A Secure and Efficient Sample Filtering Protocol for Massive Data

Linghui Chen,Xiqiang Dai,Hong Wang,Yuanyuan Cen,Xiaochuan Peng,Hongyu Li,Xiaolin Li✉
AI Institute, Tongdun Technology
{linghui.chen, xiqiang.dai, hong.wang, yuanyuan.cen, xiaochuan.peng, hongyu.li,
xiaolin.li}@tongdun.net

Abstract

Private Set Intersection (PSI) is widely used for secure sample alignment in federated learning. However, as the size of private set increases, the communication and computation cost increase almost linearly, which makes it hardly scale to large sets containing over hundreds of millions of elements. This paper proposes a server-aided sample filtering protocol, as a preprocessing phase of PSI to efficiently filter out most of the elements not in the intersection set. By applying random permutation and substitution, the size and the content of original set cannot be recovered by the 3rd party. Experimental results show that applying filtering protocol significantly reduces the total running time and communication cost for sample alignment, and is particularly efficient when the ratio of the intersection to the data set is very small.

1 Introduction

Federated learning framework was firstly proposed by Google in 2016. It aims to build machine learning models based on data distributed across multiple devices[14, 21] while preventing data leakage. After it was proposed, federated learning framework has been further studied to overcome statistical challenges [10] and improve security [3, 9, 15]. According to whether the data is distributed among various parties in the sample space or feature space, federated learning can mainly be divided into cross-sample federated learning and cross-feature federated learning. In this paper, we consider cross-feature federated learning. It applies to the cases where two data sets share the same sample ID space but differ in feature space. For example, consider an insurance company and a hospital in the same city. Their user sets are likely to contain most of the residents of the area, so the intersection of their user space is large. Since the insurance company records users' insurance information, while the hospital records users' physical health data, they hold different feature data of same users, which will have an important influence on the training model in the area of insurance fraud.

However, since the user sets of the two companies are not the same, the system needs to confirm the common users of the two parties before training. To solve the problem, many Private Set Intersection (PSI) protocols were proposed, where two parties P_A and P_B who hold sets X and Y respectively can identify the intersection $X \cap Y$ without revealing any information about elements that are not in the intersection. In [20, 4, 5], PSI protocols are proposed based on RSA encryption, Full-Domain Hash and Diffie-Hellman Exchange protocol. However, they lead to low efficiency and high communication overhead since all the IDs in both sets need to be encrypted several times with public-key encryption and the length of the transmitted data is several times the length of the ciphertext. In [6], several Boolean circuits for PSI were proposed and evaluated using Yao's garbled circuits. It shows that for larger security parameters, its performance is better than the blind RSA protocol of [4]. However, the computational complexity is proportional to the number of AND gates in the circuit. What's more, since the circuits required for calculation must be completely stored in the memory, the storage overhead is very large. In [18], a Bloom Filter-based protocol PSI based on OT extension is given,

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

which can achieve security against semi-honest and malicious adversaries. This protocol is further optimized for semi-honest adversaries in [17]. In [19], the OT-based PSI protocol in [17] has been extended to provide security against weakly malicious adversaries. In [16], the OT-based PSI protocol is proposed against semi-honest adversaries and malicious adversaries. An improved version of [16] was given in [13], which uses the OT extended protocol of [12] to propose an effective construction of an oblivious pseudorandom function (OPRF). Although the running time of these protocols is superior to that of foregoing protocols, the communication overhead is linear in the security parameter and amortizes with increasing set sizes.

PSI protocol performs well when the data sets is small, but when a party with hundreds of millions of elements, the communication cost and running time is unacceptable. In this paper, we propose a server-aided sample filtering protocol. At the stage of sample filtering, most of the non-intersecting elements are filtered out, so that parties can get relatively small data sets, and then accurately aligned sets are obtained in the subsequent PSI stage. The protocol is efficiently implemented with a semi-trusted third party. It is feasible since the third party can be undertaken by a trusted hardware or multiple untrusted hardware [8].The main contributions of this paper are summarized as follows:

1. We propose a server-aided sample filtering protocol, which can filter out most of the non-intersecting elements securely and efficiently. The proposed protocol is supposed to be called before a PSI protocol when at least one of the private sets is very large, for example, over 10 million elements, to dramatically reduce the input size of the PSI protocol.
2. The communication cost is very low even with the huge size of the input set. The values and total number of the elements are not exposed to the 3rd party by applying random permutation and substitution.
3. We conduct plenty of experiments on several PSI schemes, and the results show that the total running time and communication overhead decrease significantly with the aid of our proposed protocol.

2 Preliminaries

2.1 Bloom Filter

A Bloom Filter (BF) is a probabilistic data structure invented in 1970 by Bloom [2]. BFs can efficiently determine an element's membership in a set, with a computational complexity of $\mathcal{O}(1)$ [1].

Internally, a BF uses a zero-initialized bit array $BF[\,]$ of size l. The set of elements is denoted by S with element capacity $|S| = n$. To insert an element, it is hashed with k different hash functions. The hashed results are used to index positions where the bits are set to 1 subsequently in $BF[\,]$. In the testing phase, only all bits at k identified positions are set to 1, the element is – most likely – contained in the BF.

2.2 Format-Preserving Encryption

Format-Preserving Encryption (FPE) is a cryptographic algorithm that encrypts certain types of plaintext with the same format. Therefore, it is suitable for encryption in practical scenarios such as the privacy protection of a social security number consisting of 13 decimal digits or a credit card number consisting of 16 decimal digits .

AES-FF1 algorithm is an FPE for block ciphers in the NIST.SP.800-38G standard [7]. It is used in this paper as a random permutation generation algorithm in Bloom Filters. What's more, a special case with radix of 2 is used to speed up the generation of permutations.

3 Sample Filtering

In this section, we describe the details of the sample filtering protocol. Sample filtering is designed for reducing the size of private set to accelerate the PSI protocol with low computation and communication cost. When the protocol is finished, each party gets a much smaller set (usually different for each party) than the original one, while the intersection set is involved. The idea is using bloom filter to represent the data owned by each party. Then the third party intersects the bloom filters and returns

the result to each party. Finally all parties get the candidate ID list by mapping '1's in bloom filter to IDs in their private set respectively. To make sure that no useful information is leaked to the third party, random permutation and substitution are applied to the bloom filters. The third party receives two encrypted bloom filters thus cannot get any extra information from them. Specifically, the random permutation is generated by a format preserving encryption algorithm called FF1 [7]. The substitution is performed by calling XNOR on the permuted bloom filter and a random bloom filter which is generated by hmac_drbg algorithm , a cryptographically secure pseudorandom number generator (CSPRNG).

For simplicity, The protocol involves three parties: one plays as a server for assisting the computation, noted as P_C; the other two parties are data providers, noted as P_A and P_B. It's worth noting that the protocol can also handle the filtering for more than two data providers.

The detailed steps are shown in Table 1.

Algorithm $\Pi_{SF}(\{P_A, P_B\}, P_C)$

Input: P_A, P_B holds ID sets S_A and S_B, respectively.
Output: P_A, P_B gets ID sets $S_A^{'}$ and $S_B^{'}$ respectively, where $(S_A \cap S_B) \subseteq S_A^{'} \subseteq S_A$, and $(S_A \cap S_B) \subseteq S_B^{'} \subseteq S_B$.

1. For $i \in \{A, B\}$, P_i generates $B_i = BF(S_i)$, where $BF(\cdot)$ represents a function that maps input to a bloom filter with length l. Denote the j-th bit of B_i as $B_{i,j}$. P_i saves the map table between j and $S_{i,j}$ as *Table*$_i$.
2. P_A and P_B negotiate two random numbers r_1, r_2 by calling Diffie-Hellman key exchange protocol twice.
3. *Permutation*:
   For $i \in \{A, B\}$, P_i computes $k = H_1(r_1)$, where H_1 is SHA256, a hash function. Then, for all $j \in [0, l-1]$, P_i generates an all-zero bloom filter B_i', then fills the $E_k(j)$-th bit of B_i' with the j-th element of B_i, where E is FF1 algorithm, and k is the secret key.
4. *Substitution*:
   For $i \in \{A, B\}$, P_i generates a random bit string of l length by adopting hmac_drbg algorithm initialized by seed r_2, then converts it to a bloom filter M_i. P_i computes $B_i^{''}$, where the j-th bit of $B_i^{''}$ is computed by $B_{i,j}' \odot M_{i,j}$. $\odot$ represents XNOR operation.
5. For $i \in \{A, B\}$, P_i sends $B_i^{''}$ to P_C.
6. P_C computes B_C, where the j-th bit of B_C is computed by $B_{A,j}^{''} \odot B_{B,j}^{''}$, and sends B_C to P_A and P_B.
7. *Inverse substitution*:
   For $i \in \{A, B\}$, P_i computes B_i^{**}, where the j-th bit of B_i^{**} is computed by $B_{C,j} \wedge B_{i,j}'$.
8. *Inverse permutation*:
   For $i \in \{A, B\}$, P_i generates an all-zero bloom filter B_i^{*}, then fills the j-th bit of B_i^{*} with the $E_k(j)$-th element of B_i^{**}.
9. For $i \in \{A, B\}$, P_i maps all the 1s in B_i^{*} to IDs using *Table*$_A$, and obtains the intersection set.

Table 1: Our Server-aided Sample Filtering Protocol

4 Analysis

4.1 Correctness

We first prove that the result of the protocol is correct without using encryption, then prove the encrypted version in section 3 has the same output as the encryption-free version. Note that in encryption-free version, party P_C performs bitwise AND operation on two bloom filters.

Proposition 1. *In encryption-free filtering protocol, each party involving in sample filtering protocol returns a set that contains the intersection of the input set.*

Proof. Party P_C calculates bitwise AND operation on the two received bloom filters from P_A and P_B and sends it back to both party. A '0' in the result filter means that, for at least one of the party, there is no input element mapping to that position of the filter. Thus the elements which are filtered out contain no elements of the intersection. □

Proposition 2. *The encrypted version of sample filtering outputs the same as the encryption-free version.*

Proof. In Table 1, step 8 performs an inverse permutation on the filter, which is exactly the reverse operation of permutation used in step 3. So in the next paragraph, we only need to prove the correctness of step 4 to 7, which is, for P_A and P_B, the output of step 7 is the result of bitwise AND operation on the two input of step 4 for P_A and P_B.

Without loss of generality, we assume the bloom filter is consisted of only 1 bit. Denote the input of step 7 for P_A and P_B as b_1 and b_2 respectively, r is the random bloom filter generated in step 4 for substitution. P_A computes $b_1 \odot r$, P_B computes $b_2 \odot r$, where $\odot$ represents XNOR. In step 6, P_C computes $(b_1 \odot r) \odot (b_2 \odot r)$ and sends it to P_A and P_B. In step 7, P_A computes $((b_1 \odot r) \odot (b_2 \odot r))\&b_1$, P_B computes $((b_1 \odot r) \odot (b_2 \odot r))\&b_2$. For the correctness of step 4 to 7, we need to verify:

$$((b_1 \odot r) \odot (b_2 \odot r))\&b_1 = b_1\&b_2 \tag{1}$$

and

$$((b_1 \odot r) \odot (b_2 \odot r))\&b_2 = b_1\&b_2 \tag{2}$$

We only prove the correctness of equation (1), because the proofs of equation (2) and (1) are similar. In equation (1), if b_1 is 0, then the left part and the right part all equal to 0. If b_1 is 1, when $b_2 = 0$, both the left side and right side equal to 0, else they both equal to 1. Thus the correctness of step 4 to 7 is verified. Since we have declared step 3 and step 8 are mutually inverse operation, then equation (1) holds. □

4.2 Security

We consider semi-honest adversaries, i.e., participants are assumed to follow specifications in the protocol but attempt to infer more information, during or after protocol execution. We announce the protocol is secure if used alone. When combined with PSI protocol, the whole security is a little different from a single PSI protocol. We analyze the information leakage of the whole protocol and show that it is very hard for an attacker to expose an element owned by the other party using the extra information in practice.

In section 3, the sample filtering protocol uses random permutation and substitution to encrypt the bloom filter, making the protocol safe to other parties. Random permutation is to mix up the source location of '1's in bloom filter. By using this, the location of the elements in the bloom filter are hidden. Substitution is to confuse the amount of '1's in bloom filter, by using XNOR bit operation on the permuted bloom filter and a random bloom filter which is generated by a cryptographically secure pseudorandom number generator, the size of the original set is hidden to the server P_C. The server P_C gets $B_A^{''}$ and $B_B^{''}$, which is permuted and substituted from B_A and B_B with two secret keys unrevealed to P_C, so the server cannot recover the original information from current results. Likewise, P_A and P_B can only get the intersected bloom filter, from which they can obtain a set that is a litter larger than the intersection set, and cannot know any information about the other party.

If sample filtering protocol is used to filter the datasets alone, which can achieve security against semi-honest adversaries. but when considering sample filtering and PSI as an integrated sample alignment protocol, the security of the new protocol is slightly weakened comparing with the PSI protocol. The security that a PSI protocol requires, no information other than the intersection should be leaked, is not obeyed by the new protocol. We will point out what extra information is leaked out, and analyze the difficulty for conducting a real attack.

Once a party (A) get the intersection set after running a PSI protocol, he may find some '1's in bloom filter of sample filtering do not related to an element of the intersection set. Then he can get the

knowledge that the party at opposite side has at least one element maps to that position of the bloom filter. Equally, the party knows a small portion of the hash value of a element owned by the other party (B), for example, the last 30 bits out of 256 bits hash value. This will only happen for the parties who can get the final intersection, the server or the 3rd party will not get any extra information.

A curious party A may try to guess what element party B is owned by the extra information. Here we explain it is still very hard to expose a certain element. Firstly, only a small potion of '1' in the bloom filter may expose extra information, this is because most of the elements not appeal in the intersection set have been filtered out in the result filter. Secondly, because of the irreversibility of the hash function, party A need to generate a huge number of elements and execute a rainbow table attack, which requires huge amounts of computing and storage resources. Thirdly, suppose the previous attack has been successfully performed, party A may found there exists a list of candidates correlated to a same bit '1' of the filter, he cannot confirm how many and which elements party B has.

5 Performance

We use 3 PCs to simulate 3 parties in the protocol, one for the server P_C, the other two for the parties that provide data, noted as P_A and P_B. Each PC has two Intel Xeon Gold 5118 CPU and 128GB RAM. The PCs are connected by LAN with 1000Mbps bandwidth. Five input sets are generated for P_A and P_B, their size varies from 2.2 million to 210 million, and the intersection size is from 0.17 million to 20.48 million, the details are list in table 2.

Dataset	P_A	P_B	Intersection
1	22	2.2	0.17
2	22	22	1.66
3	210	2.2	0.21
4	210	22	2.13
5	210	210	20.48

Table 2: The size of Datasets and Intersection (Unit: Million)

The hmac_drbg algorithm used in the protocol deploys hash function SHA256, we enter an entropy with bit length 512 to initialize it in our experiment. The secret key length for AES-FF1 algorithm is 256 bit. For computational efficiency, the filter length is restricted to power of 2, which makes modulus in bloom filter the same as binary truncation. As a result, a special case of AES-FF1 algorithm with radix 2 is deployed. For each set, we run 4 sets of experiments for 4 different filter lengths.

During experiments, four PSI protocols are considered to accompany with the filtering protocol, 1) naïve hash; 2) server-aided AES, which we would explain in the next paragraph; 3) blind RSA based [11]; 4) OT based [18]. For 1) and 4), we use implementation from PSI[1]. For 3), we adapt implementation from PyPSI[2]. The server-aided AES protocol is implemented locally.

Server-aided AES. Here we briefly introduce an efficient server-aided PSI protocol. The two parties first consult with each other to get a same secret key by a key exchange protocol. Then each party encrypts the IDs in his private set by a symmetric encryption algorithm using the secret key, and sends the IDs to the 3rd party. The 3rd party intersects the two encrypted ID sets and sends back the corresponding indexes to all the parties. Finally, both parties extract the intersection set from the indexes.

Table 4 shows the ratio of output set size after filtering to the final intersection size for party A and party B under different datasets. As expected, both ratios decrease when bloom filter size is increasing. When the length of bloom filter is above 2 times the maximum of two datasets sizes, both ratios are at an acceptable level, most of them are less than 5.

We compare the communication cost and running time cost of pure PSI protocols and PSI + sample filtering protocols.

[1]https://github.com/encryptogroup/PSI

[2]https://github.com/OpenMined/PyPSI

Communication cost. In the following, we evaluate the communication cost of protocols with and without the aid of the proposed sample filtering protocol, which is one of the bottlenecks of protocols. The results are shown in Table 3. We first introduce some necessary information of these protocols. The SHA256 hash function is used in the naïve Hash protocol to encrypt the dataset. AES-based protocol deploys ECB encryption mode after all the elements in the dataset are hashed by SHA256. The communication cost of the OT-based protocol is calculated by $512\epsilon n_2 + (k+s)n_1 l$ [18], where n_1, n_2 are the sizes of datasets of the two parties, ϵ , k, s are the Hash parameters, $l = \lambda + \log n_1 + \log n_2$, and λ is the statistical security parameters. Since the communication cost of sample filtering varies with the changing of bloom filter length, we use the average cost of multiple sample filtering protocols when calculating the total cost of sample filtering plus a PSI protocol. The individual cost of each phase is list in table 5 of Appendix A. When using sample filtering as a preprocessing phase before the PSI protocol, the total communication overhead are all decreased in all four circumstance. When the data set size get larger, the effect is more significant.

Protocols	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5
naïve Hash	0.73	1.41	6.34	7.04	13.74
Filtering+naïve Hash	0.14	0.62	0.33	1.12	5.19
AES	0.73	1.41	6.34	7.04	13.74
Filtering+AES	0.14	0.62	0.33	1.12	5.19
RSA	1.22	5.95	6.79	11.56	56.93
Filtering+RSA	0.44	1.96	0.71	3.12	15.20
OT	0.82	2.35	6.67	8.42	22.89
Filtering+OT	0.18	0.82	0.37	1.36	6.42

Table 3: Communication Cost for PSI protocols and Filtering+PSI protocols (GB)

Running time. Figure 1 shows the running time of different protocols. The shadow slot and the hollow slot indicate the running time of pure PSI protocol and the combination of the sample filtering and PSI protocol respectively. The empty slot indicates the protocol fails to get a result, which may either cause by out of memory or exceeding the time limit (48 hours). The results show that applying filtering at first stage significantly reduce the total running time, no matter which PSI protocol is used. Another improvement by filtering is that less memory is needed in the second stage, making the system be able to run on larger datasets. For example, OT-based PSI protocol fails to get a result on large datasets when running alone. With the help of sample filtering, it can handle larger dataset. It should be noted that the running time of the "Filtering+PSI" protocols does not include the time mapping from the bloom filter to the datasets, because it depends on the database used and the hardware performance. From our experience, the total running time is still much less than the pure PSI protocol even taking the mapping time into consideration.

In this section, we compared the communication cost and the running time of four different protocols with and without the aid of the sample filtering. Communication cost of 'Filtering+PSI' is significantly less than the pure PSI protocol, and the experimental results of running time further demonstrate the effectiveness of our sample filtering protocol.

6 Conclusion

This paper proposes a sample filtering protocol. The protocol aims to securely filter out most of the non-intersecting elements at low cost without exposing any valid information to the 3rd party. After that, the PSI protocol can then be deployed on much smaller sets. Compared with using PSI protocol alone, PSI protocols with the sample filtering show significant advantages in communication and computation cost, especially when the size of private sets is very huge.

References

[1] G. Bianchi, L. Bracciale, and P. Loreti. "better than nothing" privacy with bloom filters: To what extent? In J. Domingo-Ferrer and I. Tinnirello, editors, *Privacy in Statistical Databases - UNESCO Chair in Data Privacy, International Conference, PSD 2012, Palermo, Italy, September*

26-28, 2012. Proceedings, volume 7556 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2012.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017.

[4] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptol. ePrint Arch.*, 2009:491, 2009.

[5] E. D. Cristofaro and G. Tsudik. On the performance of certain private set intersection protocols. (and some remarks on the recent paper by huang et al. in ndss'12). *IACR Cryptol. ePrint Arch.*, 2012:54, 2012.

[6] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 789–800. ACM, 2013.

[7] M. Dworkin. Recommendation for block cipher modes of operation: Methods for format-preserving encryption. 2019.

[8] M. Fischlin, B. Pinkas, A. Sadeghi, T. Schneider, and I. Visconti. Secure set intersection with untrusted hardware tokens. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.

[9] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017.

[10] I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors. *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017.

[11] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings on Privacy Enhancing Technologies*, 2017(4):177–197, 2017.

[12] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2013.

[13] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016.

[14] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs/1610.02527, 2016.

[15] G. Liang and S. S. Chawathe. Privacy-preserving inter-database operations. In H. Chen, R. Moore, D. D. Zeng, and J. Leavitt, editors, *Intelligence and Security Informatics, Second Symposium on Intelligence and Security Informatics, ISI 2004, Tucson, AZ, USA, June 10-11, 2004, Proceedings*, volume 3073 of *Lecture Notes in Computer Science*, pages 66–82. Springer, 2004.

[16] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In J. Jung and T. Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

[17] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In K. Fu and J. Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 797–812. USENIX Association, 2014.

[18] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[19] P. Rindal and M. Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 297–314. USENIX Association, 2016.

[20] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 653–664. ACM, 2007.

[21] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *CoRR*, abs/1902.04885, 2019.

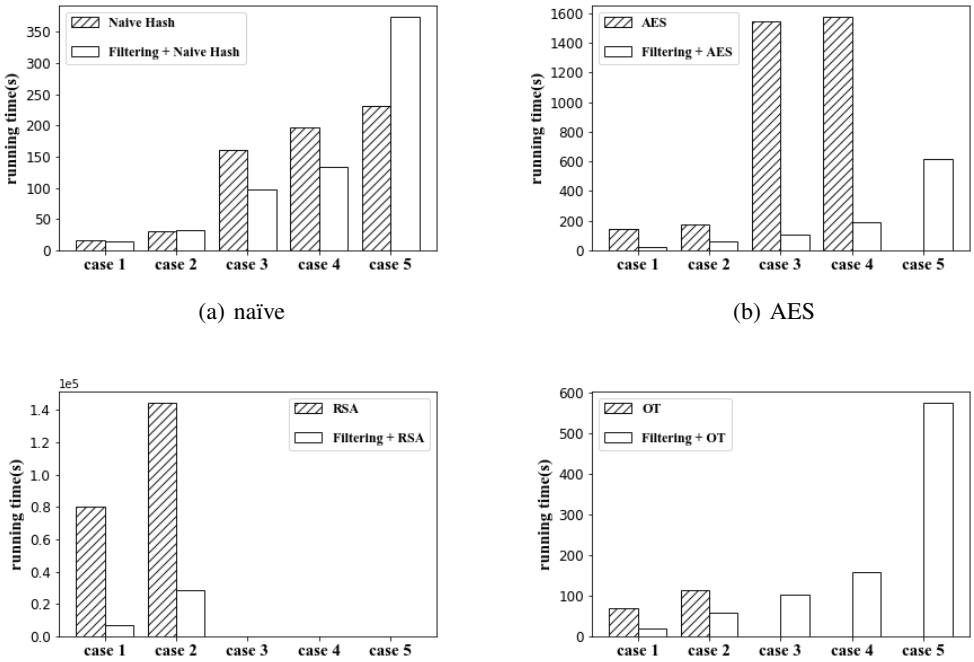
A Tables

Dataset	Ratio	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}
1	$ratio_{P_A}$	16.78	9.15	5.14	3.08					
	$ratio_{P_B}$	9.73	6.74	4.33	2.80					
2	$ratio_{P_A}$		6.91	4.43	2.85	1.96				
	$ratio_{P_B}$		6.91	4.43	2.85	1.96				
3	$ratio_{P_A}$				17.25	9.15	5.08	3.05		
	$ratio_{P_B}$				8.50	6.14	4.06	2.68		
4	$ratio_{P_A}$					8.67	4.91	2.98	1.99	
	$ratio_{P_B}$					6.06	4.01	2.66	1.87	
5	$ratio_{P_A}$						3.99	2.64	1.86	1.44
	$ratio_{P_B}$						3.99	2.64	1.86	1.44

Table 4: Ratio of the selected set size after filtering to the exact intersection size for different bloom filter size

Datasets		PSI				Filtering+PSI				
Case	Filter size	naïve Hash	AES	RSA	OT	Filtering	naïve Hash	AES	RSA	OT
1	filter=2^{24}	0.73	0.73	1.22	0.82	0.02	0.14	0.14	0.73	0.21
	filter=2^{25}					0.03	0.09	0.09	0.41	0.13
	filter=2^{26}					0.06	0.06	0.06	0.24	0.08
	filter=2^{27}					0.13	0.04	0.04	0.14	0.05
2	filter=2^{25}	1.41	1.41	5.95	2.35	0.03	0.78	0.78	3.12	1.21
	filter=2^{26}					0.06	0.54	0.54	2.02	0.78
	filter=2^{27}					0.13	0.38	0.38	1.32	0.50
	filter=2^{28}					0.25	0.29	0.29	0.92	0.34
3	filter=2^{27}	6.34	6.34	6.79	6.67	0.06	0.17	0.17	0.92	0.24
	filter=2^{28}					0.13	0.11	0.11	0.50	0.15
	filter=2^{29}					0.25	0.07	0.07	0.29	0.09
	filter=2^{30}					0.50	0.05	0.05	0.18	0.06
	filter=2^{31}					1.00	0.04	0.04	0.12	0.04
4	filter=2^{28}	7.04	7.04	11.56	8.42	0.13	1.06	1.06	4.85	1.54
	filter=2^{29}					0.25	0.69	0.69	2.82	0.96
	filter=2^{30}					0.50	0.49	0.49	1.75	0.62
	filter=2^{31}					1.00	0.37	0.37	1.20	0.43
5	filter=2^{29}	13.74	13.74	56.93	22.89	0.25	6.10	6.10	22.56	8.84
	filter=2^{30}					0.50	4.45	4.45	15.13	5.83
	filter=2^{31}					1.00	3.49	3.49	10.84	4.09
	filter=2^{32}					2.00	2.98	2.98	8.53	3.16

Table 5: Detailed Communication Cost for PSI protocols and Filtering+PSI protocols with different Filter size (GB)



(a) naïve (b) AES

(c) RSA (d) OT

Figure 1: Running Time for PSI protocols and Filtering+PSI protocols (s)