Conversion of Real-Numbered Privacy-Preserving Problems into the Integer Domain

Wilko Henecka, Nigel Bean, and Matthew Roughan

School of Mathematical Sciences, University of Adelaide, Australia
{wilko.henecka,nigel.bean,matthew.roughan}@adelaide.edu.au

Abstract. Secure Multiparty Computation (SMC) enables untrusting parties to jointly compute a function on their respective inputs without revealing any information but the outcome. Almost all techniques for SMC support only integer inputs and operations. We present a *secure scaling* protocol for two parties to map real number inputs into integers without revealing any information about their respective inputs. The main component is a novel algorithm for privacy-preserving random number generation. We also show how to implement the protocol using Yao's garbled circuit technique.

1 Introduction

For the last 30 years the field of privacy-preserving techniques for distributed computation, also called *Secure Multiparty Computation* (SMC), has been growing. It offers solutions for multiple parties to compute functions without revealing their respective inputs to each other. These techniques have come a long way from the first theoretical ideas to practical solutions for problems such as electronic voting, auctions, data mining, network management and optimisation.

Almost all secure multiparty computation techniques have a message space consisting of a finite set of integers and the operations they provide are only defined over the integers. What if you want to engage in a privacy preserving protocol with real numbers, or floating point approximations? You can either extend a SMC technique to support fixed-point [1] or floating-point [2,3] arithmetic, or you create a mapping from the inputs into the integer space and then use conventional SMC [4–6]. The first approach introduces more complexity and limits the choice of techniques to just a few, the latter raises an interesting privacy question: *How do you agree on a mapping without revealing information about the inputs?*

In this paper we present the first secure scaling protocol for two parties. It enables them to agree on a mapping (by scaling) in a privacy-preserving manner. The key building block for this protocol is a novel algorithm for privacy-preserving random number generation. We also provide an efficient implementation of the protocol.

T.W. Chim and T.H. Yuen (Eds.): ICICS 2012, LNCS 7618, pp. 131–141, 2012.

2 Secure Multiparty Computation

Secure Multiparty Computation (SMC) protocols enable parties to carry out distributed computation tasks without having to reveal their inputs to each other. The most famous example is the *millionaires problem*: Two millionaires want to find out who is richer without revealing their actual wealth to each other. When SMC was introduced in 1982 by Yao [7], he used this example as a motivation.

Yao's Garbled Circuits. The earliest generic solution for SMC was proposed by Yao in 1986 [8]. It is a constant-round protocol for securely computing a two-party function while at the same time keeping the inputs private. Let Alice and Bob be two parties holding the inputs $x^{(A)}$ and $x^{(B)}$ respectively and f be a polynomial-time function. The first step is to view f as a Boolean circuit C.

Boolean Circuit: A Boolean circuit consists of wires and gates. The wires transmit a value $\{0,1\}$ and the gates compute a Boolean function on their input wires, and output the result to another wire. This wire may then be connected to the input of another gate or be an output value of the circuit (Figure 1). Mathematically we describe a circuit by a series of functions $g_i(\alpha,\beta)$, $\alpha,\beta \in \{0,1\}$, $g_i : \{0,1\}^n \to \{0,1\}$.

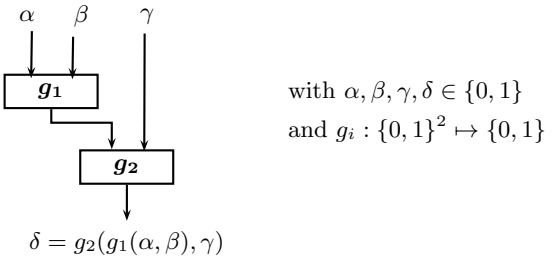

Fig. 1. A Boolean circuit consisting of 2 two-input gates

Once the input wires to a gate are given values α,β, it is possible to compute $g_1(\alpha,\beta)$ and assign it to the output wire which becomes an input to $g_2(\cdot,\cdot)$, etc. The output of the circuit is given by the values of the output wires of the circuit. Thus, computing the circuit C is essentially just allocating appropriate Boolean values to all wires of the circuit.

Privacy: The values for some wires are provided by Alice, and others by Bob. These represent private inputs that should not be leaked to the other party. Likewise all intermediate values have to remain hidden, since they could reveal information about the inputs. The only values learned should be the outputs. The protocol works by having one party (say Alice) first generate a *garbled* version of the circuit and then send it to Bob. To create the garbled circuit Alice

first assigns random labels for the 0 and 1 states of all wires. She then uses these labels as keys to encrypt the truth-tables of the gates and finally sends the encrypted values to Bob. In this form it doesn't leak any information to Bob. However, he can obtain the output of the circuit by decrypting it, using the labels given to him by Alice. In order to ensure that Bob learns nothing more than the output itself, Bob is only given the labels for the actual input values (not all possible inputs). He receives the labels for his input by running an oblivious transfer protocol with Alice (See [9] for further details).

Security Model: Yao's protocol for SMC is secure in the *semi-honest* model, *i.e.,* parties are assumed to correctly follow the protocol, and there is no efficient adversary that can extract more information from the transcript of the protocol execution than is revealed by that party's private input and the result of the function. There are also extensions to the protocol which are secure against certain types of active adversaries: (See Lindell and Pinkas [9] and the citations therein).

Practicality: Recent contributions [10, 11] improved the efficiency of implementations of Yao's protocol significantly.

Over the past few years several implementations for generic secure two-party computation using garbled circuits have been developed [11–14]. They differ in abstraction level, supported optimisation techniques and efficiency. We use [14] because it allows construction of dynamic loops.

There are other protocols for secure multiparty computation, varying in assumptions, security guaranties, number of supported parties, performance and supported operations (see [15] for an overview). However, our protocol translates naturally into a Boolean circuit.

3 Secure Scaling

Almost all secure multiparty computation techniques support only integers as inputs and operations on the integers. To engage in a privacy preserving protocol having real numbers, or floating point approximations, as inputs, you can define a mapping from the real inputs into the integer space and then use conventional SMC.

The obvious trivial approach to map real numbers to integers is scaling and quantisation. Let $r \in \mathbb{R}$ be the real number input. Then $i = \lceil s \cdot r \rfloor$ is the mapping from r to i, where $\lceil \cdot \rfloor$ is the function that rounds to the nearest integer, and s is a scaling factor the parties agree on.

It is easy to see that the scaling factor leaks information about the inputs, since it has to be chosen such that all inputs are mapped into the finite set and are still distinguishable. Each party can support a different set of scaling factors, depending on their respective inputs. Revealing these sets to each other leaks information. They want to agree on a scaling factor without having to reveal any information about their supported sets other than they contain the chosen scaling factor.

We propose a *Secure Scaling* protocol to pick a scaling factor at random out of the intersection of ranges given by two parties. The basic idea is to first compute a secure set intersection and then, without revealing the intersection, pick an element at random (see Section 4.4). While secure set intersection protocols are readily available we propose the first protocol we know of to draw a random number from a private range.

4 Drawing Random Numbers from a Private Range

We don't want to reveal the range of the scaling factors, once it is computed with the privacy-preserving set-intersection protocol. Instead we keep it in the encrypted space and use it as the input to the random number algorithm. The goal of this algorithm is to pick an element uniformly at random out of the range without giving the participants any more information than the randomly drawn element itself.

We first show the simple case where the range starts with 0 and has a power of two elements. Then we allow for an arbitrary number of elements, still starting with 0, and finally we present the algorithm where both bounds of the range are arbitrary values.

4.1 Range $N_{2^m-1} = \{0, 1, 2, \ldots, 2^m - 1\}$

The set $N_{2^m-1} = \{0, 1, 2, \ldots, 2^m-1\}$ is the set of integers that can be represented by an m-bit number. If we choose m random bits, each with probability $1/2$, the binary number denoted by these bits will be uniformly distributed over N_{2^m-1}. The algorithm combines random bits chosen by both parties, and then chooses m of these.

Let the private input m come from a finite set $I = \{0, 1, \ldots, n\}$, which is agreed on by both parties, and, $N_{2^n-1} = \{0, 1, \ldots, 2^n - 1\}$ be the set of all n-bit integers. Now both parties choose $r^{(A)}, r^{(B)} \in_R N_{2^n-1}$, respectively, where $\in_R$ means chosen uniformly at random from the set. The algorithm first combines the random n-bit inputs by the bitwise exclusive OR operation (XOR) to get r, and then selects the m least significant bits of r by computing the output $x = r \bmod 2^m$.

Algorithm 1. *urandom1:* Drawing x randomly from $\{0, 1, 2, \ldots, 2^m - 1\}$

Inputs: (private) $m \in I$
Outputs: $x = \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$
  $r \leftarrow r^{(A)} \,\text{XOR}\, r^{(B)}$ $\{r^{(A)}, r^{(B)} \in_R N_{2^n-1}$, where $r^{(i)}$ is provided by party $i\}$
  $x \leftarrow r \bmod 2^m$
  return x

Correctness: It is easy to see that $x \in \{0, 1, \ldots, 2^m - 1\}$ since that is exactly the co-domain of $r \bmod 2^m$. We also have to show that x is a uniformly distributed random variable over that range.

Lemma 1. *If at least one of $r^{(A)}$ and $r^{(B)}$ is chosen uniformly at random out of $N_{2^n-1} = \{0, 1, \ldots, 2^n - 1\}$ then $r = r^{(A)} \operatorname{XOR} r^{(B)}$ is a random number uniformly distributed over N_{2^n-1}.*

Proof. Let $N_{2^n-1} = \{0, 1, 2, \ldots, 2^n - 1\}$ be the set of all integers that can be represented by n bits. If $r^{(A)}$ is a random variable on N_{2^n-1}, then there exists a unique random vector $(r_1^{(A)}, \ldots, r_n^{(A)})$ on $\{0, 1\}^n$ such that $r^{(A)} = \sum_{i=1}^{n} 2^{i-1} r_i^{(A)}$. If $r^{(A)}$ is uniformly distributed over N_{2^n-1} then the $r_i^{(A)}$'s are mutually independent Bernoulli random variables with parameter $1/2$. $r = r^{(A)} \operatorname{XOR} r^{(B)}$ can now be written as $r = \sum_{i=1}^{n} 2^{i-1} r_i$ with $r_i = r_i^{(A)} \operatorname{XOR} r_i^{(B)}$. Note that the XOR operation returns 1 iff both arguments are different.

Assume that $r^{(A)}$ is uniformly distributed. Therefore

$$\Pr[r_i = 1 | r_i^{(B)} = 0] = \Pr[r_i^{(A)} = 1] = 1/2$$
$$\Pr[r_i = 1 | r_i^{(B)} = 1] = \Pr[r_i^{(A)} = 0] = 1/2.$$

Note that the value of $r_i^{(B)}$ has no influence on $\Pr[r_i = 1]$. Thus $\Pr[r_i = 1] = \Pr[r_i = 0] = 1/2$. XOR is a bitwise operation and the r_i are mutually independent and thus r is uniformly distributed over N_{2^n-1}. □

Now $x = r \bmod 2^m$ can be rewritten as $x = \sum_{i=1}^{m} 2^{i-1} r_i$ since the $\bmod 2^m$ operation selects the m least significant bits of r. The r_i are mutually independent Bernoulli random variables with parameter $1/2$, hence x is a random variable uniformly distributed over $\{0, 1, \ldots, 2^m - 1\}$.

Security: We want to keep the input m private. We will ensure that the parties don't learn it using the garbled circuit technique (see Section 5). What's left to show is that neither party can choose their input to manipulate the output. A successful attack would distort the uniform distribution of the output. However, we know from Lemma 1 that the output is uniformly distributed as long as at least one input is uniformly distributed. So even if party A (or party B) deviates from the protocol and deliberately chooses a specific value for $r^{(A)}$ (or $r^{(B)}$) the output will remain uniformly distributed.

4.2 Range $N_q = \{0, 1, \ldots, q\}$

In this section, we relax the restriction that the size of the range must be an exact power of two. Now we allow any range $N_q = \{0, 1, \ldots, q\}$ with $q \in \mathbb{N}$. The number of elements in that range is not necessarily a power of two and therefore we can't directly apply Algorithm 1.

We use the acceptance-rejection method to constructing a Las Vegas type algorithm that uses Algorithm 1 repeatedly until it produces a value in the required range. To do this, we extend N_q to N_{2^m-1} so that it is of the form of Algorithm 1. That is, we choose the unique $m \in \mathbb{N}$ with $2^{m-1} - 1 < q \leq 2^m - 1$, and then run the algorithm as described in Algorithm 2. This approach translates naturally into a compact circuit with a number of gates that is linear in the input size.

Algorithm 2. *urandom2:* Drawing x randomly from $\{0, 1, \ldots, q\}$

Inputs: (private) $q \in N_{2^n-1}$
Outputs: $x = \text{urandom2}(\{0, 1, \ldots, q\})$
  $m \leftarrow \lfloor log_2(q) + 1 \rfloor$
  repeat
    $x \leftarrow \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$
  until $x \leq q$
  return x

Correctness: When the algorithm terminates $x \in_R \{0, 1, \ldots, q\}$ since the exit condition ensures $x \leq q$, and *urandom1* produces non-negative numbers, and x is uniformly distributed since acceptance-rejection sampling of a subset of a uniform distribution is again uniformly distributed.

The number of iterations of the loop follows a geometric distribution. Let X be a random variable describing how many iterations Algorithm 1 takes to get a valid result. The probability that $X \leq k$ with $k \in \mathbb{N}$ is

$$\Pr[X \leq k] = 1 - (1 - \Pr[x \leq q])^k.$$

The probability that the exit condition is fulfilled in one iteration is

$$\Pr[x \leq q] \geq \frac{2^{m-1} + 1}{2^m} > \frac{1}{2},$$

because $2^{m-1} - 1 < q \leq 2^m - 1$, and so $\Pr[X \leq k] > 1 - (1/2)^k$.

That means that even in the worst case the expected number of iterations is less than 2, and the probability of less than 10 iterations is greater than 99.9%. We illustrate the performance in Section 5.2.

Security: Again, neither party can distort the uniform distribution of the random value by the same argument as for Algorithm 1.

4.3 Range $N_{p,q} = \{p, p + 1, \ldots, q\}$

In the most general case where the range is arbitrary we first shift it to zero and then use Algorithm 2 to compute a random value and finally shift it back to the initial range (See Algorithm 3 for details).

Algorithm 3. *urandom3:* Drawing x randomly from $\{p, p+1, \ldots, q\}$

Inputs: (private) $p, q \in N_{2^n}$
Outputs: $x = \text{urandom3}(\{p, p+1, \ldots, q\})$
  $m \leftarrow \lfloor \log_2(q-p) + 1 \rfloor$
  repeat
    $s \leftarrow \text{urandom1}(\{0, 1, \ldots, 2^m - 1\})$
  until $s \leq q - p$
  return $x = s + p$

Correctness and Security: Correctness and security follow from the same arguments as for Algorithm 2.

4.4 Secure Scaling

Once we have a random number generator, we can build an efficient solution for the secure scaling problem.

Both parties input their smallest $(p^{(A)}, p^{(B)})$ and biggest $(q^{(A)}, q^{(B)})$ possible scaling factors. The first step is to determine the intersection of these ranges by computing the boundaries of the intersection as $p = \max(p^{(A)}, p^{(B)})$ and $q = \min(q^{(A)}, q^{(B)})$. In the second step we use the random number generator to select an element out of $\{p, p+1, \ldots, q\}$.

Algorithm 4. The Secure Scaling algorithm

Inputs: $p^{(A)}, q^{(A)}, p^{(B)}, q^{(B)} \in N_{2^n - 1}$
Outputs: $s \in_R \{p^{(A)}, p^{(A)} + 1, \ldots, q^{(A)}\} \cap \{p^{(B)}, p^{(B)} + 1, \ldots, q^{(B)}\}$
  $p \leftarrow \max(p^{(A)}, p^{(B)})$
  $q \leftarrow \min(q^{(A)}, q^{(B)})$
  $s \leftarrow \text{urandom3}(\{p, p+1, \ldots, q\})$
  return s

Correctness and Security: Correctness and security follow from the same arguments as for Algorithm 2. In the following section we show how to implement all of the steps needed in Algorithm 4 using garbled circuits.

5 Secure Scaling with Boolean Circuits

We compute the secure scaling algorithm with Yao's garbled circuit technique by expressing it as a Boolean circuit. Boolean circuits are easily combined, so we will show the subcircuits corresponding to the elementary operations in the algorithm.

We will describe the complexity of each subcircuit by the number of non-linear two-input gates in relation to the number of bits l needed to represent the inputs $p^{(A)}, p^{(B)}, q^{(A)}, q^{(B)}$. A linear input gate has an even number of zeros and ones in the truth table. The linear gates for a constant output or the (negated) identity of an input wire can be trivially optimised away, e.g. XOR gates can be evaluated essentially for free [16], therefore the dominating factor for efficiency of the circuits is the number of non-linear gates.

- $(\min(q^{(A)}, q^{(B)}), \max(p^{(A)}, p^{(B)}))$: To compute these we use the integer comparison circuit described by Kolesnikov *et al.* [17], it has a complexity of l non-linear gates.
- $(m \leftarrow \lfloor \log_2(q-p)+1 \rfloor)$: In this step we don't actually have to compute m, because all we need later on is a bit mask to select the $\lfloor \log_2(q-p)+1 \rfloor$ least significant bits. Therefore we first compute $t = q - p$ with the integer subtraction circuit described in [17] and then we use a chain of OR-gates (see Figure 2) to calculate the mask $2^{\lfloor (\log_2(t)+1 \rfloor} - 1$. This circuit consists of $l - 1$ non-linear gates.

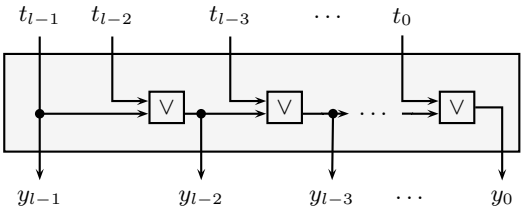

Fig. 2. A chain of OR gates to compute $y = 2^{\lfloor \log_2(t)+1 \rfloor} - 1$

- $(r = r^{(A)} \text{ XOR } r^{(B)})$: r is just a bitwise XOR between $r^{(A)}$ and $r^{(B)}$. Therefore the complexity is 0 non-linear gates.
- $(s = r \bmod 2^m)$: Computing modulo a power of two is the special case where we just want to select the m least significant bits of r. We can achieve this by computing a bitwise AND between r and $2^m - 1$, the bit-mask with the m least significant bits set to 1. This is exactly the bit-mask we computed before. The complexity is l non-linear gates.
- (repeat until $s \leq q - p$): Note that this loop has an unknown number of iterations, therefore it is impossible to generate the whole circuit to compute the loop beforehand. However, in this case, where the exit condition of the loop does not reveal any sensitive information, we can use a step-by-step approach. That is, the creator generates the circuit for one round of the loop and then the evaluator evaluates the circuit and reveals the result of the exit condition. Depending on that result the creator then generates either another round of the loop or goes on with the rest of the algorithm. Note that the disclosure of the result of the exit condition gives neither party an advantage in inferring the other parties input as long as their random inputs are kept private. This privacy is guaranteed by the garbled circuit technique.
  For the exit condition we can reuse $q - p$ which we computed before. Thus we only need an integer comparison circuit [17] which has a complexity of l non-linear gates.
- $(x = s + p)$: We use the addition circuit of [17] to compute this sum. Again, the complexity is l non-linear gates.

Overall Complexity: Let X describe the number of iterations of the repeat-until loop in Algorithm 2. Then the number of non-linear gates add up to $2l+2l-1+X(2l)+l = 5l-1+2lX$. Since X follows a geometric distribution with success probability $1/2 < p \leq 1$, we know that $1 \leq \mathbb{E}[X] < 2$, thus the expected overall complexity is less than $9l$.

5.1 Implementation

We chose the EFSFE framework of Henecka and Schneider [14] to implement our example of the random scaling factor. Amongst other optimisation techniques used in this framework the following are particularly useful for our application:

- Pipelined circuit execution: The circuit generation and evaluation processes are overlapped in time [11] thereby removing the need to construct the complete circuit before the evaluation, which is useful here because we cannot build the circuit in advance, since the number of iterations is dynamic.
- Oblivious-transfer extension: In [18], Ishai *et al.* show how to efficiently extend Oblivious transfer. You first have to execute a certain amount of conventional OTs and then by using this result you can generate a virtually unlimited number of very efficient OTs. (The initial OTs take ~ 0.5 s, and then every additional OT takes only 3.5 μs).

The EFSFE framework contains a library of circuits for common arithmetic which can be easily combined to describe the desired function. You can combine circuits from and add circuits to the library by extending the *CompositeCircuit* class. For example, the implementation of the chain of OR-gates circuit as shown in Figure 2 is done by defining subcircuits and connecting them with wires as follows:

```
public class NextBitMask extends CompositeCircuit {
  protected void createSubCircuits() throws Exception {
    for(int i=0; i<l-1; i++){
      subCircuits[i] = OR_2_1.newInstance();
    }
    super.createSubCircuits();
  }
  protected void connectWires() throws Exception {
    for(int i=0; i<l-1; i++){
      inputWires[i].connectTo(subCircuits[i].inputWires, 0);
    }
    inputWires[l-1].connectTo(subCircuits[l-2].inputWires, 1);
    for(int i=0; i<l-2; i++){
      subCircuits[i+1].outputWires[0].connectTo(
                                subCircuits[i].inputWires, 1);
    }
  }
  protected void defineOutputWires() { ... }
```

5.2 Measurements

All our measurements were run on an iMac with a Core i3 3Ghz processor, running Mac OS X 10.6.8 and Java 1.6.0_31. We ran measurements for four different input sizes (10, 100, 1000 and 10000 bits). For each size we ran the secure scaling algorithm 10000 times with inputs $(p^{(A)}, p^{(B)}, q^{(A)}, q^{(B)})$ generated uniformly at random from the set of non-negative integers able to be represented by the given number of bits. The resolution of the measurements is 1 ms, therefore the data points for the 10 bit input size are not very precise and only included in the graph to underline the overall trend. Figure 3 shows the distributions of the runtimes for the different input sizes. The single red line denotes the median, the blue box include the data points from the 25th to the 75th percentile and the whiskers include all points up to 1.5 times the size of the blue box. The linear circuit complexity with respect to input bit lengths is clear. Note also the very strong right skewness of the data.

Figure 4 shows the complementary cumulative distribution functions of the number of iterations for different input sizes. That is the probability that a run has more than X iterations. We also added the worst case scenario for 1000 bit inputs, that is the inputs are chosen such that the private range is 2^{999} and therefore the probability that the exit condition of the loop is fulfilled is $(2^{999}+1)/2^{1000} \approx 1/2$. We see that the input size has little effect on the distribution. Even for the worst case the probability for a high number of iterations drops rapidly.

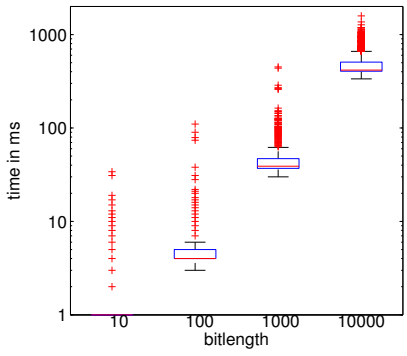

Fig. 3. Runtime distributions of the secure scaling algorithm for different input sizes

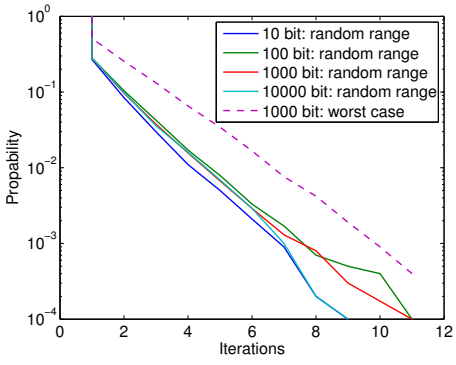

Fig. 4. Complementary cumulative distribution functions of the number of iterations for different input sizes

6 Conclusions

This paper presents a protocol to solve the secure scaling problem. Its main component is, to our knowledge, the first privacy-preserving random number generator. We believe that it might be a useful component for other privacy-preserving protocols. We show the practicality of our solution by an implementation of the protocol.

Acknowledgement. The authors would like to acknowledge the support of an Adelaide Scholarship International, a supplementary Scholarship of the Defence Systems Innovation Centre, and Australian Research Council grant DP0985063.

References

1. Catrina, O., Saxena, A.: Secure Computation with Fixed-Point Numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (2010)
2. Fouque, P., Stern, J., Wackers, G.: Cryptocomputing with Rationals. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 136–146. Springer, Heidelberg (2003)
3. Franz, M., Deiseroth, B., Hamacher, K., Jha, S., Katzenbeisser, S., Schroeder, H.: Secure computations on Non-Integer values. Technical report (2010)
4. Nguyen, H., Roughan, M.: Multi-Observer privacy preserving hidden markov models. In: IEEE/IFIP NOMS, pp. 514–517 (2012)
5. Blanton, M., Aliasgari, M.: Secure computation of biometric matching. Technical Report CSE Technical Report 2009-03, University of Notre Dame (April 2009)
6. Bianchi, T., Piva, A., Barni, M.: On the implementation of the discrete fourier transform in the encrypted domain. IEEE Transactions on Information Forensics and Security, 86–97 (March 2009)
7. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160–164 (1982)
8. Yao, A.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE (October 1986)
9. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
10. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
11. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security Symposium (2011)
12. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - a secure two-party computation system. In: USENIX Security Symposium (2004)
13. Henecka, W., Kögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: tool for automating secure two-party computations. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, pp. 451–462 (2010)
14. Henecka, W., Schneider, T.: EFSFE: Even faster secure function evaluation (submission, 2012)
15. Frikken, K.: Secure multiparty computation. In: Algorithms and Theory of Computation Handbook, 2nd edn., pp. 1–16. Chapman & Hall/CRC (2009)
16. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
17. Kolesnikov, V., Sadeghi, A., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 1–20. Springer, Heidelberg (2009)
18. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)