Efficient Data Intensive Secure Computation: Fictional or Real?

Changyu Dong(✉)

Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
changyu.dong@strath.ac.uk

Abstract. Secure computation has the potential to completely reshape the cybersecruity landscape, but this will happen only if we can make it practical. Despite significant improvements recently, secure computation is still orders of magnitude slower than computation in the clear. Even with the latest technology, running the killer apps, which are often data intensive, in secure computation is still a mission impossible. In this paper, I present two approaches that could lead to practical data intensive secure computation. The first approach is by designing data structures. Traditionally, data structures have been widely used in computer science to improve performance of computation. However, in secure computation they have been largely overlooked in the past. I will show that data structures could be effective performance boosters in secure computation. Another approach is by using fully homomorphic encryption (FHE). A common belief is that FHE is too inefficient to have any practical applications for the time being. Contrary to this common belief, I will show that in some cases FHE can actually lead to very efficient secure computation protocols. This is due to the high degree of internal parallelism in recent FHE schemes. The two approaches are explained with Private Set Intersection (PSI) as an example. I will also show the performance figures measured from prototype implementations.

1 Introduction

In the past a few years, we have seen a dramatic increase in the scale and financial damage caused by cyber attacks. Data security is now of paramount importance for most organizations. Compounding the problem, changes in computing – particularly the booming of Cloud computing and collaborative data analysis – has added another layer of complexity to the security landscape. Traditionally, an organization can lock their data in secure storage and process it within an in-house facility operated by trusted staff. But increasingly, data processing is moving out of the trusted zone and security mechanisms that used to be effective do not work any more. A promising solution to solve this problem is *secure computation*. Secure computation allows for computation of arbitrary functions directly on encrypted data and hides all information about the data against untrusted parties, even if the untrusted parties are involved in the computation. It is a transformative technology that will completely change the game.

DOI: 10.1007/978-3-319-26096-9_35

One prediction says that within 15 years, the secure computation sector will be bigger than the anti-malware sector which currently has the largest share of the IT security industry [1].

Secure computation research started in the 1980s. Yao first defined the concept of secure computation in his seminal paper [2]. The goal of secure computation is to allow multiple parties to jointly compute a function over their inputs, and keeping these inputs private. There are several different approaches for achieving this goal. One prominent secure computation technique is Yao's garbled circuits protocol [3]. In this protocol, a function converted into an encrypted Boolean circuit and the parties evaluate the circuit with encrypted inputs. Another Boolean circuit based technique is the GMW protocol by Goldreich et al. [4]. Also Cramer et al. showed that secure computation can be done with arithmetic circuits and secret shared inputs [5]. Gordon et al. proposed a technique for secure computation in a von Neumann-style Random Access Machine (RAM) model by using an Oblivious RAM [6]. Recently, the development of Fully Homomorphic Encryption (FHE) provided a new direction in secure computation [7]. Apart from those generic secure computation techniques, there are also many special-purpose protocols that are designed for specific secure computation problems, e.g. private set intersection [8] and oblivious polynomial evaluation [9]. Secure computation is an obvious solution for a class of problems in which parties must provide input to a computation, but no party trusts any other party with that data. Examples include e-voting, auctions, information retrieval, data sharing, data mining and many more. Despite the fact that it has so many potential applications, secure computation has remained purely theoretical for many years. Efficiency is one of the main reasons.

Recently there have been a few efforts aiming to turn secure computation from a theorists' toy to a real world tool. Significant progress has been made in the last five years to improve the efficiency of secure computation by algorithmic advancements. For example, various protocols designed to efficiently compute a specific function securely; improvements on garbled circuits including free XOR [10], efficient OT extension [11] and fast cut-and-choose [12]; more efficient share-based multiparty secure computation protocols including Sharemind [13] and SPDZ [14]; more efficient RAM program based secure computation [15]; optimizations for FHE including SIMD operations [16] and polylog FHE [16]. The improvement is significant. Taking garbled circuit based secure computation as an example, after integrating many optimizations to date, the FastGC framework [17] is 10^4 times faster than FairPlay [18] which was implemented in 2004.

That said, secure computation is still far from being practical. Despite all the improvements, secure computation is still tens of thousand to billions times slower than computation in the clear. The overhead might be acceptable if the data to be processed were small, but can be prohibitive when the data is big. Imagine we have a secure computation mechanism which slows down the computation by 10,000 times, then what we can do in the clear in 10 seconds now needs more than 1 day to complete, and what we can do in the clear in 10 hours now needs more than 10 years! Paradoxically, when talking about the killer apps of secure computation, people often use examples such as companies having so much data that they do not have resources to process and have to process it in untrusted clouds, or two mutually untrusted parties have to mine their massive datasets together. Although the examples show the necessity of secure computation, current secure computation technology is incapable of handling such data-intensive applications. This becomes a major impediment to widespread use of secure computation.

How to make data intensive secure computation practical? In the rest of this paper, I will show two new approaches that have great potential: by designing data structures and by using newly developed FHE techniques. I will present the ideas using Private Set Intersection (PSI) protocols as an example.

2 Private Set Intersection: Background

A PSI protocol is a two-party protocol in which a client and a server want to jointly compute the intersection of their private input sets in a manner that at the end the client learns the intersection and the server learns nothing. PSI protocols have many practical applications. For example, PSI has been proposed as a building block in applications such as privacy preserving data mining [19,20], human genome research [21], homeland security [22], Botnet detection [23], social networks [24], location sharing [25] and cheater detection in online games [26]. Many applications requires massive datasets as inputs. The first PSI protocol was proposed by Freedman et al. [8]. There are several approaches for PSI protocols. Some of them are based on oblivious polynomial evaluation [8,27,28], some are based on oblivious pseudorandom function (OPRF) evaluation [22,29–31], and some are based on generic garbled circuits [32].

3 Data Structural Approach

In computer science, traditionally an effective approach to improve the efficiency of data intensive computation is by using an appropriate data structure, but in secure computation, the power of data structures has been largely overlooked. The reason for that is probably because in the past secure computation research focused on showing feasibility and the use cases were limited to those with small data input. But when we are moving towards real world applications in which data plays the central role and drives the computation, data structural design will become an indispensable part of secure computation. A good example of this data structural approach is the garbled Bloom Filter and the PSI protocol based on this data structure [33].

3.1 From Bloom Filter to Garbled Bloom Filter

A Bloom filter [34] is a compact data structure for probabilistic set membership testing. It is an array of m bits that can represent a set S of at most n elements.

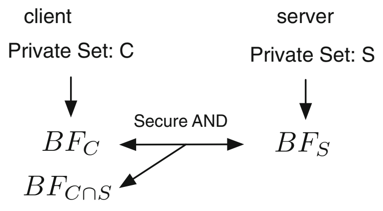

Fig. 1. The naive PSI protocol based on bloom filters.

A Bloom filter comes with a set of k independent uniform hash functions $H = \{h_0, ..., h_{k-1}\}$ that each h_i maps elements to index numbers over the range $[0, m-1]$ uniformly. Let us use BF_S to denote a Bloom filter that encodes the set S, and use $BF_S[i]$ to denote the bit at index i in BF_S. Initially, all bits in the array are set to 0. To insert an element $x \in S$ into the filter, the element is hashed using the k hash functions to get k index numbers. The bits at all these indexes in the bit array are set to 1, i.e. set $BF_S[h_i(x)] = 1$ for $0 \leq i \leq k-1$. To check if an item y is in S, y is hashed by the k hash functions, and all locations y hashes to are checked. If any of the bits at the locations is 0 , y is not in S, otherwise y is *probably* in S.

A standard Bloom filter trick is that if there are two Bloom filters, each encodes a set S_1 and S_2, and both are of the same size and built using the same set of hash functions, we can obtain another Bloom filter $BF_{S_1 \cap S_2}$ by bitwisely ANDing BF_{S_1} and BF_{S_2}. The resulting Bloom filter $BF_{S_1 \cap S_2}$ encodes the set intersection $S_1 \cap S_2$. It seems that we can obtain an efficient PSI protocol (Fig. 1) immediately from this trick. However, this naive protocol is not secure. The reason is that due to collisions, the resulting Bloom filter $BF_{C \cap S}$ usually contains more 1 bits than the Bloom filter built from scratch using $C \cap S$. This means $BF_{C \cap S}$ leaks information about elements in S.

To avoid information leakage, we designed the garbled Bloom filters (GBF). A garbled Bloom filter is much like a Bloom filter: it is an array of size m with k hash functions. The difference is that at each position in the array, it holds a λ-bit string rather than a bit, where λ is the secure parameter. The bit string is either a share of a set element or a random string. To encode a set S, each element $s \in S$ is inserted as follows: initially all positions in the GBF is set to NULL. We then hash the element using the k hash functions. For $0 \leq j \leq k-2$, If $GBF[h_j(s)] = NULL$ then we put an λ-bit random string at this position, and then we set $GBF[h_{k-1}(s)] = s \oplus (\bigoplus_{j=0}^{k-2} GBF[h_j(s)])$. We can see that each of the k position $GBF[h_j(s)]$ holds a share of s. The shares has the property that if all k shares are present, we can reconstruct the element from the shares $s = \bigoplus_{j=0}^{k-1} GBF[h_j(s)])$; however any subset that has less than k shares reveals no information about the element. After inserting all elements in s to the GBF, we put a λ-bit random string at each position that is still NULL. To query an element y, y is hashed by the k hash functions and we test $\bigoplus_{j=0}^{k-1} GBF[h_j(y)] \overset{?}{=} y$. If the test is true, then y is in the set S.

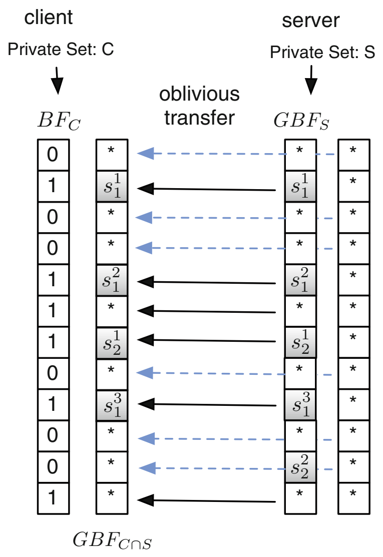

Fig. 2. The oblivious bloom intersection protocol.

A secure PSI protocol can then be built using a Bloom filter and a garbled Bloom filter (Fig. 2). In the protocol, the client encodes its set into a Bloom filter BF_C, the server encodes its set into a garbled Bloom filter GBF_S. The server also generates an array contains m random bit strings of length λ. For each position $0 \le i \le m-1$, the client and server run a (2,1)-Oblivious Transfer protocol [35] such that if the bit $BF[i] = 1$, the client receives $GBF[i]$, if the bit $BF[i] = 0$, the client receives the ith string from the random string array. At the end of the protocol, the result is a garbled Bloom filter $GBF_{C\cap S}$ that encodes the intersection.

By using a garbled Bloom filter, we fix the information leakage problem. In the intersection garbled Bloom filter $GBF_{C\cap S}$, there might still exist residue shares that belong to elements not in the intersection. However, if an element s is not in $C \cap S$, then the probability of all its shares remain in $GBF_{C\cap S}$ is

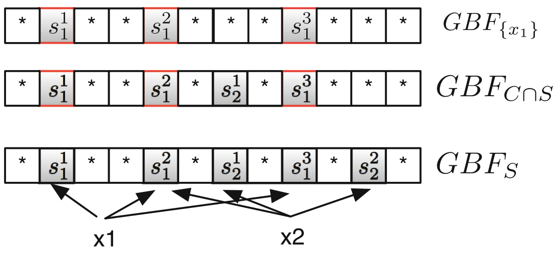

Fig. 3. Indistinguishability of the intersection garbled bloom filter.

negligible. Then by the property of the shares, the residue shares of s in $GBF_{C\cap S}$ leak no information about s. For example, in Fig. 3, s_2^1 in $GBF_{C\cap S}$ is a share of x_2 which is not in the intersection. The element x_2 has 3 shares and one of the share s_2^2 is not transferred to the client in the protocol. Then the other two shares remain in $GBF_{C\cap S}$ look uniformly random and do not leak information about x_2.

3.2 Performance Comparison

The PSI protocol obtained from garbled Bloom filter has many advantages: it has linear complexity, is easy to parallelize, relies mainly on symmetric key operations and it is much efficient than previous best protocols. We compared the performance with the previous best protocols. One protocol is by Huang et al. based the garbled circuits approach [32], and another is by De Cristofaro et al. based on ORPF evaluation [22]. Figure 4 shows the performance improvement at 128-bit security. The numbers displayed in the figure are ratios of running time (previous protocol to our protocol).

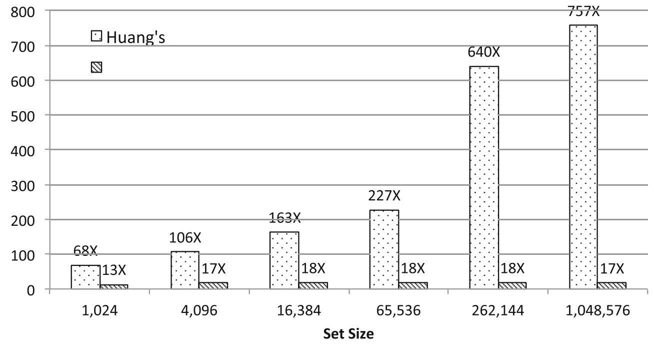

Fig. 4. Performance comparison.

4 Fully Homomorphic Encryption Approach

FHE is a newly established area in cryptography. An FHE scheme allows (any) computation to be carried out on encrypted data directly. FHE is a powerful tool and at the same time is notorious for its inefficiency. It is a common belief that FHE is too inefficient to be practical yet. However, this common belief is not always true. In this section I will show how to build a more efficient PSI protocol using fully homomorphic encryption.

4.1 The BGV FHE Scheme

In 2009, Gentry [7] developed the first FHE scheme. Following the breakthrough, several FHE schemes based on different hardness assumptions have been proposed, e.g. [36, 37].

The RLWE variant of BGV [37] is among the most efficient FHE schemes; it operates in certain polynomial rings. Namely, let $\Phi_m(x)$ be the m-th cyclotomic polynomial with degree $\phi(m)$, then we have a polynomial ring $\mathbb{A} = \mathbb{Z}[x]/\Phi_m(x)$, i.e. the set of integer polynomials of degree up to $\phi(m) - 1$. Here $\phi(\cdot)$ is the Euler's totient function. The ciphertext space of the BGV encryption scheme consists of polynomials over $\mathbb{A}_q = \mathbb{A}/q\mathbb{A}$, i.e. elements in $\mathbb{A}$ reduced modulo q where q is an odd integer[1]. The plaintext space is usually the ring $\mathbb{A}_p = \mathbb{A}/p\mathbb{A}$, i.e. polynomials of degree up to $\phi(m) - 1$ with coefficients in $\mathbb{Z}_p$ for some prime number $p < q$.

There are three basic algorithms in the BGV scheme:

- $G(p, \lambda, L)$: The key generation algorithm. Given p, λ and L such that p is the prime number that defines the plaintext space, λ is the security parameter and L is the depth of the arithmetic circuit to be evaluated, generate a secret key, the corresponding public key and a set of public parameters.
- $E_{pk}(\bar{m})$: The encryption algorithm. Given a public key pk, encrypt an element $\bar{m} \in \mathcal{A}_p$.
- $D_{sk}(c)$: The decryption algorithm. Given the secret key sk, decrypt a ciphertext c.

Being a fully homomorphic encryption scheme, the BGV scheme supports both multiplication and addition operations over ciphertexts. Let us denote homomorphic addition by $\boxplus$ and homomorphic multiplication by $\boxtimes$. We can homomorphically add or multiply two ciphertexts together. We can also homomorphically add or multiply a ciphertext with a plaintext.

4.2 Polynomial Representation of a Set

Freedman et al. [8] first proposed to use a polynomial for representing a set in PSI. Given a set S, we can map each element in S to an element in a sufficiently large field R. Then S can be represented as a polynomial (in a ring $R[x]$). The polynomial is defined as $\rho(x) = \prod_{s_i \in S}(x - s_i)$. The polynomial $\rho(x)$ has the property that every element $s_i \in S$ is a root of $\rho(x)$. For two polynomials ρ_1 and ρ_2 that represent the two sets S_1 and S_2 respectively, the the greatest common divisor of the two polynomials $gcd(\rho_1, \rho_2)$ represents the set intersection $S_1 \cap S_2$. Based on this, we can design protocols to securely obtain the set intersection. Without loss of generality, let both ρ_1 and ρ_2 to be of degree δ and let γ_1 and γ_2 to be two uniformly random degree δ polynomials in $R[x]$, Kissner and Song proved in [27] that $\gamma_1 \cdot \rho_1 + \gamma_2 \cdot \rho_2 = \mu \cdot gcd(\rho_1, \rho_2)$ such that μ is a uniformly random polynomial. This means if ρ_1 and ρ_2 are polynomials representing sets S_1 and S_2, then the polynomial $\gamma_1 \cdot \rho_1 + \gamma_2 \cdot \rho_2$ contains only information about $S_1 \cap S_2$ and no information about other elements in S_1 or S_2. This forms the basis of their PSI protocol in which a party obtains $\gamma_1 \cdot \rho_1 + \gamma_2 \cdot \rho_2$ to find the set intersection but learns nothing more about elements in the other party's set.

[1] In the BGV encryption scheme, there are actually a chain of moduli $q_0 < q_1 < \cdots < q_L$ defined for modulus switching. But for simplicity we just use q throughout the paper.

However, Kissner's protocol is not practical due to the facts that it uses expensive Paillier encryption and the computational complexity is quadratic in the size of the sets.

4.3 The Private Set Intersection Protocol Based on FHE

We parallelize computation by utilizing the native plaintext space of BGV to load multiple data items. The native plaintext space of BGV is a polynomial ring, therefore a set can be easily represented in the plaintext space. To simplify the description, I will start from the case where $|C| = |S| = \frac{\phi(m)}{2} - 1$. In the protocol, the client has a BGV key pair (pk, sk) and a set C. The server has a set S. The two parties encode their sets into ρ_C and ρ_S that are polynomials in $\mathbb{A}_p$. The protocol is shown in Fig. 5:

1. The client encrypts its set polynomial ρ_C and sends the ciphertext c to the server.
2. The server chooses random polynomial γ_C and γ_S in $\mathbb{A}_p$, each of degree $\frac{\phi(m)}{2} - 1$, then the server computes homomorphically $c' = (c \boxtimes \gamma_C) \boxplus (\rho_S \cdot \gamma_S)$. The server sends c' to the client, who then decrypts the ciphertext and obtains the polynomial $\rho_C \cdot \gamma_C + \rho_S \cdot \gamma_S$.
3. The client then evaluates the polynomial obtained in the last step with elements in C. For each element, if it is a root then it is in the intersection. The client then outputs the intersection $C \cap S$.

To compute the intersection of sets whose sizes are larger than $\frac{\phi(m)}{2} - 1$, we can use bucketization. Bucketization is a process to partition a large set into disjoint subsets (buckets). The two parties use a public uniform hash function $H : \{1, 0\}^* \rightarrow [1, k]$ to map their set elements into k buckets. This is done by hashing each element to get a bucket number and putting the element into the bucket with this number. If the size of the set to be bucketized is n, then each bucket will have around n/k elements. The two parties can choose k so that with a high probability, each bucket has no more than $\frac{\phi(m)}{2} - 1$ elements. To prevent information leakage through bucket size, the two parties pad each bucket with random elements so that all buckets have the same size $\frac{\phi(m)}{2} - 1$. They then run the PSI protocol k times. In the ith run, each party uses its ith bucket as the input to the PSI protocol. The union of outputs is the intersection of the two sets.

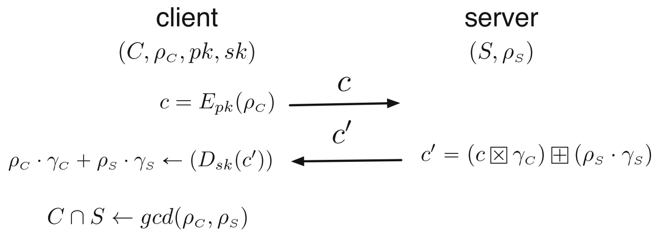
client
(C, ρ_C, pk, sk)
server
(S, ρ_S)
$c = E_{pk}(\rho_C)$
c
c'
$\rho_C \cdot \gamma_C + \rho_S \cdot \gamma_S \leftarrow (D_{sk}(c'))$
$c' = (c \boxtimes \gamma_C) \boxplus (\rho_S \cdot \gamma_S)$
$C \cap S \leftarrow gcd(\rho_C, \rho_S)$

Fig. 5. The PSI protocol.

4.4 Efficiency

The protocol is very efficient. This is due to the high degree of parallelism provided by the BGV scheme. In the protocol, we process a set of $\frac{\phi(m)}{2} - 1$ elements in one go, rather than processing them individually. Therefore the total computational cost is amortized by $\frac{\phi(m)}{2} - 1$. The parameter $\phi(m)$ is large, therefore the amortized cost is small.

Table 1. Performance of PSI protocols.

	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
GBF-PSI	0.67	1.99	8.21	32.41	130.42	530.36
FHE-PSI	0.11	0.14	0.45	1.55	5.91	23.48
Improvement	6X	14X	18X	21X	22X	23X

* Running Time in seconds

Table 1 shows the performance comparison of the GBF based and the FHE based PSI protocols. In the experiment, security parameter is set to 128-bit. The parameters for the BGV keys were $|p| = 32, L = 1, |q| = 124, \phi(m) = 5002$. The set size varied from 2^{10} (1024) to 2^{20} (1,048,576). As we can see, the FHE based PSI protocol is much faster. For two 1 million elements input sets, the running time is less than half a minute, which is only 1 - 2 orders of magnitude slower than the computation in the clear.

5 Conclusion

In this paper, I presented two approaches that could lead to practical data intensive secure computation. One approach is by designing better data structures. The rationale behind this approach is that when the data to be processed is big, arranging it into certain data structures may make it more amendable for computation. Another approach is by using fully homomorphic encryption. Recent fully homomorphic encryption schemes provide us facilities to parallelize computation, which can greatly reduce the overall cost if the computation task is data parallel. The two approaches can be combined. For example, when using bucketization in the PSI protocol, the list of buckets is essentially a hash table data structure. The research along these two lines is still in an early stage, but further investigation will lead to fruitful results.

References

1. Evans, D.: Secure computation in 2029: Boom, bust, or bonanza. Applied Multi-Party Computation Workshop (2014)
2. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3–5 November 1982, pp. 160–164 (1982)

3. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27–29 October 1986, pp. 162–167 (1986)
4. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, USA, pp. 218–229 (1987)
5. Cramer, R., Damgård, I.B., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)
6. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: ACM Conference on Computer and Communications Security (2012)
7. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, 31 May–2 June 2009, pp. 169–178 (2009)
8. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
9. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, 1–4 May 1999, Atlanta, Georgia, USA, pp. 245–254 (1999)
10. 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)
11. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM Conference on Computer and Communications Security (2013)
12. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013)
13. Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multiparty computation for data mining applications. Int. J. Inf. Sec. **11**(6), 403–418 (2012)
14. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
15. Liu, C., Huang, Y., Shi, E., Katz, J., Hicks, M.W.: Automating efficient ram-model secure computation. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, 18–21 May 2014, pp. 623–638 (2014)
16. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 465–482. Springer, Heidelberg (2012)
17. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: 20th USENIX Security Symposium, San Francisco, CA, USA, 8–12 August 2011, Proceedings (2011)
18. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: Proceedings of the 13th USENIX Security Symposium, 9–13 August 2004, San Diego, CA, USA, pp. 287–302 (2004)
19. Aggarwal, C.C., Yu, P.S.: Privacy-Preserving Data Mining - Models and Algorithms. Advances in Database Systems, vol. 34. Springer, USA (2008)

20. Dong, C., Chen, L.: A fast secure dot product protocol with application to privacy preserving association rule mining. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014, Part I. LNCS, vol. 8443, pp. 606–617. Springer, Heidelberg (2014)
21. Baldi, P., Baronio, R., Cristofaro, E.D., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: ACM Conference on Computer and Communications Security, pp. 691–702 (2011)
22. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010)
23. Nagaraja, S., Mittal, P., Hong, C.Y., Caesar, M., Borisov, N.: Botgrep: finding p2p bots with structured graph analysis. In: USENIX Security Symposium, pp. 95–110 (2010)
24. Mezzour, G., Perrig, A., Gligor, V., Papadimitratos, P.: Privacy-preserving relationship path discovery in social networks. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 189–208. Springer, Heidelberg (2009)
25. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS (2011)
26. Bursztein, E., Hamburg, M., Lagarenne, J., Boneh, D.: Openconflict: preventing real time map hacks in online games. In: IEEE Symposium on Security and Privacy, pp. 506–520 (2011)
27. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
28. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 312–331. Springer, Heidelberg (2010)
29. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
30. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009)
31. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
32. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
33. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: ACM Conference on Computer and Communications Security (2013)
34. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
35. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)
36. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
37. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: ITCS, pp. 309–325 (2012)