before by selecting the row of $\mathcal{T}$ that contains an encryption of $f(x)$. As party $\mathcal{B}$ does not know the key k in the clear, he cannot access other encryptions in the table, since he cannot look up their positions in $\mathcal{T}$.

Protocol 1 Computing $f(x)$ using OPRFs

Input: Party $\mathcal{B}$: $[x]$ with $x \in X$

Output: Party $\mathcal{B}$: $[f(x)]$

```
   {Initialization:}
1: Party $\mathcal{A}$:
   Choose $k \in_R \mathbb{Z}_n$
   for each $x \in \mathcal{Z}$
       Add $g_x = g^{\frac{1}{k+x}}$ and $[f(x)]$ to table $\mathcal{T}$
   end
   Send $[k]$ and sorted table $\mathcal{T}$ to party $\mathcal{B}$

   {Evaluation:}
2: Party $\mathcal{B}$:
   Let $r \in_R \mathbb{Z}_n$ in $[y'] := [r(x+k)+k] = [k]([x][k])^r$
   Send $[y']$ to party $\mathcal{A}$.
3: Party $\mathcal{A}$:
   $y'$ = Decrypt($[y']$)
   $y = y' - k$
   $g_y = g^{(y^{-1})}$
   Send $g_y$ to party $\mathcal{B}$.
4: Party $\mathcal{B}$:
   $g_x = g_y^r$
   Look up entry $g_x$ in $\mathcal{T}$ and obtain $[f(x)]$.
```

The indirection step of computing g_x allows to use the same table $\mathcal{T}$ prepared in an initialization phase for multiple queries. Thus, the complexity of generating $\mathcal{T}$ can be amortized over a certain number of operations, yielding to a good practical performance. However, if $\mathcal{T}$ is used multiple times, there is a chance that a single value will be requested more than once. This information will be leaked to party $\mathcal{B}$. Depending on the application scenario, this may be acceptable if this event happens rarely. Note that party $\mathcal{B}$ will have no information which value was queried more than once; furthermore, he will not learn any information on values queried only once. To limit this information flow, the table $\mathcal{T}$ can be updated periodically after a number of queries have been performed: to this end, party $\mathcal{A}$ runs the initialization phase again and provides party $\mathcal{B}$ with the new table and a new key $[k']$.

Correctness. Party $\mathcal{B}$ obtains the correct value g_x, as $g_x = g_y^r = (g^{y^{-1}})^r = (g^{(r(x+k))^{-1}})^r = g^{(x+k)^{-1}}$. As party $\mathcal{A}$ used the same value k when generating the table $\mathcal{T}$, party $\mathcal{A}$ will retrieve a correct encryption of $f(x)$.

Security. The table $\mathcal{T}$ itself does not leak any information: the first row consists only of pseudo-random elements, whereas the second one contains semantically secure Paillier encryptions.

On computation of the value g_x, information can only be leaked when party $\mathcal{A}$ decrypts the value $[y']$ or when party $\mathcal{B}$ obtains the value g_y. However, the value y' is obfuscated by multiplicative blinding and therefore is indistinguishable from a value chosen uniformly at random from $\mathbb{Z}_n$. Furthermore, under the Decisional q-Diffie-Hellman Inversion Assumption (see [20]) party $\mathcal{B}$ can not, in particular, compute the discrete logarithm for g_y to base g in $\mathbb{Z}_p$, thus no information can be inferred from seeing g_y.

Optimizations. As $\mathcal{T}$ can become very large, depending on the choices made for f and X, we present several optimizations to the OPRF construction which help to decrease the size of $\mathcal{T}$.

Table index. As each g_z is a random value of approximately 1024 bits, but $\mathcal{T}$ has at most $|\mathcal{T}| \ll 2^{1024}$ entries, it is sufficient to choose only the least significant bits of g_x as an index. For example, for realistic examples such as $|\mathcal{T}| = 2^{16}$ it suffices to choose the 32 least significant bits.

Statistical hiding. Also the size of the second column can be reduced. Let $u > 0$ and $r_0, \dots, r_u$ be random ω-bit numbers, where ω is a security parameter. Rather than storing an encryption of the value $f(x)$ in the table $\mathcal{T}$, it suffices to store a value $f(x)+r$, where $r = \sum_{i=0}^{u} \alpha_i \cdot r_i$ is a linear combination of random values $r_0, \dots, r_u$. If we provide party $\mathcal{B}$ with encryptions $[r_i]$, and if the values α_i are chosen pseudo-randomly, i.e., by extracting them as random chunks (of small bit-length) of the value g_z, then party $\mathcal{B}$ can reconstruct the value $f(x)$ by encrypting the table entry in row with index g_z and then subtracting $[\alpha_i r_i]$. As long as party $\mathcal{B}$ does not access more than $u-1$ blinded values from table $\mathcal{T}$ (which each gives him a linear combination of the form $\sum_{i=0}^{u} \alpha_i \cdot r_i$), the values r_i remain hidden and thus also the values $f(x)$ remain statistically hidden from $\mathcal{B}$.

IV. SECURE COMPUTATIONS ON REALS

In this section we present a framework which allows two parties to perform secure computations on values from some interval $\mathcal{D}_\ell = [-\ell; +\ell] \subset \mathbb{R}$. Note that we describe the framework in its full generality; in case that computations are performed only on a limited set of numbers (e.g., only on probabilities in the interval $[0; 1]$) further optimizations are possible.

IV-A. Data Representation

We start by giving a description of how values from the real domain $\mathcal{D}_\ell$ will be represented. We require the representation to meet the following conditions:

- First, the scheme should allow a representation as integer values, in order to use efficient methods from generic Secure Multiparty Computation for some sub-problems.
- Second, the representation should have a similar behavior as floating point arithmetic: the precision for numbers close to 0 should be higher than for numbers far away from 0. This is desirable in order to achieve a bounded relative error when representing both very small and rather big numbers.
- The data encoding should allow us to perform certain arithmetic operations directly on the encoded values.

Our solution to this problem builds on the work of Kingsbury and Rayner [21]. We use a logarithmic representation for the numbers in the interval $\mathcal{D}_\ell = [-\ell; \ell]$ with $\ell > 0$. Let $S \in \mathbb{N}$ be a scaling factor, $B > 0$ a suitable base for some logarithm and C be a constant with $C \geq \ell$. We represent a value $x \in \mathcal{D}_\ell$ as a tuple $(\rho_x, \sigma_x, \tau_x)$. Here, $\rho_x \in \{0, 1\}$ is a flag indicating whether the represented number is zero, i.e., $\rho_x = 1$ if x is not equal to zero and $\rho_x = 0$ otherwise. Similarly, σ_x encodes the sign, i.e., $\sigma_x = 1$ if $x \geq 0$ and $\sigma_x = -1$ otherwise. If $\rho_x = 1$ we compute an encoding τ_x of the absolute value as $\tau_x = \lceil -S \cdot \log_B(\frac{|x|}{C}) \rfloor$, where $\lceil \cdot \rfloor$ denotes rounding to the nearest integer. In case of $\rho_x = 0$ both σ_x and τ_x can contain arbitrary values. If we limit the space for the values τ_x to some interval $[0; 2^\kappa - 1]$ by storing τ_x as κ-bit number, we can exactly represent a set of $2^{\kappa+1} + 1$ distinct numbers from $\mathcal{D}_\ell$.

We denote the representation of any given $x \in \mathcal{D}_\ell$ in $\mathcal{L}_\kappa := \{0, 1\} \times \{-1, 1\} \times \{0, \dots, 2^\kappa - 1\}$ by $\overline{x}$. The backward transformation $\mathcal{L}_\kappa \to \mathcal{D}_\ell$ is given by

$$(\rho_x, \sigma_x, \tau_x) \mapsto \rho_x \cdot \sigma_x \cdot C \cdot B^{(-\tau_x/S)}.$$

IV-B. Parameters

According to the desired level of accuracy and the size ℓ for the interval of represented numbers the parameters S, B, C and κ can be adjusted. We first note that for each pair (S_1, B_1) and a base B_2 there exists a unique S_2 such that $S_1 \log_{B_1}(x) = S_2 \log_{B_2}(x)$ for all $x \in \mathbb{R}^+$. Thus it suffices to consider only one fixed base B and adjust the parameters S, C and κ accordingly.

The relative distance $\Delta x/x$ between two subsequent numbers in $\mathcal{D}_\ell$, which can exactly be represented in our encoding, is

$$\begin{aligned} \Delta x/x &= \frac{C \cdot B^{-\tau_x/S} - C \cdot B^{-(\tau_x+1)/S}}{C \cdot B^{-\tau_x/S}} \\ &= 1 - B^{-1/S}, \end{aligned} \tag{1}$$

and thus depends only on B and S. To achieve a fixed maximum relative error $\Delta x/x$ it thus suffices to fix a base B and compute S as $S = -(\log_B(1 - \Delta x/x))^{-1}$. For example, for a base $B = 2$ and scaling factor $S = 100$, the maximal relative representation error is given by $6.9 \cdot 10^{-3}$.

Other factors to consider when choosing the parameters are the smallest and the greatest positive value which can be represented; in our encoding we have $C \cdot B^{-(2^\kappa - 1)/S}$ for the minimal and $C \cdot B^{-1/S}$ for the maximal positive value. As the scheme represents numbers from the interval $\mathcal{D}_\ell$, we choose C as $C > \ell \cdot B^{1/S}$. Thus, for most choices of S and B we can let $C \approx \ell$. For fixed S, B and C, we can set the parameter κ to adjust the smallest number which can be represented by the scheme. For example, for parameter values of $C = 1, B = 2, S = 100$ and $\kappa = 16$ the smallest positive value that can be represented is $5.25 \cdot 10^{-198}$. In addition, κ influences the number of values from $\mathcal{D}_\ell$ which can be represented exactly. There are no general guidelines on how to choose the parameters; they must be chosen according to the problem domain. Usually, this process involves an experimental analysis of the propagation of errors during the computation.

IV-C. Arithmetic Operations

We will now describe how to perform basic arithmetic operations on encoded values $\overline{x}, \overline{y} \in \mathcal{L}_\kappa$.

Multiplication and Division. Due to the logarithmic representation it is quite easy to compute an encoding $\overline{xy}$ of the product $x \cdot y$, given encodings $\overline{x}$ and $\overline{y}$. Let $\tau_{C^2} = \lceil S \cdot \log_B(C) \rfloor$. The function

$$\begin{aligned} \textbf{LPROD}(\overline{x}, \overline{y}) &= \textbf{LPROD}((\rho_x, \sigma_x, \tau_x), (\rho_y, \sigma_y, \tau_y)) \\ &= (\rho_x \cdot \rho_y, \sigma_x \cdot \sigma_y, \tau_x + \tau_y - \tau_{C^2}) \\ &= \overline{xy} \end{aligned} \tag{2}$$

achieves the desired functionality. Note that $\textbf{LPROD}(\overline{x}, \overline{y}) = \overline{xy}$: If either $x = 0$ or $y = 0$, then also $\overline{xy}$ is an encoding of zero, and if both $x, y \neq 0$ then

$$\begin{aligned} \tau_{xy} &= \tau_x + \tau_y - \tau_{C^2} \\ &= -S \log_B(x/C) - S \log_B(y/C) - S \log_B(C) \\ &= -S \log_B(xy/C). \end{aligned}$$

Dividing two numbers $\overline{x}, \overline{y} \in \mathcal{L}_\kappa$ with $y \neq 0$ is similar:

$$\begin{aligned} \textbf{LDIV}(\overline{x}, \overline{y}) &= \textbf{LDIV}((\rho_x, \sigma_x, \tau_x), (\rho_y, \sigma_y, \tau_y)) \\ &= (\rho_x, \sigma_x \cdot \sigma_y, \tau_x - \tau_y + \tau_{C^2}) \\ &= \overline{x/y}. \end{aligned} \tag{3}$$

Addition and Subtraction. Given $\overline{x}$ and $\overline{y}$, computing $\overline{x+y}$ and $\overline{x-y}$, which will be denoted by **LSUM** and **LSUB** in the sequel, is more involved. We first note that in order to compute a subtraction, it suffices to swap the sign σ_y of $\overline{y}$ and then perform an addition. Thus we limit ourselves to describe the addition here. The new sign of σ_{x+y} of the value $\overline{x+y}$ depends on the values σ_x, σ_y as well as τ_x and τ_y. Let $\lambda_1, \lambda_2 \in \{0, 1\}$, where $\lambda_1 = 1$ if $\tau_x < \tau_y$ and $\lambda_1 = 0$ otherwise, and $\lambda_2 = 1$ if $\tau_x = \tau_y$ and $\lambda_2 = 0$ otherwise. Then $\sigma_{x+y} = \sigma_x + \lambda_1 \cdot (\sigma_y - \sigma_x)$. The new value ρ_{x+y} can be obtained by

$$\rho_{x+y} = (1 - \lambda_2(\sigma_x \sigma_y + 1)2^{-1})(\rho_x + \rho_y - \rho_x \cdot \rho_y). \tag{4}$$

For $x, y \neq 0$ with the same sign, τ_{x+y} can be computed using the *Kingsbury-Rayner-Formula* [21]:

$$\tau_{x+y} = \tau_y - S \cdot \lceil \log_B(1 + B^{(\tau_x - \tau_y)/S}) \rfloor. \tag{5}$$

If the signs of x and y differ, we have

$$\tau_{x-y} = \tau_y - S \cdot \lceil \log_B(1 - B^{(\tau_x - \tau_y)/S}) \rfloor. \tag{6}$$

This leads to the general formula for arbitrary $\overline{x}$ and $\overline{y}$:

$$\begin{aligned} \tau_{x \pm y} &= \rho_x \tau_x + \rho_y \tau_y + \rho_x \rho_y \\ &\qquad (-S \lceil \log_B(1 \pm B^{(\tau_x - \tau_y)/S}) \rfloor - \tau_x). \end{aligned} \tag{7}$$

IV-D. Secure Operations

In this section we describe how to perform secure computations on values that are represented by the encoding described in Section IV-A. In this paper, we assume a two-party scenario, where computation is interactively performed by parties $\mathcal{A}$ and $\mathcal{B}$. Party $\mathcal{A}$ has generated secret keys in the past and is thus the only party who has access to the private key. We encrypt each component of $\overline{x} \in \mathcal{L}_\kappa$ separately and denote the encryption $[\overline{x}]$ of $\overline{x}$ by $[\overline{x}] = ([\rho_x], [\sigma_x], [\tau_x])$.

Multiplication and Division. The arithmetic operations **LPROD** and **LDIV** can be computed in a straightforward manner by utilizing the properties of homomorphic encryption (see Section II) to implement Equations (2) and (3). The product operation requires two secure binary multiplications:

$$\begin{aligned} [\overline{xy}] &= \textbf{LPROD}([\overline{x}], [\overline{y}]) \\ &= ([\rho_x] \circledast [\rho_y], [\sigma_x] \circledast [\sigma_y], [\tau_x][\tau_y][\tau_{C^2}]^{-1}). \end{aligned}$$

For $y \neq 0$ the division can be computed by:

$$\begin{aligned} [\overline{x/y}] &= \textbf{LDIV}([\overline{x}], [\overline{y}]) \\ &= ([\rho_x], [\sigma_x] \circledast [\sigma_y], [\tau_x][\tau_y]^{-1}[\tau_{C^2}]). \end{aligned}$$

Subtraction. As noted above, the operation **LSUB** can be transformed into a sum **LSUM** by using the homomorphic properties of the encryption:

$$\begin{aligned} [\overline{x-y}] &= \textbf{LSUB}([\overline{x}], [\overline{y}]) \\ &= \textbf{LSUB}(([\rho_x], [\sigma_x], [\tau_x]), ([\rho_y], [\sigma_y], [\tau_y])) \\ &= \textbf{LSUM}(([\rho_x], [\sigma_x], [\tau_x]), ([\rho_y], [\sigma_y]^{-1}, [\tau_y])). \end{aligned}$$

Addition. In the following we describe the operation **LSUM** that securely adds two encrypted values $[\overline{x}]$ and $[\overline{y}]$. The flags $[\sigma_{x+y}]$ and $[\rho_{x+y}]$ of the result can be computed in a straightforward manner once encryptions $[\lambda_1]$ and $[\lambda_2]$ are available:

$$[\sigma_{x+y}] = [\sigma_x]([\lambda_1] * [\sigma_y - \sigma_x])$$

and

$$\begin{aligned} &[\rho_{x+y}] = \delta_1 * \delta_2, \qquad \text{where} \\ &\qquad \delta_1 = [1]([\lambda_2] * (([\sigma_x] * [\sigma_y])[1])^{2^{-1}})^{-1} \text{ and} \\ &\qquad \delta_2 = [\rho_x][\rho_y]([\rho_x] * [\rho_y])^{-1}, \end{aligned} \tag{8}$$

due to Equation (4). Encryptions of λ_1, λ_2 can be obtained by using standard methods for secure computations (see Section II).

The main difficulty when computing the sum $[\overline{x+y}]$ of two encrypted values $[\overline{x}], [\overline{y}]$ consists of computing τ_{x+y}. This can be achieved by first computing a value $[z] = [\tau_x - \tau_y]$ and then evaluating the function $f_+(z) = S \cdot \lceil \log_B(1 + B^{z/S}) \rfloor$ if $\sigma_x = \sigma_y$, according to Equation (5), and $f_-(z) = S \cdot \lceil \log_B(1 - B^{z/S}) \rfloor$ if $\sigma_x = -\sigma_y$, according to Equation (6). Note that z takes on values in the interval $\{-2^\kappa - 1, \ldots, 2^\kappa - 1\}$. The final encryption $[\tau_{x+y}]$ can then be computed as

$$[\tau_{x+y}] = ([\rho_x] * [\tau_x])([\rho_y] * [\tau_y])([\rho_x] \circledast [\rho_y]) * ([f(z)][\tau_x]^{-1}),$$

where

$$\begin{aligned} [f(z)] &= [(1 + \sigma_x \sigma_y) 2^{-1} f_+(z) + (1 - \sigma_x \sigma_y) 2^{-1} f_-(z)] \\ &= (([1]([\sigma_x] * [\sigma_y])^{2^{-1}}) * [f_+(z)]) \cdot \\ &\qquad (([1]([\sigma_x] * [\sigma_y])^{-1})^{2^{-1}} * [f_-(z)]), \end{aligned}$$

which directly evaluates Equation (7) on encrypted values. Thus, $[\tau_{x+y}]$ can be computed by secure multiplications once encryptions $[f_+(z)]$ and $[f_-(z)]$ are available. Computations for $[f_+(z)]$ and $[f_-(z)]$ have to be done in a way that no party gains any information about z or $f_\pm(z)$. Since both functions are very difficult to compute analytically on encrypted values, we compute them by running the lookup-table based primitive described in Section III.

Computations on positive values. In case that computations are performed only on positive values (such as probabilities in the interval $[0, 1]$), we can further simplify the representation of values, as σ_x will always be set to one. Thus, computing the new sign can be omitted, while other arithmetic operations can be simplified. For example, to compute an addition, the value ρ_{x+y} can be computed as $[\rho_{x+y}] = [\rho_x][\rho_y]([\rho_x] * [\rho_y])^{-1}$, requiring only one secure multiplication, instead of evaluating Equation (8), which requires 4 secure multiplications.

V. EXPERIMENTAL RESULTS

In this section we present experimental results to show that the framework introduced in Section IV yields efficient solutions that enable the processing of encrypted real-valued signals. To this end, we first consider the complexity of the basic operations and subsequently show that the framework can be used to filter digital signals privately and accurately.

V-A. Experimental Setup

The framework for privacy-preserving arithmetic operations, as described in Section IV, has been implemented in C++ using the GNU GMP library version 5.0.1, in order to determine performance and reliability. Tests were performed on a computer with two 2.1 GHz AMD Opteron quad-core processors and 4GB of RAM running Linux. Both parties $\mathcal{A}$ and $\mathcal{B}$ were modeled as different threads of one program, exchanging messages with each other; thus the reported computation complexities do not include network latency. To each thread we assigned some helper threads which allowed to distribute the computational load on multiple cores; in particular, independent operations (such as pre-computing randomness for Paillier encryption or tables for the **LSUM** operation) were performed in parallel, to obtain an equally distributed load on all cores.

V-B. Complexity of LSUM

Since computing the sum of two encoded elements is the most complex basic operation, we focus on an analysis of **LSUM**, implemented by the OPRF construction of Section III including all mentioned optimizations. Its memory usage and computation complexity is dominated by storing and computing the table $\mathcal{T}$, which grows linearly with the table size $|\mathcal{T}|$. For the experiment we measured the time required to set up a table for the OPRF construction (initialization step in Protocol 1) and perform 128 evaluations of the **LSUM** over the same table. Figure 1 depicts the amortized complexity of a single **LSUM** operation. The left axis shows the number of operations that can be performed within one second, depending on the size $|\mathcal{T}|$, where preparation of the OPRF table is amortized over all 128 invocations of **LSUM**. The the right axis plots the memory usage of one **LSUM** operation, where again the size of the table is amortized over the number of

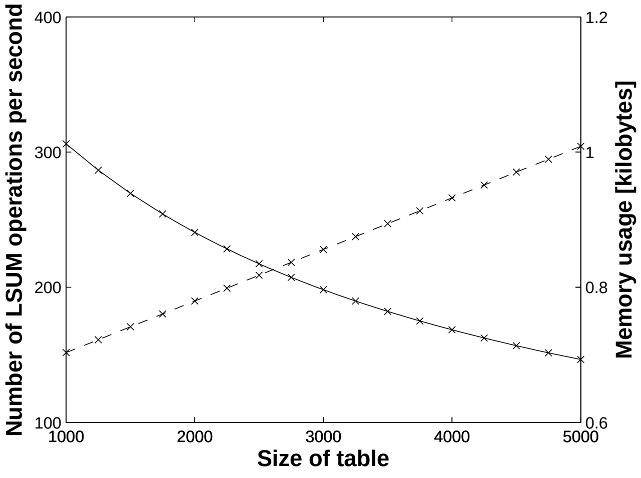

Fig. 1. Performance of the **LSUM** operation depending on the table size $|\mathcal{T}|$.

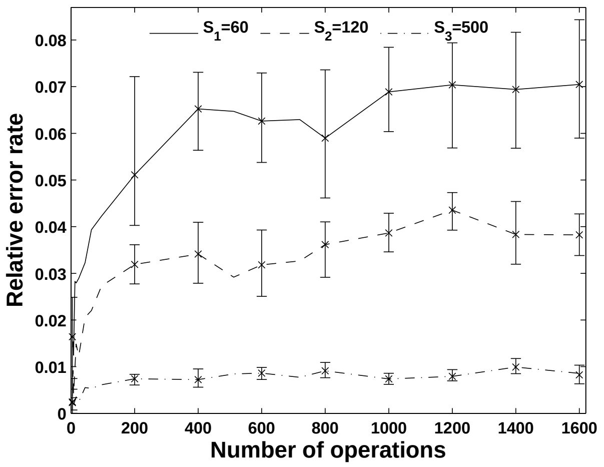

Fig. 2. Digital filter for different parameters S.

operations. We see that the latter indeed scales perfectly linear with the table size, while 147 to 306 secure **LSUM** operations can be performed within one second on the tested hardware platform.

Note that there is a tradeoff between accuracy and complexity. The number of values of the interval D_ℓ that can be represented without error in our encoding is determined by both the table size $|\mathcal{T}|$ and the parameter S. The larger this number, the better the accuracy, due to smaller rounding errors. At the same time, complexity of the **LSUM** operation grows linearly in $|\mathcal{T}|$.

V-C. Digital filters

In order to demonstrate the reliability of the proposed framework, we consider the use case of digitally filtering encrypted real-valued signals. In particular, we have implemented an averaging filter $f_n(x) = \sum_{i=1}^{n} \frac{1}{n} x_i$ and tested its performance when applied to signals whose components x_i are randomly chosen positive 8-bit numbers. We fixed the parameters of the number representation as $B = 2.71$ (i.e. we use the natural logarithm) and consider the cases of $S_1 = 60$, $S_2 = 120$ and $S_3 = 500$. For each parameter S_i, the table sizes $\mathcal{T}_1 = 500, \mathcal{T}_2 = 1200, \mathcal{T}_3 = 5000$ were chosen in a way that they can represent the smallest and biggest value occuring during the computation. Figure 2 depicts the relative error of the filtered signal coefficients, when compared to a standard floating point implementation: the x-axis depicts the length of the filter (and thus the number of dependent operations) and the y-axis plots the relative error. We repeated the experiment 20 times with randomly chosen signals and depict the average relative error as well as the maximum and minimum values. The experiments show that for even small values such as $\frac{1}{n} = \frac{1}{800} = 1.25 \cdot 10^{-3}$ accurate results can be achieved (with a relative error of less than 1% for $S = 500$).

VI. CONCLUSIONS

In this paper we presented for the first time a framework that allows to compute securely with values taken from a real interval in a numerically stable way. To be able to process both very small and very large values, we chose a logarithmic encoding and provided secure two-party protocols to perform all arithmetic operations on encoded values in an oblivious way. To demonstrate its practicality, we implemented the framework and performed measurements to show that the most complex operation (the addition) can be performed in a reasonable amount of time; furthermore we analyzed the propagation of quantization errors when applied to the problem of privately filtering signals.

ACKNOWLEDGEMENTS

This work was supported in part by the German Research Foundation (DFG) and the Center for Advanced Security Research Darmstadt (CASED). Kay Hamacher gratefully acknowledges financial support by Fonds der chemischen Industrie.

VII. REFERENCES

[1] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel, “Privacy-preserving remote diagnostics,” in *ACM Conference on Computer and Communications Security*, 2007, pp. 498–507.

[2] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft, “Privacy-preserving face recognition,” in *Privacy Enhancing Technologies Symposium (PET)*, 2009, pp. 235–253.

[3] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen, “Multiparty computation from threshold homomorphic encryption,” in *EUROCRYPT*, 2001, pp. 280–299.

[4] A. C.-C. Yao, “Protocols for Secure Computations (Extended Abstract),” in *Annual Symposium on Foundations of Computer Science — FOCS ’82*. November 3-5, 1982, pp. 160–164, IEEE.

[5] O. Goldreich, S. Micali, and A. Wigderson, “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority,” in *ACM Symposium on Theory of Computing — STOC ’87*. May 25-27, 1987, pp. 218–229, ACM.

[6] M. Jacobsson and A. Juels, “Mix and match: Secure function evaluation via ciphertexts,” in *Advances in Cryptology – ASIACRYPT’00*, T. Okamoto, Ed. 2000, vol. 1976 of *LNCS*, pp. 162–177, Springer-Verlag.

[7] Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh, “Secure function evaluation with ordered binary decision diagrams,” in *ACM Conference on Computer and Communications Security*, 2006, pp. 410–420.

[8] M. Naor and K. Nissim, “Communication preserving protocols for secure function evaluation,” in *ACM Symposium on Theory of Computing*, 2001, pp. 590–599.

[9] M. Naor and K. Nissim, “Communication complexity and secure function evaluation,” *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 8, no. 062, 2001.

[10] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers, “Cryptocomputing with rationals,” in *Financial Cryptography*, 2002, pp. 136–146.

[11] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Financial Cryptography*, 2010.

[12] Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft, "A practical implementation of secure auctions based on multiparty integer computation," in *Financial Cryptography*, 2006, pp. 142–147.

[13] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen, "On secure scalar product computation for privacy-preserving data mining," in *7th ICISC*. 2004, vol. 3506 of *LNCS*, pp. 104–120, Springer.

[14] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, New York, NY, USA, 2005, pp. 593–599, ACM Press.

[15] J. F. Canny, "Collaborative filtering with privacy.," in *IEEE Symposium on Security and Privacy*, 2002, pp. 45–57.

[16] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider, "Secure evaluation of private linear branching programs with medical applications," in *14th European Symposium on Research in Computer Security (ESORICS)*, 2009, pp. 424–439.

[17] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. May 2-6, 1999, vol. 1592 of *LNCS*, pp. 223–238, Springer.

[18] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, Joe Kilian, Ed. 2005, vol. 3378 of *Lecture Notes in Computer Science*, pp. 303–324, Springer.

[19] Carmit Hazay and Yehuda Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*. 2008, vol. 4948 of *Lecture Notes in Computer Science*, pp. 155–175, Springer.

[20] Stanislaw Jarecki and Xiaomin Liu, "Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection," in *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, Omer Reingold, Ed. 2009, vol. 5444 of *Lecture Notes in Computer Science*, pp. 577–594, Springer.

[21] P.J.W. Kingsbury, N.G. Rayner, "Digital filtering using logarithmic arithmetic," *Electronics Letters*, pp. 56–58, 1971.