Private Set Intersection Analysis Protocol

Andrada-Teodora Ciulei, Andrei Miron, Stefania Andries, Marian Cretu

Adrian Iftene, Ph.D. , Sorin Iftene, Ph.D.

Alexandru Ioan Cuza University of Iasi, Faculty of Computer Science

1 Problem presentation

Private Set Intersection (PSI) is a cryptographic technique that allows two parties to compute the intersection of their datasets without revealing nothing more than the intersection result. Protocols of Private Set Intersection have a great importance and use in many fields because of their property of saving confidentiality of data. The cases where such a protocol should be used is whenever one or both parties have to discover information about the intersection of their sets which contain sensitive and private data.

To represent the problem in a more accurate way, we can consider Bob and Alice with two sets of data: X = $\{x_1, x_2,..., x_n\}$ - Bob's set and Y = $\{y_1, y_2,..., y_n\}$ - Alice's set.

There are various variants of PSI protocols where both Bob and Alice can compute the intersection of X and Y or only one of them can discover the intersection, in such cases in literature we can find the entities as sender and receiver (just the receiver will gain information). Hence, the problem can be represented not just in a unique way:

Input: X and Y, Bob and Alice sets **Output**: X ∩ Y for both Bob and Alice	**Input**: X and Y, Bob is the **SENDER** and Alice is the **RECEIVER** **Output**: Alice: X ∩ Y Bob: ∅

2 State of the art

In the last several years, protocols of Private Set Intersection have been vastly studied and many implementations with great results were given. When we look at the efficiency of a PSI protocol we have to consider two major aspects: the computation cost and the communication cost.

The first one is the necessary computing time to run the protocol and as we optimise it we can limit the computational resources. The computationally efficient protocols are based on Oblivious Transfer (OT), a cryptographic hash function, symmetric-key cryptographic operations and bitwise operations, [1], [2], [3] presents various OT implementations.

The second aspect refers to the amount of communication in the protocol and due to limited network bandwidth minimizing the communication cost becomes a critical side of efficiency. Combining these two aspects, computation and communication, a single metric can be computed: the total running time of the protocol, which includes both computation time and communication time. Obviously, this time can vary depending on the network settings so different protocols can reach a better time in different settings.

A great balance between computation time and communication time was reached in [4] by Pinkas. A variant of oblivious transfer extension that they call sparse OT was used. It relies heavily on manipulating high-degree polynomials over large finite fields.

[5] which is also our base reference in the proposed project, introduces the question if [4] achieves the best balance between computation cost and communication cost. The authors, Melissa Chase and Peihan Miao, come with a negative answer proposing a new PSI protocol which is a new multi-point oblivious pseudorandom function(OPRF) protocol based on oblivious transfer extension. This new protocol achieves a better balance between computation and communication which is based only on oblivious transfer, hashing, symmetric-key and bitwise operations.

3 Implementation

In this section, we will present our implementation for the PSI, followed by pictures of the application. Our implementation is based on a client-server architecture, where a single client can run the protocol with the server at any given time.

On the server side, before the server can be started the user must choose to what port and address the server will be bound and choose a file, csv or xlsx, from which he will be able to choose one of the columns present in the file to use the data from that column as data in the protocol. Once the server is started, it will wait for any incoming connection from a client. When a client connects, his address and port will be shown in the server interface, a example being illustrated in Figure 1.

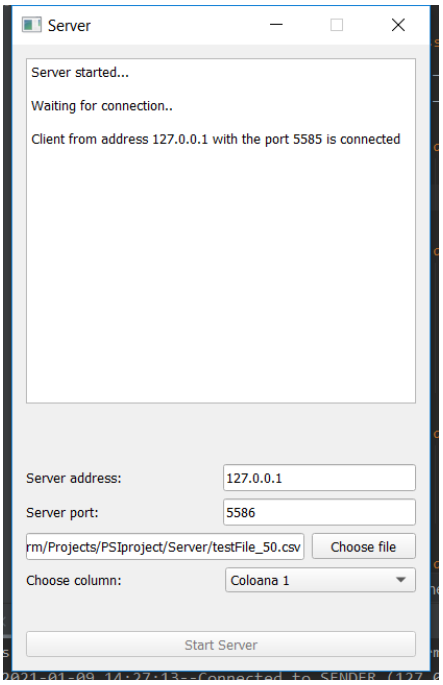


Figure 1: Server interface

On the client side, the user will have to go trough a connection tab, shown in Figure 2, where he must choose to what port and address the client will be bound and to offer the address and the port to the server they want to connect. Once the connection is validated, the user will be sent to a second tab, where he will choose what parameters he wants to use, similarly to the server choose a file and a column as seen in Figure 3. The second tab also has a big text browser where the results of the PSI will be showed, as seen in Figure 6.

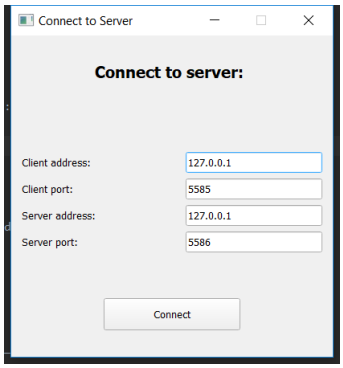


Figure 2: Connection interface

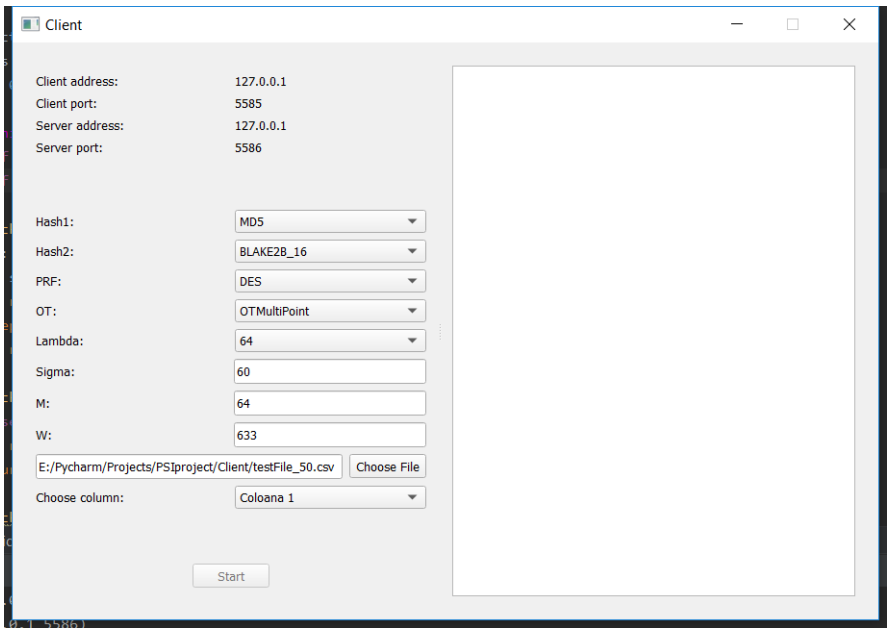


Figure 3: Initial client interface

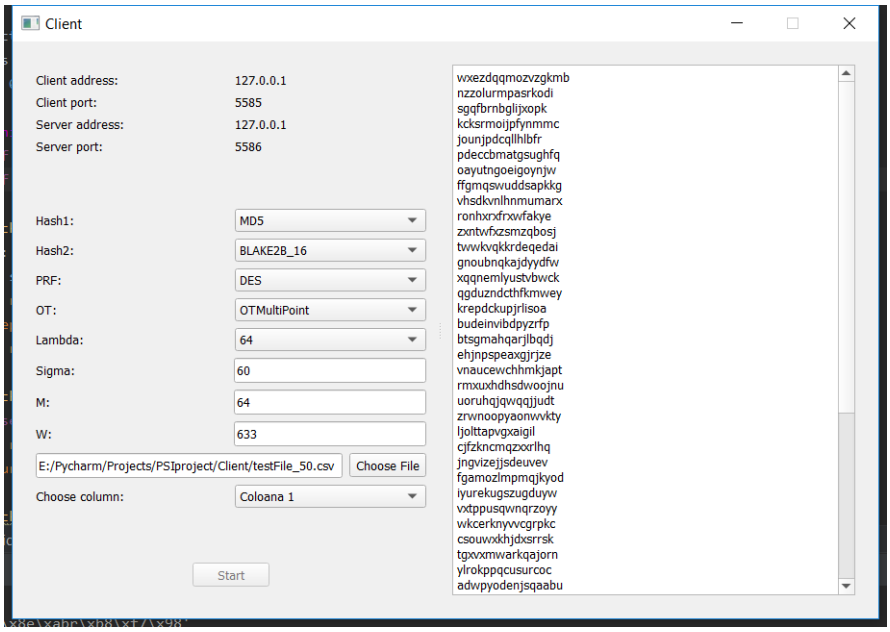


Figure 4: Client interface after executing the protocol

As we can see in the Figures 5 and 6, our application encapsulates the main algorithm service . The focus was on the nucleus algorithm and also on the whole application. The entire application was implemented in Python. For the application, we used *PyQt*5 for GUI, and for the whole communication service, we used the library *socket* from Python. The communication service is used for initiating connection, exchanging RSA keys, exchanging AES key/IV encrypted a priori with corresponding RSA keys, and for all protocol operations.

We incorporated some elements of advanced software engineering techniques, including aspects implemented in *aspectlib*. For example, we implemented with aspects an extension which allows to encrypt/decrypt the content transmitted through CommunicationService. We choose to implement this feature using aspects because it's an extra feature which can be discarded anytime in algorithm without much refactoring.

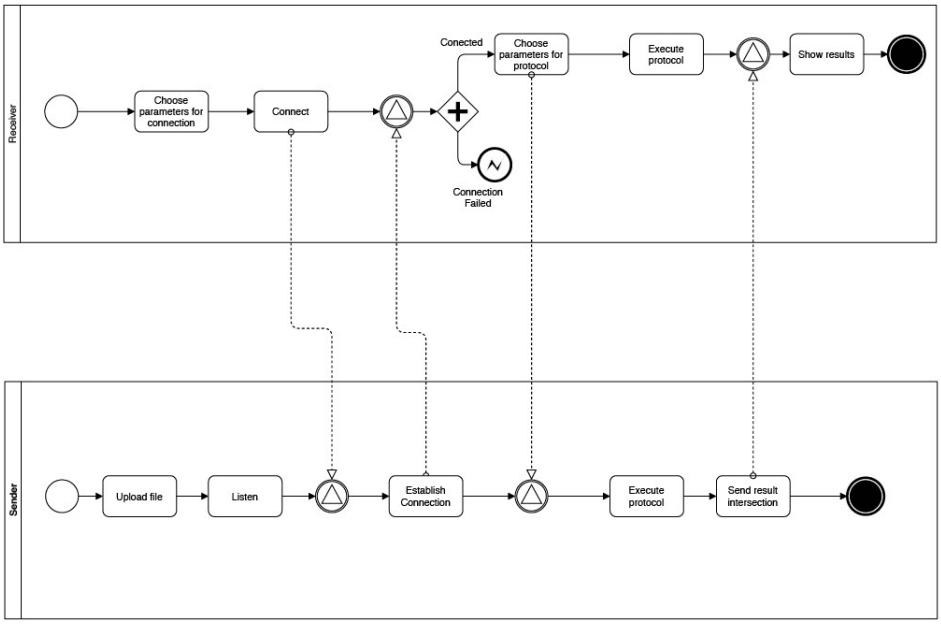


Figure 5: Diagram of the application

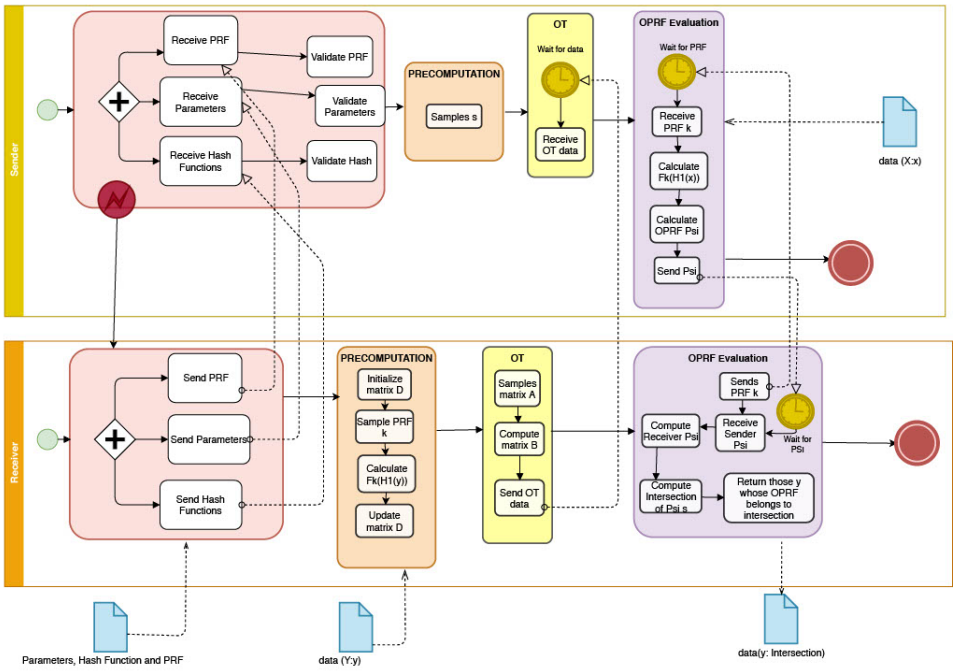


Figure 6: Nucleus algorithm BPMN diagram

A significant role in the entire protocol is played by the primitives that we can see as the bricks on which the entire protocol is built.

The smallest primitives we used are hash functions: BLAKE2, MD5, SHA1, SHA256, SHA384, SHA512, `SHA3_256`, `SHA3_384`, `SHA3_512` and ciphers: AES, DES and DES3 that used in different modes such as CTR, ECB or CBC play different roles: any cipher in the CTR mode is considered to be a psudorandom generator (we note it with PRG), while using it in the ECB mode represents a pseudorandom function (we note it with G). The CBC mode is considered to be a generic mode.

All of these small primitives were implemented using *Pycryptodome* library available in Python 3.

In the following, we present the scheme for the F function (applied on a key k and input $x = x_0||x_1$) used in our implementation, which was taken from the aforementioned article.

$F_k(x) = G_{k_1}(G_{k_0}(x_0) \oplus x_1)||G_{k_2}(G_{k_0}(x_0) \oplus x_1)|| \dots ||G_{k_t}(G_{k_0}(x_0) \oplus x_1)$

where $G : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$,

$PRG : \{0,1\}^\lambda \to \{0,1\}^{(t+1)\cdot\lambda}$, $t = \left\lceil \frac{w \cdot log(m)}{\lambda} \right\rceil$

and $k_0||k_1|| \dots ||k_t \leftarrow PRG(k)$.

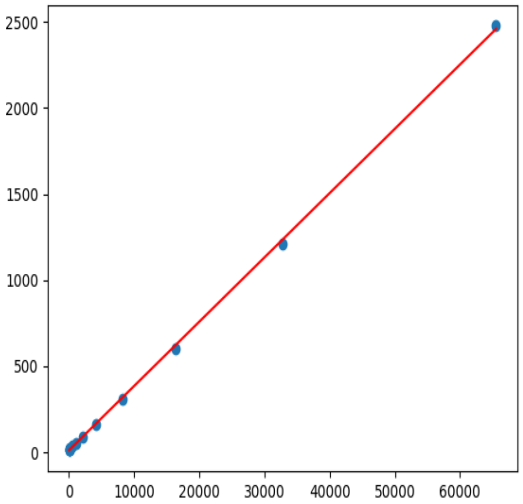
4 Results, Measurements and Statistics

In this section, we will present the measurements we did on our implemented protocol in Python and with our desired architecture. Keep in mind that the results measured are only for algorithm, included the security features (AES encryption of the channel). Our scope is to prove statistically the pseudoliniarity of the protocol, and to get the best optimal combination of primitives.

First of all, we would like to test that our protocol is linear. So, we have some data scaling reports for the following constant hyperparameters: $lambda = 128$, $m = 256$, $w = 633$, $hash1 = SHA256$, $hash2 = MD5$, $prf = AES$, $otVariant = 1$:

Table 1: Measurements for $n = len(data) = 512$

n	Time
8	16.907427072525024
16	17.147517442703247
32	18.162928342819214
64	18.556200742721558
128	20.88942790031433
256	25.276923894882202
512	34.543391942977905
1024	53.31533193588257
2048	89.99437379837036
4096	162.95270085334778
8192	308.8733296394348
16384	603.2397153377533
32768	1212.2749524116516
65536	2477.1293470859528



The results below are for the following hyperparameters set: $w = 100, m = len(data)$

Table 2: Measurements for $n = len(data) = 512$

Hash1	Hash2	PRF	Time OT1	Time OT2
MD5	MD5	DES	11.63084864616394	11.165813446044922
MD5	BLAKE2B_16	DES	10.806557416915894	11.301727533340454
MD5	SHA3_256	DES	11.29605484008789	11.225828409194946
MD5	SHA256	DES	11.358386039733887	11.593780279159546
MD5	BLAKE2B_32	DES	11.390985012054443	11.261141061782837
MD5	BLAKE2B_48	DES	11.363512992858887	10.883297681808472
MD5	SHA384	DES	11.396617650985718	11.427547216415405
MD5	SHA3_384	DES	11.408205032348633	11.179348707199097
BLAKE2B_16	MD5	DES	11.37966012954712	11.381012916564941
BLAKE2B_16	BLAKE2B_16	DES	15.250828504562378	14.502021312713623
BLAKE2B_16	SHA3_256	DES	12.624895095825195	12.397151470184326
BLAKE2B_16	SHA256	DES	12.551755666732788	11.058485984802246
BLAKE2B_16	BLAKE2B_32	DES	11.855385541915894	11.520566940307617
BLAKE2B_16	BLAKE2B_48	DES	11.44810962677002	11.09655213356018
BLAKE2B_16	SHA384	DES	11.083648681640625	11.508599281311035
BLAKE2B_16	SHA3_384	DES	11.434541463851929	11.695915460586548
SHA3_256	MD5	AES	9.624945878982544	9.653274774551392
SHA3_256	BLAKE2B_16	AES	9.498143672943115	9.660670042037964
SHA3_256	SHA3_256	AES	9.575755834579468	9.664479494094849
SHA3_256	SHA256	AES	9.3541738986969	9.652188062667847
SHA3_256	BLAKE2B_32	AES	9.516450881958008	9.673266887664795
SHA3_256	BLAKE2B_48	AES	9.692655801773071	10.384363651275635
SHA3_256	SHA384	AES	9.971112728118896	11.308344602584839
SHA3_256	SHA3_384	AES	11.14855432510376	11.312012195587158
SHA256	MD5	AES	11.035212993621826	9.56265377998352
SHA256	BLAKE2B_16	AES	9.69684624671936	9.807061672210693
SHA256	SHA3_256	AES	9.76620364189148	9.760502099990845
SHA256	SHA256	AES	9.129978656768799	9.613179922103882
SHA256	BLAKE2B_32	AES	9.710575103759766	9.64811372756958
SHA256	BLAKE2B_48	AES	9.522791147232056	9.692003011703491
SHA256	SHA384	AES	9.475827932357788	9.755620241165161
SHA256	SHA3_384	AES	9.653841257095337	9.736819505691528
BLAKE2B_32	MD5	AES	9.660560369491577	9.690344333648682
BLAKE2B_32	BLAKE2B_16	AES	9.52011752128601	10.004932403564453
BLAKE2B_32	SHA3_256	AES	10.054861068725586	10.225496530532837
BLAKE2B_32	SHA256	AES	11.443130254745483	11.676531076431274
BLAKE2B_32	BLAKE2B_32	AES	11.332979679107666	11.87242579460144
BLAKE2B_32	BLAKE2B_48	AES	11.182630777359009	10.813763618469238
BLAKE2B_32	SHA384	AES	11.609355211257935	11.807516813278198
BLAKE2B_32	SHA3_384	AES	10.833362340927124	10.089388370513916
BLAKE2B_48	MD5	DES3	12.345481395721436	11.038431882858276
BLAKE2B_48	BLAKE2B_16	DES3	11.101988077163696	10.825244426727295
BLAKE2B_48	SHA3_256	DES3	11.273300409317017	11.501542806625366
BLAKE2B_48	SHA256	DES3	11.297710180282593	11.899436473846436
BLAKE2B_48	BLAKE2B_32	DES3	11.383509159088135	11.38988208770752
BLAKE2B_48	BLAKE2B_48	DES3	12.092130422592163	11.104553461074829
BLAKE2B_48	SHA384	DES3	10.990689277648926	10.996483087539673
BLAKE2B_48	SHA3_384	DES3	11.132358074188232	10.602033376693726
SHA384	MD5	DES3	10.97469973564148	10.842397689819336
SHA384	BLAKE2B_16	DES3	10.756850242614746	11.001320600509644
SHA384	SHA3_256	DES3	10.593814849853516	10.607944011688232
SHA384	SHA256	DES3	10.343361377716064	10.805780410766602
SHA384	BLAKE2B_32	DES3	10.476927280426025	10.776776552200317
SHA384	BLAKE2B_48	DES3	10.511518239974976	10.688711166381836
SHA384	SHA384	DES3	10.547427415847778	10.747191429138184
SHA384	SHA3_384	DES3	10.479281663894653	10.416140079498291
SHA3_384	MD5	DES3	11.071538925170898	11.114330530166626
SHA3_384	BLAKE2B_16	DES3	11.257049560546875	11.116895198822021
SHA3_384	SHA3_256	DES3	10.590781450271606	10.556920528411865
SHA3_384	SHA256	DES3	10.088867902755737	10.771724224090576
SHA3_384	BLAKE2B_32	DES3	10.29731559753418	11.003913402557373
SHA3_384	BLAKE2B_48	DES3	10.741192817687988	10.528929948806763
SHA3_384	SHA384	DES3	10.971017122268677	10.793152570724487
SHA3_384	SHA3_384	DES3	10.731109142303467	10.610780000686646

Table 3: Measurements for $n = len(data) = 1024$

Hash1	Hash2	PRF	Time OT1	Time OT2
MD5	MD5	DES	17.225221157073975	17.37646794319153
MD5	BLAKE2B_16	DES	17.357712745666504	17.176451921463013
MD5	SHA3_256	DES	18.73711633682251	18.820461988449097
MD5	SHA256	DES	18.930413722991943	18.973309993743896
MD5	BLAKE2B_32	DES	19.520297527313232	19.06158971786499
MD5	BLAKE2B_48	DES	19.136394262313843	18.81047797203064
MD5	SHA384	DES	19.011428833007812	19.30599284172058
MD5	SHA3_384	DES	19.390262842178345	18.923003673553467
BLAKE2B_16	MD5	DES	18.896342039108276	20.581936836242676
BLAKE2B_16	BLAKE2B_16	DES	19.037595510482788	18.74752426147461
BLAKE2B_16	SHA3_256	DES	18.272668600082397	18.75015902519226
BLAKE2B_16	SHA256	DES	17.97750163078308	19.056209325790405
BLAKE2B_16	BLAKE2B_32	DES	18.956537008285522	18.01084327697754
BLAKE2B_16	BLAKE2B_48	DES	18.354613065719604	18.20868158340454
BLAKE2B_16	SHA384	DES	18.04795813560486	17.961941242218018
BLAKE2B_16	SHA3_384	DES	18.624703407287598	17.214723110198975
SHA3_256	MD5	AES	14.249455690383911	13.552728652954102
SHA3_256	BLAKE2B_16	AES	13.53704833984375	13.486570119857788
SHA3_256	SHA3_256	AES	13.22245168685913	12.991517782211304
SHA3_256	SHA256	AES	13.121551513671875	13.286731719970703
SHA3_256	BLAKE2B_32	AES	13.043191909790039	13.130682468414307
SHA3_256	BLAKE2B_48	AES	12.448774337768555	12.910959959030151
SHA3_256	SHA384	AES	12.841716527938843	13.243304014205933
SHA3_256	SHA3_384	AES	13.004760980606079	13.278780937194824
SHA256	MD5	AES	12.364851474761963	13.152294635772705
SHA256	BLAKE2B_16	AES	13.343289375305176	13.30451512336731
SHA256	SHA3_256	AES	13.345860481262207	13.32010793685913
SHA256	SHA256	AES	13.381382942199707	13.224286556243896
SHA256	BLAKE2B_32	AES	13.49772572517395	13.20004940032959
SHA256	BLAKE2B_48	AES	12.7383451461792	13.284029006958008
SHA256	SHA384	AES	12.874261140823364	13.142417192459106
SHA256	SHA3_384	AES	13.306595087051392	13.473414182662964
BLAKE2B_32	MD5	AES	13.296676874160767	13.21343731880188
BLAKE2B_32	BLAKE2B_16	AES	13.143001079559326	13.301005125045776
BLAKE2B_32	SHA3_256	AES	13.032232284545898	13.055453062057495
BLAKE2B_32	SHA256	AES	13.354897022247314	13.248682498931885
BLAKE2B_32	BLAKE2B_32	AES	13.284666776657104	13.433532238006592
BLAKE2B_32	BLAKE2B_48	AES	13.302583694458008	13.223508358001709
BLAKE2B_32	SHA384	AES	13.284364223480225	13.285138368606567
BLAKE2B_32	SHA3_384	AES	13.078558206558228	13.15620756149292
BLAKE2B_48	MD5	DES3	17.1925151348114	17.36695170402527
BLAKE2B_48	BLAKE2B_16	DES3	17.08529496192932	17.29667329788208
BLAKE2B_48	SHA3_256	DES3	17.393718481063843	17.125810861587524
BLAKE2B_48	SHA256	DES3	16.77791404724121	17.403290271759033
BLAKE2B_48	BLAKE2B_32	DES3	17.558778285980225	17.45000195503235
BLAKE2B_48	BLAKE2B_48	DES3	17.455129146575928	17.106792449951172
BLAKE2B_48	SHA384	DES3	16.85513186454773	17.257157564163208
BLAKE2B_48	SHA3_384	DES3	17.69700860977173	17.418318510055542
SHA384	MD5	DES3	17.31925082206726	17.02530074119568
SHA384	BLAKE2B_16	DES3	17.406959533691406	17.23665189743042
SHA384	SHA3_256	DES3	17.407065391540527	17.677969932556152
SHA384	SHA256	DES3	17.166463613510132	17.356370449066162
SHA384	BLAKE2B_32	DES3	17.032288074493408	17.262125253677368
SHA384	BLAKE2B_48	DES3	17.26066565513611	17.514639854431152
SHA384	SHA384	DES3	17.247604846954346	17.210556268692017
SHA384	SHA3_384	DES3	17.21727681159973	17.02304720878601
SHA3_384	MD5	DES3	17.493645429611206	17.134007215499878
SHA3_384	BLAKE2B_16	DES3	17.18049907684326	17.20743203163147
SHA3_384	SHA3_256	DES3	17.59969687461853	17.133952856063843
SHA3_384	SHA256	DES3	17.480902910232544	17.208557605743408
SHA3_384	BLAKE2B_32	DES3	17.598394870758057	17.554622173309326
SHA3_384	BLAKE2B_48	DES3	17.462876081466675	17.475228309631348
SHA3_384	SHA384	DES3	17.2588312625885	17.119513750076294
SHA3_384	SHA3_384	DES3	16.785064220428467	17.20242714881897

Table 4: Measurements for $n = len(data) = 2048$

Hash1	Hash2	PRF	Time OT1	Time OT2
MD5	MD5	DES	31.462194204330444	31.181671380996704
MD5	BLAKE2B_16	DES	31.564910650253296	31.5129714012146
MD5	SHA3_256	DES	33.95671105384827	34.11079812049866
MD5	SHA256	DES	31.75911593437195	31.516820430755615
MD5	BLAKE2B_32	DES	31.837295293807983	31.702070951461792
MD5	BLAKE2B_48	DES	31.086499452590942	31.874284744262695
MD5	SHA384	DES	32.23031687736511	31.78095531463623
MD5	SHA3_384	DES	32.11810064315796	31.464161157608032
BLAKE2B_16	MD5	DES	32.39213848114014	31.87548041343689
BLAKE2B_16	BLAKE2B_16	DES	32.64737582206726	31.534859657287598
BLAKE2B_16	SHA3_256	DES	31.14119791984558	31.140640258789062
BLAKE2B_16	SHA256	DES	31.14320683479309	32.356513261795044
BLAKE2B_16	BLAKE2B_32	DES	31.355746030807495	31.284966468811035
BLAKE2B_16	BLAKE2B_48	DES	31.265866994857788	30.79434370994568
BLAKE2B_16	SHA384	DES	31.2282497882843	33.49408316612244
BLAKE2B_16	SHA3_384	DES	31.69678521156311	31.681474447250366
SHA3_256	MD5	AES	21.515230417251587	22.030182123184204
SHA3_256	BLAKE2B_16	AES	21.455468893051147	22.11597466468811
SHA3_256	SHA3_256	AES	21.460983514785767	21.527307271957397
SHA3_256	SHA256	AES	23.100158214569092	21.97428321838379
SHA3_256	BLAKE2B_32	AES	21.713921070098877	21.80008101463318
SHA3_256	BLAKE2B_48	AES	21.778438806533813	22.692944288253784
SHA3_256	SHA384	AES	22.614890813827515	22.629551887512207
SHA3_256	SHA3_384	AES	22.76878571510315	21.70355224609375
SHA256	MD5	AES	21.20930290222168	21.654982328414917
SHA256	BLAKE2B_16	AES	21.530063152313232	21.818922996520996
SHA256	SHA3_256	AES	22.056843042373657	21.856691598892212
SHA256	SHA256	AES	21.785206079483032	21.776365518569946
SHA256	BLAKE2B_32	AES	21.54906129837036	21.688472270965576
SHA256	BLAKE2B_48	AES	21.772254943847656	21.76401138305664
SHA256	SHA384	AES	21.61393713951111	23.414042711257935
SHA256	SHA3_384	AES	22.665583610534668	23.97363543510437
BLAKE2B_32	MD5	AES	22.812836408615112	21.858166694641113
BLAKE2B_32	BLAKE2B_16	AES	21.485961198806763	21.713208198547363
BLAKE2B_32	SHA3_256	AES	21.500590801239014	22.29860234260559
BLAKE2B_32	SHA256	AES	21.95273995399475	21.593382596969604
BLAKE2B_32	BLAKE2B_32	AES	20.977794647216797	21.767558813095093
BLAKE2B_32	BLAKE2B_48	AES	21.828673839569092	21.82671093940735
BLAKE2B_32	SHA384	AES	21.38731050491333	21.5998797416687
BLAKE2B_32	SHA3_384	AES	21.85459804534912	21.545931100845337
BLAKE2B_48	MD5	DES3	28.408084630966187	28.116218328475952
BLAKE2B_48	BLAKE2B_16	DES3	28.140973567962646	28.744975566864014
BLAKE2B_48	SHA3_256	DES3	30.150547981262207	28.757289171218872
BLAKE2B_48	SHA256	DES3	29.147310972213745	28.83419132232666
BLAKE2B_48	BLAKE2B_32	DES3	28.06528925895691	29.300199031829834
BLAKE2B_48	BLAKE2B_48	DES3	28.45148253440857	28.47708225250244
BLAKE2B_48	SHA384	DES3	28.285056114196777	28.377455711364746
BLAKE2B_48	SHA3_384	DES3	28.63585376739502	29.48725128173828
SHA384	MD5	DES3	28.024115800857544	28.63653802871704
SHA384	BLAKE2B_16	DES3	27.991986989974976	28.0236554145813
SHA384	SHA3_256	DES3	28.38384771347046	28.395939111709595
SHA384	SHA256	DES3	29.620504140853882	28.091763496398926
SHA384	BLAKE2B_32	DES3	29.386098861694336	28.753032207489014
SHA384	BLAKE2B_48	DES3	28.09493374824524	27.79188060760498
SHA384	SHA384	DES3	29.480567693710327	28.049057245254517
SHA384	SHA3_384	DES3	28.11603593826294	28.42272686958313
SHA3_384	MD5	DES3	28.23486638069153	29.417449951171875
SHA3_384	BLAKE2B_16	DES3	28.352436065673828	28.30423355102539
SHA3_384	SHA3_256	DES3	28.212011098861694	28.29524517059326
SHA3_384	SHA256	DES3	27.88572883605957	28.20648193359375
SHA3_384	BLAKE2B_32	DES3	27.91737174987793	28.05398201942444
SHA3_384	BLAKE2B_48	DES3	28.17871880531311	28.263413667678833
SHA3_384	SHA384	DES3	29.460013151168823	28.12145757675171
SHA3_384	SHA3_384	DES3	28.35428762435913	28.43854594230652

Table 5: Measurements for $n = len(data) = 4096$

Hash1	Hash2	PRF	Time OT1	Time OT2
MD5	MD5	DES	58.19271230697632	60.49761748313904
MD5	BLAKE2B_16	DES	58.25435709953308	58.903547048568726
MD5	SHA3_256	DES	58.85389256477356	57.826281785964966
MD5	SHA256	DES	57.75263071060181	59.08253359794617
MD5	BLAKE2B_32	DES	58.85891819000244	59.26866054534912
MD5	BLAKE2B_48	DES	59.33717751502991	58.870731592178345
MD5	SHA384	DES	58.513051986694336	59.45649456977844
MD5	SHA3_384	DES	58.13161826133728	59.10706353187561
BLAKE2B_16	MD5	DES	58.28734517097473	58.5336971282959
BLAKE2B_16	BLAKE2B_16	DES	58.8207950592041	58.52919387817383
BLAKE2B_16	SHA3_256	DES	58.59842276573181	59.27908802032471
BLAKE2B_16	SHA256	DES	58.601120948791504	59.69383692741394
BLAKE2B_16	BLAKE2B_32	DES	58.24229907989502	58.50032448768616
BLAKE2B_16	BLAKE2B_48	DES	59.00385928153992	59.23209834098816
BLAKE2B_16	SHA384	DES	58.90710806846619	59.01182579994202
BLAKE2B_16	SHA3_384	DES	59.26155662536621	59.726070165634155
SHA3_256	MD5	AES	40.33641219139099	40.628509283065796
SHA3_256	BLAKE2B_16	AES	40.26899600028992	39.99667286872864
SHA3_256	SHA3_256	AES	41.89494180679321	40.25378203392029
SHA3_256	SHA256	AES	40.17916560173035	40.52242398262024
SHA3_256	BLAKE2B_32	AES	40.15518641471863	40.427775144577026
SHA3_256	BLAKE2B_48	AES	40.259679317474365	39.98810434341431
SHA3_256	SHA384	AES	40.955002307891846	40.89901638031006
SHA3_256	SHA3_384	AES	40.86719727516174	40.423604249954224
SHA256	MD5	AES	39.9898624420166	40.61107635498047
SHA256	BLAKE2B_16	AES	39.96238589286804	40.39533734321594
SHA256	SHA3_256	AES	40.46559715270996	40.452024936676025
SHA256	SHA256	AES	39.88452887535095	40.54161238670349
SHA256	BLAKE2B_32	AES	40.32319211959839	40.424076318740845
SHA256	BLAKE2B_48	AES	40.16559386253357	40.09137487411499
SHA256	SHA384	AES	40.45912432670593	40.59420394897461
SHA256	SHA3_384	AES	40.3824508190155	40.22903656959534
BLAKE2B_32	MD5	AES	40.284011363983154	40.476752281188965
BLAKE2B_32	BLAKE2B_16	AES	40.44367980957031	39.84769368171692
BLAKE2B_32	SHA3_256	AES	39.629249572753906	40.93134355545044
BLAKE2B_32	SHA256	AES	39.930094957351685	40.31002736091614
BLAKE2B_32	BLAKE2B_32	AES	40.07827091217041	41.99993443489075
BLAKE2B_32	BLAKE2B_48	AES	40.55048131942749	40.38550662994385
BLAKE2B_32	SHA384	AES	40.04999041557312	40.672343015670776
BLAKE2B_32	SHA3_384	AES	40.18826222419739	40.564993381500244
BLAKE2B_48	MD5	DES3	57.5276517868042	57.17940807342529
BLAKE2B_48	BLAKE2B_16	DES3	57.37957572937012	57.02113628387451
BLAKE2B_48	SHA3_256	DES3	57.28379702568054	57.15096068382263
BLAKE2B_48	SHA256	DES3	56.91145634651184	57.16837954521179
BLAKE2B_48	BLAKE2B_32	DES3	56.48204731941223	57.11736488342285
BLAKE2B_48	BLAKE2B_48	DES3	56.89824175834656	56.92065119743347
BLAKE2B_48	SHA384	DES3	56.246726274490356	57.469489336013794
BLAKE2B_48	SHA3_384	DES3	56.916229486465454	57.18917655944824
SHA384	MD5	DES3	56.815810203552246	56.45619010925293
SHA384	BLAKE2B_16	DES3	56.71985602378845	58.484485149383545
SHA384	SHA3_256	DES3	58.69017052650452	57.198150396347046
SHA384	SHA256	DES3	60.11647129058838	58.7365505695343
SHA384	BLAKE2B_32	DES3	56.72756481170654	57.06016397476196
SHA384	BLAKE2B_48	DES3	57.15867900848389	57.447484493255615
SHA384	SHA384	DES3	57.12779378890991	59.00008225440979
SHA384	SHA3_384	DES3	57.04750728607178	57.47371172904968
SHA3_384	MD5	DES3	57.44695711135864	56.52378964424133
SHA3_384	BLAKE2B_16	DES3	57.287219762802124	58.11640667915344
SHA3_384	SHA3_256	DES3	56.68175411224365	56.589179039001465
SHA3_384	SHA256	DES3	56.8877911567688	57.2078058719635
SHA3_384	BLAKE2B_32	DES3	56.18602442741394	57.40674376487732
SHA3_384	BLAKE2B_48	DES3	57.09860587120056	56.68674969673157
SHA3_384	SHA384	DES3	56.919158697128296	57.263962507247925
SHA3_384	SHA3_384	DES3	56.791351079940796	59.98237466812134

5 Comparison with other solutions

Our goal so far has been to implement a version of the Private set intersection protocol that is robust and efficient enough. The main reference, for this purpose, is *Private Set Intersection in the Internet Setting From Lightweight Oblivious PRF* published by Melissa Chase and Peihan Miao and presented at *Advances in cryptology – CRYPTO 2020*. We decided that this approach is the most appropriate choice because it is based only on oblivious transfer, hasing, symmetric-key and bitwise operations, so it is facil to understand and implement, it was published in 2020, hence it belongs to actual interest and, summarizing the results contained in the article, it achieves a better balance between computation and communication than all the other existent PSI protocols. It is also mentioned in the article that this protocol is the fastest in networks with moderate bandhwidth and, in terms of monetary cost, it compares favourably when running it on a cloud computing service. Underlying the PSI protocol described in the reference, is a new lightweight multi-point pseudorandom function (OPRF) protocol based on oblivious transfer (OT) extension which can also be found in the reference.

In the article, this construction is based on the BLAKE2 hash function (whose result is up to 256 bits) and AES (128 bits).

As an element of novelty for this construction, we diversified the OPRF protocol using BLAKE2, MD5, SHA1, SHA256, SHA384, SHA512, `SHA3_256`, `SHA3_384`, `SHA3_512` as hash functions and AES, DES, DES3 in different modes as pseudorandom generators and pseudorandom functions. The main advantage of this option would be the user flexibility of choosing between multiple sets of parameters and the efficiency of the protocol.

From the implementation point of view, despite the performance and memory efficiency, we decided to use Python 3 instead of C++ as a programming language because the syntax makes the code easy to read and develop, it allowed us to manage a big amount of data with an easy and cost-effective way and the most important, Python comes with many pre-built libraries, which made the development process easy. Consequently, the most relevant elements that we used are *Pycrypyodome*, a self-contained Python package of low-level cryptographic primitives for the pseudorandom functions and generators implementation and *hashlib* for the hash functions.

Although the Python language provides *oblivious*, a library that serves as an API for common primitives used to implement OPRF and OT protocols, we decided to implement these from scratch for a better understanding of the protocol and the most significant primitives. In the implementation of all communication protocol stages, we used the Python library *socket*. However, this opens the way to an even richer diversification of the protocol in the future.

6 Future work

Our future work plan would consist, primarily, in developing a multi-threaded implementation for the communication protocol that allows connecting several clients to the server at the same time.

Furthermore, we can expand the current version of the implementation by finding more cryptographic primitives. For instance, we could develop another version of the F function or use the implementations for $OPRF$ and OT from the library provided by $Python3$, named *oblivious*. Another example would be to find other hash functions and to build different pseudorandom functions and generators.

On the other hand, another direction of development we thought of, would be to try implementing the entire protocol in the $C++$ programming language in order to gain more efficiency.

Finally, we might try a direction which implies a Map-Reduce approach, on multi-nodes distributed system (we would chunk the sender data in multiple nodes to apply the Map-Reduce paradigm). We would like to explore here some different perspectives about optimization, with applicability in Big Data.

7 Conclusions

In conclusion, we succeeded to port the C/C++ variant of the protocol, proposed by the authors of the main article, on the Python in a conventional way. Keeping in mind the language used and the conventional way of implementing the OT, we remark that the obtained statistics are satisfactory. If we are looking into **Section 4: Results, Measurements and Statistics**, at all tables, we conclude that the combinations with Hash1 in {SHA256, SHA3_256, BLAKE2B_32} with PRF construction based on AES, are the most efficient ways of building our algorithm.

References

(1) C. Dong L. Chen, Z. W., *When private set intersection meets big data: an efficient and scalable protocol*, https://eprint.iacr.org/2013/515.pdf, 2013.

(2) B. Pinkas T. Schneider, M. Z., *Faster private set intersection based on {OT} extension*, https://eprint.iacr.org/2014/447.pdf, 2014.

(3) B. Pinkas T Schneider, G. S., *Phasing: Private set intersection using permutation-based hashing*, https://eprint.iacr.org/2015/634.pdf, 2015.

(4) B. Pinkas M. Rosulek, N. T., *Spot-light: Lightweight private set intersection from sparse ot extension*, https://eprint.iacr.org/2019/634.pdf, 2019.

(5) M. Chase, P. M., *Private Set Intersection in the Internet Setting From Lightweight Oblivious PRF*, https://eprint.iacr.org/2020/729.pdf, 2020.