Performance Comparison of Secure Comparison Protocols

Florian Kerschbaum, Debmalya Biswas
SAP Research
Vincenz-Priessnitz-Strasse 1
Karlsruhe, Germany
Email: firstname.lastname@sap.com

Sebastiaan de Hoogh
Dept. of Mathematics and Computer Science
TU Eindhoven
Eindhoven, The Netherlands
Email: s.j.a.d.hoogh@tue.nl

Abstract

Secure Multiparty Computation (SMC) has gained tremendous importance with the growth of the Internet and E-commerce, where mutually untrusted parties need to jointly compute a function of their private inputs. However, SMC protocols usually have very high computational complexities, rendering them practically unusable. In this paper, we tackle the problem of comparing two input values in a secure distributed fashion. We propose efficient secure comparison protocols for both the homomorphic encryption and secret sharing schemes. We also give experimental results to show their practical relevance.

1. Introduction

Secure Multiparty Computation (SMC) allows a number of mutually distrustful parties to carry out a joint computation of a function of their inputs, while preserving the privacy of the inputs. SMC protocols have widespread applications [4, 5] in various real-life problems, such as matchmaking, electronic voting, private bidding and auctions, etc.

A great deal of work (e.g., [1, 2, 3, 7]) has already been done in this area. It has been shown that any function can be securely computed using generic circuit based protocols [2, 13]. However, the general protocols tend to be very inefficient, hence our main objective is to design efficient protocols for specific functions.

In this work, we focus on the secure comparison function. Given a pair of input values x_0 and x_1, the objective is to compute the value of the boolean expression $[x_0 \leq x_1] \in \{0, 1\}$, while ensuring that none of the parties gain knowledge of the actual input values. We further require the outputs of our protocols to be private as well, i.e. the output bit is either encrypted or shared among the parties.

We propose efficient secure comparison protocols for both the homomorphic encryption (Section 2.1) and secret shares (Section 2.2) based SMC schemes. Both the schemes have their pros and cons when it comes to securely computing basic arithmetic operations, such as addition and multiplication. Homomorphic encryption schemes allow arithmetic operations to be performed locally on the plaintext values, based on their encrypted values (ciphertext). The fact that the operations are actually performed on the encrypted data leads to their performance being dependent on the size of the key sizes used for encryption. In secret sharing schemes on the other hand, while addition can be performed locally by an addition of the local (plaintext) shares, multiplication requires distributed collaboration among the parties. Thus, it is difficult to theoretically compare the performance of protocols based on the two schemes. To overcome this, we have implemented both the protocols and present experimental results in Section 2.3.

2. Secure Comparison Protocols

Let n be a modulus and we have a threshold cryptosystem that has number field $\mathbb{Z}_n$. In a modulo field, negative numbers are represented by the upper half of the range $[0, n-1]$:

$$[\left\lceil \frac{n}{2} \right\rceil, n-1] \equiv [-\left\lfloor \frac{n}{2} \right\rfloor, -1]$$

To compute $x_0 \leq x_1$, we compute $d = x_0 - x_1$ and see whether d is positive. The basic idea of our protocols is to hide the difference d by multiplicative hiding. Multiplicative hiding consists of hiding a value d by multiplying it with a much larger random number r. In order to prevent factoring of the result [6], we additionally subtract a smaller random value $r' < r$, since otherwise the hidden value would consist of the factors of the result. We thus assume the existence of random values r and r', such that $rd - r'$ is statistically close to uniform on some set containing all possible values of d.

We need n to be odd and n should satisfy the following property:

$$log_2(n) > k + l + 2$$

for l-bit input values and k bit random numbers.

2.1. Comparison Protocol based on Homomorphic Encryption

The first protocol is based on a threshold, homomorphic encryption system. Let $E()$ and $D()$ denote encryption and decryption, respectively, in the homomorphic encryption system. We require the homomorphic property to allow (modular) addition of the plaintexts. It then holds that

$$D(E(x)E(y)) = x + y$$

From which by simple arithmetic it follows that

$$D(E(x)^y) = xy$$

The homomorphic encryption system is public-key, i.e. any party can perform the encryption operation $E()$ (by itself). The ciphertexts are semantically secure, i.e. their ciphertext reveals nothing about the plaintext. More precisely, the ciphertexts are indistinguishable under chosen plaintext attack (IND-CPA). This implies an important property of re-randomization where an input ciphertext is modified, such that it cannot be linked to its original anymore without modifying the plaintext. In our encryption system, this is best performed by “adding 0”: $E(x)E(0) = \widehat{E}(x)$, but $E(x)$ and $\widehat{E}(x)$ are indistinguishable.

In a threshold encryption system the decryption key is replaced by a distributed protocol. Let m be the number of parties. Only if $t \leq m$ or more parties collaborate they can perform a decryption. No coalition of less than t parties can decrypt a ciphertext. We require a collaboration of all parties, i.e. $t = m$ (since we operate in the semi-honest model and do not consider faults). Then a ciphertext can only be decrypted if all the parties collaborate.

An encryption system satisfying all our requirements has been described in [7], a variation of [8].

2.1.1 Protocol

Let x_0 and x_1 be the numbers to compare. The inputs to the protocol are then ciphertexts of x_0 and x_1 (values encrypted under $E()$), i.e. all the m parties receive $E(x_0)$ and $E(x_1)$ as input. The output of the protocol is an encrypted bit indicating the result of the comparison $x_0 \leq x_1$. The output is thus secret.

Given this, the protocol proceeds as follows:

1. Party X_1 computes the following value c. He randomly chooses very large random numbers r_1 and r'_1 as discussed earlier (that can multiplicatively hide $x_0 - x_1$). He then computes

$$\begin{aligned} E(c) &= (E(x_0)E(-x_1))^r E(-r') \\ &= E(r(x_0 - x_1) - r') \end{aligned}$$

   Note that X_1 can compute the negation of the plaintext of $E(x_1)$ by computing the multiplicative inverse of the ciphertext, i.e. he can compute it without knowing the modulus of the homomorphic operation, which would reveal the secret key in our encryption system [7].

2. Party X_1 sends $(a_1^1, a_2^1, a_3^1) = (E(1), E(0), E(c))$ to X_2.

3. Each party X_i ($k = 2, \ldots, m$) re-randomizes the ciphertexts, i.e. selects two very large random numbers $r'_i < r_i$, flips a coin $b_i \in \{0, 1\}$, and sends (a_1^i, a_2^i, a_3^i) to $X_{i+1}(X_m$ sends to $X_1)$, where

$$\begin{aligned} a_1^i &= a_{1+b_i}^{i-1} E(0) \\ a_2^i &= a_{2-b_i}^{i-1} E(0) \\ a_3^i &= (a_3^{i-1})^{(-1)^{b_i} r_i} E((-1)^{(1-b_i)} r'_i) \end{aligned}$$

4. All parties X_i ($i = 1, \ldots, m$) collaboratively engage in the decryption protocol and decrypt a_3^m. If a_3^m is negative, then the first ciphertext a_1^m is the ciphertext of the boolean expression $E([x_0 \leq x_1])$, else the second ciphertext a_2^m is the value of $E([x_0 \leq x_1])$. A ciphertext of the smaller value can be computed by one multiplication protocol: $E(x_1 + [x_0 \leq x_1](x_0 - x_1))$.

The protocol is correct since if $a_3 < 0$, then the boolean expression $[x_b \leq x_{1-b}] = 1 \Leftrightarrow [x_0 \leq x_1] = 1 - b$, which is encrypted by a_1. If $a_3 > 0$, then $[x_b \leq x_{1-b}] = 0 \Leftrightarrow [x_0 \leq x_1] = b$, which is encrypted by a_2. Due to re-randomization, no one can infer which side of the coin showed up at any party X_i ($i = 2, \ldots, m$). Rerandomization does not change the sign of a_3, unless the ciphertexts are switched as well.

2.2. Comparison Protocol based on Secret Shares

Secret sharing refers to a method for distributing a secret amongst a group of parties, each of which is allocated a share of the secret. The secret can be reconstructed only when the shares are combined together (individual shares are of no use on their own). In Shamir’s secret sharing scheme [9], the sharing of a secret x is achieved as follows: Each party X_i holds a value $x_i = f_x(i)$ where f is a random $t-$degree polynomial subject to the condition that $f_x(0) = x$. It is very simple to see that a coalition of t players has no information about the value x. In our setting (semi-honest model and no faults), we require $t \geq m/2$.

It is easy to extend Shamir secret sharing to let the parties compute any linear combination of secrets without gaining information on intermediate results of the computation. To add (subtract) two shared secrets together, the players need only add (subtract) together individual shares at each evaluation point. Computing the product of two secrets is not so trivial, but it is still possible to reduce it to a linear computation [10, 11, 13]. Thus, it is possible to compute any "arithmetic" function (i.e., function involving only addition, subtraction, and multiplication) of secrets securely and robustly.

2.2.1 Protocol

Let x_0 and x_1 be the numbers to compare. Each party X_i then holds two shares $x_{0_i} = f_{x_0}(i)$ and $x_{1_i} = f_{x_1}(i)$ for secrets x_0, x_1, respectively. The output of the protocol is the bit $[x_0 \leq x_1] \in \{0,1\}$, shared among the m parties. Thus, the output is secret.

Given this, the protocol proceeds as follows:

1. Each party X_i ($i = 1, \ldots, m$) randomly chooses very large random numbers r_i and r'_i (that can multiplicatively hide $x_0 - x_1$). He then chooses a random bit $b_i \in \{0,1\}$, computes $s_i = (-1)^{b_i} r_i$ and $s'_i = (-1)^{1-b_i} r'_i$, and generates the Shamir secret shares of s_i and s'_i with threshold t.

2. Each party X_i ($i = 1, \ldots, m$) sends the generated pair of shares (s_{i_j}, s'_{i_j}) to the respective parties X_j. At the end of this step, each party X_j ($j = 1, \ldots, m$) has the set:
$$S_j = \{s_{1_j}, \cdots, s_{m_j}\}$$
corresponding to the shares of the random values s chosen by all the m parties. With respect to shares of the random value s', each party X_j ($j = 1, \ldots, m$) locally computes
$$p'_j = \sum_{i=1}^{m} s'_{i_j}$$

3. For simplicity, we assume that m is even (the extension for an odd m is trivial). All the parties X_i ($i = 1, \ldots, m$) engage in collaborative pairwise multiplications to compute the product values:
$$(s_1 \times s_2), (s_3 \times s_4), \cdots, (s_{m-1} \times s_m)$$
leading to $m/2$ product values shared among m players. A collaborative multiplication [10, 11, 13] of $s_1 \times s_2$ takes the shares of s_1 and s_2 as input, and results in each party X_i receiving its respective share of the computed product: $(s_1 \times s_2)_i$.

4. Repeat the above step to pairwise multiply the $m/2$ values to obtain $m/4$ product values, then pairwise multiply the $m/4$ values to obtain $m/8$ product values, and so on. At the end of this step, we have computed the product $p = s_1 \times s_2 \times \cdots \times s_m$ shared among the m players.

5. All parties X_i ($i = 1, \ldots, m$) locally subtract $d_i = a_i - b_i$, and collaboratively compute the product $c = p \times d$.

6. All parties X_i ($i = 1, \ldots, m$) locally subtract the sum p'_i of their s' values from c_i, i.e. $c_i = c_i - p'_i$.

7. All parties collaboratively reconstruct c. Let $e = 0$ if c is positive, else $e = 1$. The value of the boolean expression $[x_0 \leq x_1]$ is then $[x_0 \leq x_1] = e \oplus b_1 \oplus \cdots \oplus b_m$, where $\oplus$ refers to the XOR logical operation [14].

The correctness of the protocol can be argued as follows: Let $[x_0 \leq x_1] = 0$, i.e. $x_0 > x_1$. Given this, if $e = 0$ (i.e., c is positive), then it implies that the number of coins flipped $b_i = 1$ is even. As a result, their XOR value is 0, which further XORed with $e = 0$ and coins flipped $b_j = 0$ is still $0 = [x_0 \leq x_1]$. The correctness arguments for the remaining scenarios follow analogously.

2.3. Analysis

We compare the two protocols based on the following complexity criterions:

- Computation complexity: is measured with respect to the number of basic operations that need to be executed for a run of the protocol.

- Round complexity: This complexity parameter quantifies the need for synchronization among the parties. It refers to the number of times during the protocol, a party on reaching a certain stage, needs to wait for the other parties to reach that stage before proceeding.

Let m be the number of parties. The protocol based on homomorphic encryption (Section 2.1.1) executes in a ring based fashion with only one party active at a time while computing c. Thus, the protocol needs m rounds for all the parties to compute c, plus one round to decrypt c, giving the round complexity $O(m)$. The secret shares based protocol (Section 2.2.1) on the other hand has round complexity

$O(\log_2 m)$. The log rounds complexity arises from the collaborative multiplications needed to compute the product of the randomly chosen values s.

While the second protocol appears favorable from a round complexity perspective, it is difficult to compare the computational complexities of the two protocols. The main problem is with respect to identifying common basic operations used in both the protocols. While the basic operations used by both the protocols are same: addition, subtraction, multiplication, encryption and decryption (sharing and reconstruction in the secret shares setting), their complexities are incomparable. For instance, while multiplication can be performed locally in a homomorphic setting, it needs to be performed in a collaborative fashion in the secret shares setting. Similarly, while addition and subtraction can be performed locally in both the settings, their underlying logics are still very different. The same applies for encryption and decryption operations as well. To accommodate this inherent difference, we give some experimental results in the sequel.

2.3.1 Experimental Analysis

The implementation was done in Java and evaluated on a computer with a 2 GHz Intel Core 2 Duo processor and 2 GB of RAM running Windows Vista using version 1.6 of Sun's Java SDK. The communication infrastructure is based on Java RMI [16].

We used a 1024-bit RSA modulus for the homomorphic encryption. The choice of a minimum key-length of 1024-bits seems to be acceptable[1].

For the secret shares field $\mathbb{Z}_q$, we used q of bit length 256.

The run time comparison of the two protocols is summarized by the graph in Fig. 1. It is easy to see that the secret shares based protocol outperforms the homomorphic one, and would continue to do so even as the number of parties increases.

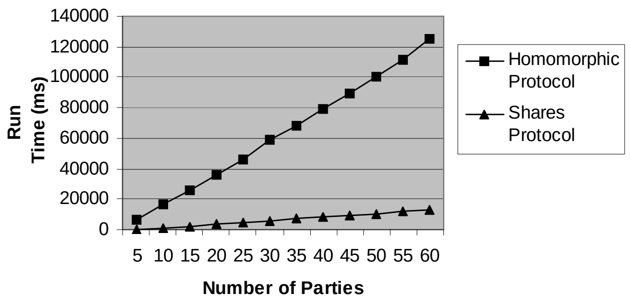

Figure 1. Run time comparison of the two protocols vs. the number of parties

3. Conclusion

Secure comparison protocols are a basic building block of SMC, which implies that an efficient secure comparison protocol would improve the efficiency of any SMC protocol using it. In this work, we proposed two efficient protocols for secure comparison, one for the homomorphic encryption setting, and the other for the secret shares setting. We also provided experimental results to substantiate our claims about their efficiency. Based on our experiments, the secret shares based protocol seems to be faster than the other one.

In our experiments, we did not study the impact of network/communication issues on the protocols. Observe that the curve corresponding to the secret shares based protocol increases linearly (and not in a logarithmic fashion) in Fig. 1. This is probably because the tests were performed on a single machine with the different parties simulated locally. In future, we would like to perform the tests in a real distributed setting, to study the impact of network/communication issues on the protocols.

Acknowledgement

We would like to thank Octavian Catrina for providing the JSMC framework used in the implementations.

References

[1] A. Yao, *Protocols for Secure Computation*, in proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS), pp. 160164, 1982.

[2] O. Goldreich, S. Micali, and A. Wigderson, *How to Play Any Mental Game*, in proceedings of the 19th Annual Symposium on Theory of Computing (STOC), pp. 218229, 1987.

[3] T. Nishide and K. Ohta, *Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol*, in proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography (PKC), pp. 343-360, 2007.

[4] W. Du and M. J. Atallah, *Secure Multi-party Computation Problems and their Applications: A Review and Open Problems*, in proceedings of the New Security Paradigms Workshop (NSPW), pp. 12-21, 2001.

[1] In 2009, a machine costing about 250 million dollars can factor a 1024-bit RSA key in a day [15], so a 10 million dollars machine would take just under a month.

[5] P. Bogetoft, D. L. Christensen, I. Dmgard, M. Geisler, T. Jakobsen, M. Krigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, *Multi-party Computation Goes Live*, Cryptology ePrint Archive, Report 2008/068, 2008.

[6] E. Kiltz, G. Leander, and J. Malone-Lee, *Secure Computation of the Mean and Related Statistics*, in proceedings of the 2nd Theory of Cryptography Conference (TCC), Spring LNCS vol. 3378, pp. 283-302, 2005.

[7] I. Damgard and M. Jurik, *A Generalization, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System*, in proceedings of the 4th International Workshop of Practice and Theory in Public Key Cryptography (PKC), Springer LNCS vol. 1992, pp. 119-136, 2001.

[8] P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, in proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT), Spring LNCS 1592, pp. 223-238, 1999.

[9] A. Shamir, *How to Share a Secret*, Communications of the ACM, 22(11): 612-613, 1979.

[10] R. Gennaro, M O. Rabin, and T. Rabin, *Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography*, in proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC), pp. 101-111, 1998.

[11] V. Nikov, S. Nikova, and B. Preneel, *On Multiplicative Linear Secret Sharing Schemes*, in proceedings of the 4th International Conference on Cryptology in India (INDOCRYPT), pp. 135-147, 2003.

[12] F. Bahr, M. Boehm, J. Franke, and T. Kleinjung, *RSA200*, Available at http://www.crypto-world.com/announcements/rsa200.txt, 2005.

[13] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computations*, in: proceedings of the 20th Annual Symposium on the Theory of Computing (STOC), ACM Press, pp. 110, 1988.

[14] *Exclusive Disjunction*, Available at http://en.wikipedia.org/wiki/Exclusive_disjunction.

[15] A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, 14(4): 255293, 2001.

[16] *Java Remote Method Invocation Home*, Available at http://java.sun.com/javase/technologies/core/basic/rmi/.