Scalable Private Set Intersection Cardinality for Capture-Recapture with Multiple Private Datasets

Sebastian Zander, Lachlan L. H. Andrew, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 130930A
Swinburne University of Technology
Melbourne, Australia
szander@swin.edu.au, landrew@swin.edu.au, garmitage@swin.edu.au

***Abstract*—In many scenarios it is beneficial to share private data among parties without mutual trust. One such scenario is the use of capture-recapture (CR) techniques to estimate population sizes from multiple private data sources of observed individuals. With Private Set Intersection Cardinality (PSIC) techniques, we can use CR with datasets from multiple parties while ensuring the privacy of the data of each party. However, existing PSIC techniques have limitations, for example they only work for two parties, and they do not scale well for large datasets. We propose an improved technique based on commutative encryption and deterministic hash-based sampling that is secure *and* scalable, and also prevents so-called probing attacks. We demonstrate with a prototype that our technique scales easily to datasets of at least 1–2 billion entries at the cost of a small sampling error.**

***Index Terms*—Secure set intersection cardinality, capture-recapture**

I. Introduction

In many scenarios it is beneficial to share private data among parties without mutual trust. One scenario is the application of capture-recapture (CR) techniques. CR techniques are used to estimate the size of populations based on multiple incomplete data sources of observed individuals. The population size is estimated from the number of individuals in each source and the *sizes of intersections* between all combinations of sources. CR arose in ecology [1]–[2], but is now also widely used in epidemiology [3]–[4], as well as in information technology [5]. A private set intersection cardinality (PSIC) technique would allow using CR with datasets from multiple parties while ensuring the privacy of the data of each party.

As a concrete example, we used CR to estimate the population of actively used IP version 4 (IPv4) addresses in the Internet from multiple data sources, such as server logs or traffic data [6]. Most of the IPv4 address space has already been *allocated* [7], but it is unclear how many allocated addresses are *actively used*. Knowing how many IPv4 addresses are used is important to predict the value and costs of an IPv4 address market and the timing of the transition to IP version 6 (IPv6). A diverse set of data sources from multiple collaborators is required to achieve Internet-wide “coverage”. A PSIC protocol allows collaborators to contribute to this study while ensuring anonymity of the IP addresses each collaborator has observed.

The PSIC approach would also be useful in other areas where CR is used. For example, in epidemiology CR is used to estimate populations of people with certain illnesses or drug consumption behaviours using data sources, such as patient records or police records. Typically the matching of the records is based on personal data, such as names or birth dates. With a PSIC protocol CR could be used in epidemiology while protecting the privacy of individuals.

Our CR-driven requirements on the PSIC protocol are as follows. The PSIC protocol must not rely on trusted third parties, it must work with two or more parties (but the maximum number of parties would be relatively small, say 10–20 parties), and it must work with large datasets of at least up to 1–2 billion entries. The PSIC protocol must be computationally-secure and either must be conceptually simple or a well proven technique to allow for easy verification of its security. Since the number of participants is small and there would be no anonymous parties, security under the semi-honest (also called honest-but-curious) adversary model is sufficient.

A number of PSIC protocols have been proposed [8]–[14], but they do not meet our requirements. Some only work for two parties, some only work for more than two parties, and all of them do not scale well for large datasets. We propose an improved technique based on commutative encryption [11] and hash-based sampling

[15] that is computationally-secure, prevents so-called probing attacks and works for two or more semi-honest parties without the need for a trusted third party. Besides CR there are other applications where our approach could be used, such as secure customer data sharing between multiple organisations.

Our technique is a tradeoff between performance and precision of the estimated set intersection size. It significantly improves the scalability of existing approaches at the cost of a small sampling error. Based on an implemented prototype we will demonstrate that our algorithm scales well with datasets of up to at least 1–2 billion items with acceptable sampling errors. We will also show that the impact of the sampling on the CR estimates is tolerable: even with sample rates as low as 0.1% the error in the CR estimates is under 5%.

The key contributions of our work are:

1) We propose an improved secure and scalable PSIC algorithm for two or more semi-honest parties. We calculate the resulting sampling error.
2) We propose a novel approach to defend against probing attacks that works if parties can agree on the valid dataset items and this valid set is not prohibitively large.
3) We implemented a publicly available prototype [16], and use it to demonstrate the security, correctness and scalability of our proposed approach.
4) We analyse the impact of the proposed sampled PSIC technique on the accuracy of the CR approach developed in [6].

The report is organised as follows. Section II discusses related work. Section III provides an overview of the approach and its underlying key techniques. Section IV describes our proposed algorithm in detail. Section V describes our prototype implementation. Section VI provides performance measurements obtained with our prototype and also analyses the error introduced by the sampling (including the impact of the sampling on CR population estimates). Section VII concludes and outlines future work.

II. Related Work

In recent years several techniques have been published to solve the PSIC problem. PSIC algorithms fall into three categories depending on what kind of underlying approach they use: commutative encryption, homomorphic encryption, or secret sharing.

Agrawal *et al.* [8] proposed a PSIC technique based on commutative encryption, such as Pohlig-Hellman (PH) encryption or commutative RSA. Their scheme was limited to the client-server two-party scenario. Vaidya and Clifton [11] extended the technique to more than two parties and also proposed an optimisation that increases the efficiency and reduces the information disclosure for three or more parties. However, if there can be collusion the optimised approach is not secure. De Cristofaro *et al.* [14] proposed a variant of [8], which 1) adds an authorisation for client input data by a mutually-trusted authority and 2) combines private set intersection (PSI) with PSIC, so that the server can decide whether the client can learn the intersecting set. They also proved that the protocol is secure in the presence of semi-honest adversaries. The commutative encryption based technique has $O\left(k^2 N\right)$ total communication and computational complexity given k parties and datasets of size N.

The homomorphic encryption PSI approach proposed by Freedman *et al.* [9] has a communication complexity of $O(N)$ and a computational complexity of $O(N \ln\ln N)$ for each party and can be extended to PSIC easily. However, it works only for two parties and a multi-party solution based on their scheme is not obvious [10]. Hohenberger *et al.* [17] presented a scheme similar to the one in [9] with the same complexities.

Kissner and Song [10], [18] proposed a PSIC protocol for multiple parties based on Paillier's additive homomorphic cryptosystem. The protocol has $O\left(k^2 N\right)$ communication complexity and $O\left(k^2 N^2\right)$ computational complexity. Computationally it is less efficient than [8], [11], [14] as discussed in [19]. Also, the communication overhead is potentially twice as high compared to [8], [11], [14], since Paillier uses a 2048 bit modulus while PH/RSA with 1024 bit modules is still viewed as reasonably secure [8], [11], [14]. Sang and Shen [12] proposed a protocol that improves [10], [18] by the security in the Universal Composability (UC) security model.

Li and Wu [20] proposed an information-theoretically secure PSI protocol based on secret sharing, which can be extended to PSIC. The protocol is secure with $t < k/3$ malicious adversaries and the communication complexity is $O\left(k^4 N^2\right)$. Burkhart *et al.* [13] developed a system for secure multi-party computation based on secret sharing and homomorphic encryption that can be used for PSIC, but only with more than two parties. Their approach is resistant to $t < k/2$ colluding parties because it is based on Shamir's secret sharing [21]. However, as with [20] the communication complexity scales with N^2 [21].

Blundo *et al.* [19] proposed to combine sampling and PSIC to securely and efficiently estimate the Jaccard index, a measure of similarity of two datasets ($J = |A \cap B| / |A \cup B|$).[1] They proposed to use MinHash sampling [15] to sample the datasets and then use the technique from [8], [14] for PSIC of the sampled datasets (estimating the Jaccard index from the encrypted samples is straight-forward). However, their approach still does not scale well for large datasets, since MinHash sampling requires that each dataset is hashed completely h times and one needs $h \geq 1000$ to reduce the estimation error to a few percent [19].[2] Also, with MinHash the number of samples is independent of the dataset size, which leads to larger relative estimation errors for larger datasets (which is not desirable for CR).

Our solution is based on the commutative encryption approach proposed by [8], [11], since it is the most efficient solution, it is computationally-secure (given appropriate key and modulus lengths) and it works for two or more parties. However, the technique in [8], [11] is still impractical with large datasets due to the encryption overhead. For example, assuming a modulus of 1024 bits and assuming the input items have a size of four bytes (e.g. IPv4 addresses), an encrypted list of 1 billion items has a size of 128 GB. Also the computational overhead is high, since PH/RSA is much slower than symmetric encryption schemes.

We propose to use deterministic hash-based sampling to sample a subset of items consistent across all parties, perform PSIC on the sampled subsets, and then estimate the intersection cardinality of the original datasets from the intersection cardinality of the samples. In contrast to [19] our sampling approach only requires to hash each input dataset *once*. Our approach significantly improves the scalability and we will show that even with lower sample rates the estimation error is tolerable.

We also address the issue of probing attacks. These are attacks where one party creates a dataset with mostly invalid items, to test whether one (or a few) valid items are in other parties' datasets. An ad-hoc solution to this problem was presented in [11]: if the intersection cardinality between some datasets is small, parties treat this as probing attack. However, this approach cannot distinguish between probe attacks and legitimate datasets with small overlap. Our solution allows parties to detect and prevent probing attacks, if all parties can agree on the set of valid items and this set is not prohibitively large.

We analyse the impact of the proposed sampled PSIC technique on the accuracy of CR estimates. To the best of our knowledge no prior work has done that.

III. Algorithm Overview

First, we provide an overview about our PSIC algorithm, which is explained in more detail in Section IV. Then we discuss the two key techniques the algorithm is based on: commutative encryption and hash-based sampling. Finally, we discuss our proposed approach to prevent probing attacks.

A. Basic algorithm

Let k be the number of parties. Each party i has a dataset D_i with $N_i = |D_i|$ distinct items and a private encryption key K_i. Our algorithm is based on the approach proposed in [8], [11], which is based on commutative encryption (see Section III-B). With commutative encryption different datasets can be encrypted by all parties (item by item), and identical items in the original datasets will always result in identical encrypted items in the datasets encrypted by all parties (*fully-encrypted* datasets), regardless of the order in which the parties encrypted the datasets.

The algorithm works as follows. Each party encrypts the items of their own dataset using their own private key, randomly permutes the encrypted items, and then passes the encrypted and permuted dataset to the next party – a party that has not encrypted and permuted this dataset before. The next party encrypts and randomly permutes the received dataset with their own private key, passes it to the next party that has not had this dataset and so on, until all datasets are fully-encrypted. Finally, the parties share the fully-encrypted datasets among each other. Since the encryption is commutative, the set intersection cardinality of the ciphertexts is identical to the set intersection cardinality of the plaintexts.

The scheme is computationally secure, since no party can decrypt any of the other parties' datasets without knowing the other parties' encryption keys, and because of the random permutations no party knows which ciphertexts map to which plaintexts.

Since the scheme encrypts each item of a dataset separately, the space overhead is very large if the plaintext items are small. Also, the computational overhead is high given the exponentiation-based commutative encryption function (see Section III-B). For large datasets the above scheme is impractical. To make the scheme practical

[1] Based on J the intersection cardinality is: $|A \cap B| = J / (J+1)(|A| + |B|)$.

[2] There is a MinHash variant where datasets are hashed only once and the h elements with the smallest hash values are selected [15]. However, for this variant it is not clear how to compute the unbiased estimator of J [15] while using the PSIC technique from [8], [14].

we propose that initially each party generates a sampled dataset of much smaller size. Hash-based sampling can be used to sample entries consistently across all datasets (see Section III-C). The actual intersection cardinality can be estimated based on the intersection cardinality of the sample datasets.

The above scheme is not resistant to probing attacks. These are attacks where one party generates datasets with mostly invalid items to test whether very few valid items are in another party's dataset. Since no party can decrypt the fully-encrypted datasets, it is impossible to check whether the original data were valid items. We propose a novel approach to defend against probing attacks that works if parties can agree on the valid set items and the set of valid items is not prohibitively large (see Section III-E).

B. Commutative encryption

Let E be a secure and commutative encryption function and K_i be the secret encryption key of party i. Let $E_{K_i}(m)$ denote the encrypted version of plaintext m with key K_i. Since E is commutative, it holds $E_{K_i}\left(E_{K_j}(m)\right) = E_{K_j}\left(E_{K_i}(m)\right)$. Since E is secure, it is resistant to plaintext attacks and collisions. Given m and $E_{K_i}(m)$ it is computationally infeasible to derive K_i, for all m and K_i. Given any two plaintexts m_1 and m_2 it is computationally infeasible to find keys K_i and K_j such that $E_{K_i}(m_1) = E_{K_j}(m_2)$.

Two similar commutative encryption schemes are Pohlig-Hellman (PH) [22] and commutative RSA [23]. The encryption function is:

$$E_{K_i}(m) = m^{K_i} \pmod{p} \ ,$$

where p is large safe prime number[3] shared by all parties and where $\gcd(K_i, p-1) = 1$. It is easy to see that this function is commutative since

$$\begin{aligned} E_{K_i}\left(E_{K_j}(m)\right) &= m^{K_i K_j} \pmod{p} \\ &= m^{K_j K_i} \pmod{p} = E_{K_j}\left(E_{K_i}(m)\right) \ . \end{aligned}$$

From [22], [23] we know that this function is computationally-secure if the key size and modulus size are appropriately chosen. Requirements from NIST [24] state that the key size should be at least 224 bits and the modulus size should be at least 2048 bits to be secure until the year 2030, but many people still consider a key size of 160 bits and a modulus size of 1024 bits (NIST 2010 [24]) as relatively secure (e.g. [8], [11], [14]).

C. Hash-based sampling

Hash-based sampling was used by Broder *et al.* [15] for document comparison and Duffield *et al.* [25] for analysing IP packet trajectories in the Internet. The key idea is to randomly sample the same elements in different lists or the same IP packets at different routers.

Let H be good hash function, i.e. a hash function that generates different output even for very similar input and maps the inputs as uniformly as possible over its output range. We do not require cryptographic properties. Let the hash function be salted so that we can obtain different independent samples for the same input. This can be achieved by adding the salt s to the hash input m before applying the hash function. We can then sample the same elements from different lists by selecting only the elements with

$$H(m + s) \pmod{R} < r \ ,$$

where r/R is the sampling rate ($r, R \in \mathbb{N}$). The sampling rate can take pretty much any value, since r/R can be any rational number in $(0, 1]$. Before using the PSIC protocol, each party samples their dataset. All parties need to agree on H, s, r and R. The salt s could be computed in a shared fashion, e.g. each party contributes some bits, to prevent a single party from controlling which elements are sampled.

D. Sample size confidence intervals

We now derive confidence intervals for the sample sizes used in the detailed algorithm description in Section IV. We model the sampling as follows. Each item in a set of size N is a Bernoulli experiment with success probability equal to the sampling rate p. So the number of items n in the sampled set follows a Binomial distribution with a mean of $\bar{n} = Np$ and a variance $\sigma_{\bar{n}}^2 = Np(1-p)$. Since in our case N is typically very large ($\geq 10^6$) we can approximate the Binomial distribution with a Normal distribution with same mean and variance without using the continuity correction $\frac{0.5}{N}$ [26].

We define a single-sided lower confidence interval (CI)

$$\Pr(X \geq \bar{n} - z_\alpha \sigma_{\bar{n}}) = 1 - \alpha \ , \qquad (1)$$

which states that with a probability of $1 - \alpha$ the size of the sampled dataset is above the lower bound $\bar{n} - z_\alpha \sigma_{\bar{n}}$. Here z_α is the z-value corresponding to α.[4] Similarly, we define a double-sided CI

[3] A safe prime is a prime of the form $2p + 1$, where p is a prime.

[4] A z-value indicates how many standard deviations an observation is away from the mean of a Normal distribution.

$$\Pr\left(\bar{n} - z_{\alpha/2}\sigma_{\bar{n}} \le X \le \bar{n} + z_{\alpha/2}\sigma_{\bar{n}}\right) = 1 - \alpha \;, \qquad (2)$$

which states that with a probability of $1-\alpha$ the size of the sampled dataset is between the lower bound $\bar{n} - z_{\alpha/2}\sigma_{\bar{n}}$ and upper bound $\bar{n}+z_{\alpha/2}\sigma_{\bar{n}}$. Since the CI is two-sided we must use $z_{\alpha/2}$ (not z_α) for both lower and upper bound.

E. Probing attack detection

Besides allowing an attacker to learn whether a few probed IP addresses are in another party's dataset, probing may also help an attacker to obtain encryption keys. PH/RSA is resistant to plaintext attacks, so even if an attacker has both the plaintext and the ciphertext, the attacker cannot recover the encryption key without solving the discrete logarithm problem (DLP).[5] DLP is computationally hard for large prime fields, however recently researchers have solved the DLP with safe primes of over 500 bits [27]. Without successful probing, for a given ciphertext an attacker never knows the corresponding plaintext, making a DLP attack much harder even for smaller modulus sizes.

We propose a mechanism that allows to detect and prevent probing attacks. Our technique can be applied to all situations where the set of permitted items (valid items) is known and not prohibitively large. Our probing detection technique is optional and can be omitted if not needed.

Prior to encrypting the actual datasets, all parties agree on the set of valid items (valid set). For example, in the context of IPv4 addresses the valid set could be the set of actually advertised and routed IPv4 addresses (based on data from [28]). Then the parties essentially perform PSIC as outlined in Section III-A with the valid sets. All parties encrypt every party's valid set. Note that the parties must use the *same* encryption key used later to encrypt the data. At the end each party obtains a valid set encrypted by all parties, which can be used later to check if a fully encrypted dataset contains mostly valid items.

If the set intersection cardinality between an encrypted dataset and an encrypted valid set is under some threshold (as discussed in Section IV-B3), a probing attack is detected. The detecting party informs all parties of the attack and none of the parties sends a fully-encrypted dataset to the presumed attacker. This prevents the attack since the attacker cannot compute any intersection cardinalities. The honest but curious adversary model excludes a collusion of the detecting party with the attacker in the form of not complying with the protocol and not reporting the attack. Even if collusion was possible, the attack could still be detected (but not prevented).

The valid set may be very large, but it can be hash-sampled with a much smaller sample rate than the dataset, making the approach feasible even with larger valid sets. If sampling is used each party samples their valid set using a secret salt. Since the salt is secret and the valid sets are encrypted and permuted, no party knows which items are in another party's valid set. The dataset of an honest party will produce an overlap consistent with the valid set sampling rate, but the dataset of a probing party will produce a near zero overlap (as discussed in Section IV-B3).

We also propose that all parties agree on a minimum dataset size N_{min}. Any small dataset, even if completely valid, is effectively a probing attack. If a party encounters a dataset smaller than N_{min}, it will not encrypt and forward this dataset. Due to the time it takes to run the algorithm, a minimum dataset size also implicitly limits the probing rate. Furthermore, in some applications it may be feasible to limit the rate of set intersection cardinality computations, which would explicitly limit the probing rate.

IV. Detailed Algorithm Description

We now describe our adversary model and the proposed algorithm. We first describe the two-party case and then extend the algorithm to three or more parties. Then we discuss the security and complexity. Finally, we discuss how to choose the sample rate.

A. Adversary model

We assume that all parties participating in the PSIC protocol are potential honest-but-curious adversaries; they run the protocol correctly, but they try to learn as much information as possible. For example, they perform the encryption and permutation correctly, but they may select certain input data to attempt a probing attack. We assume that one or more honest-but-curious adversaries may collude in order to obtain more information. We assume that man-in-the-middle attacks by third parties are prevented by using secure communication, such as Secure Sockets Layer (SSL) [29] or Transport Layer Security (TLS) [30].

[5] There are more efficient attacks against RSA. However, they assume that the attacker can get the victim to decrypt a chosen ciphertext, and in our approach decryption is not used.

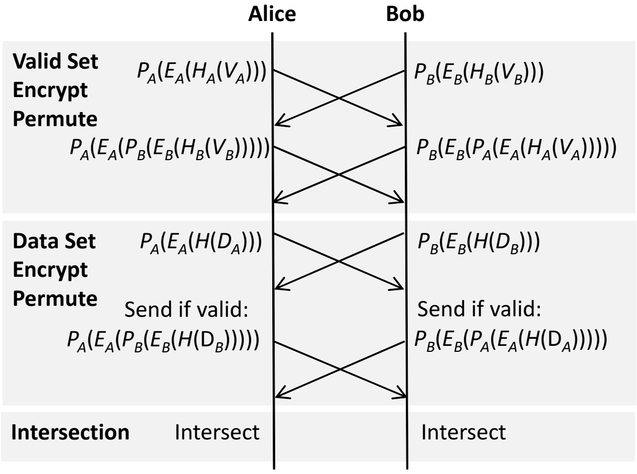

Figure 1. Protocol for two parties. Here P, E, H denote the permutation, encryption and hash-sampling of the valid set V or dataset D created by (A)lice or (B)ob. Alice and Bob use their private permutation and encryption functions. Alice and Bob use different private hash functions for valid sets, but they use the same hash function for datasets.

B. Two parties

Let the two parties be (A)lice and (B)ob with datasets D_A and D_B and valid sets V_A and V_B (for simplicity we assume $V_A = V_B$). The algorithm has four main steps:

1) Configuration (agree on parameters)
2) Valid set encryption and permutation (optional)
3) Dataset encryption and permutation (main step)
4) Intersection cardinality computation

We now discuss each step. Let E_i, H_i and P_i be the encryption, hash and permutation functions used by party i. Since for datasets all parties use the same sampling salt we drop the index and just use H in this case. Figure 1 shows an overview of the whole protocol (except the configuration).

1) Configuration: In the configuration step A and B negotiate the configuration: the modulus used for encryption, the sampling rate for the data p_D, the hash function used for sampling and the hash salt, the minimum dataset size N_{min}, and whether probing detection is used or not. Then, A and B each generate their own secret encryption key K_i independently. Since decryption is not required, A and B do not generate decryption keys.

If A and B want to use probe detection they agree on the valid set V and and the sampling rate for the valid set p_V (typically $p_V < p_D$). In this case, A and B also each generate their secret valid set hash salt independently.

2) Valid set encryption and permutation: This step is only needed if A and B use probing detection. In this step A and B generate the sampled valid set, encrypt and permute it, and send their encrypted and permuted valid sets to each other. A sends $P_A(E_A(H_A(V_A)))$ to B, and B sends $P_B(E_B(H_B(V_B)))$ to A.

A and B then encrypt and permute each others valid set and return the double-encrypted valid sets to each other. A sends $P_A(E_A(P_B(E_B(H_B(V_B)))))$ to B, and B sends $P_B(E_B(P_A(E_A(H_A(V_A)))))$ to A. A and B may compute the set intersection cardinality of the valid sets to check if their view of the valid space is consistent. Without sampling the overlap should be 100%. With sampling the overlap should be close to $100 p_V p_V\%$.

3) Dataset encryption and permutation: In this step A and B sample, encrypt and permute their own datasets, and send the encrypted and permuted sets to each other. A sends $P_A(E_A(H(D_A)))$ to B, and B sends $P_B(E_B(H(D_B)))$ to A.

Then, A and B check if each others dataset is at least of size N_{min}. If the dataset was sampled A and B estimate the unsampled size $N \approx \hat{N} = n/p_D$. For example, N_A is estimated based on the sampled size n_A to be $\hat{N}_A = n_A/p_D$. Given $\hat{N}$ and p_D, A and B compute the one-sided lower CI from Equation 1 and check if that lower bound is at least N_{min}/p_D. If yes, the checking party encrypts and permutes the received dataset. Otherwise, the checking party aborts and does not return the double-encrypted set to the other party.

Without probing detection the parties will then return the double-encrypted datasets to each other: A sends $P_A(E_A(P_B(E_B(H(D_B)))))$ to B, and B sends $P_B(E_B(P_A(E_A(H(D_A)))))$ to A. With probing detection each party first computes the set intersection cardinality between their double-encrypted valid set and the other party's double-encrypted dataset. If the valid set is not sampled ($p_V = 1$), all of the dataset items should be in the valid set. If that is the case, the checking party returns the double-encrypted dataset to the other party. Otherwise, the checking party aborts and does not return the double-encrypted set.

If the valid set is sampled, then each data item is only present in the valid set with probability p_V. As above A and B can estimate the size of each others dataset by $\hat{N} = n/p_V$ and compute the one-sided lower CI with Equation 1. For an honest party that has only valid items in the dataset, the overlap should be over the lower bound with probability $1 - \alpha$. We can choose a low α to minimise the false positive probability (the probability that an honest party is incorrectly identified as doing a probing attack). We assume that for a probing attack the number of valid items in the prober's dataset has to be below some limit $N_{probe} \ll N$ (typically we have large datasets but the number of probed items is small). Since an attacker does not know the other party's valid set sampling salt, an attacker's number of sampled valid items is on average $N_{probe}p_V$. Since $N_{probe}p_V \ll Np_V$ with high probability even for low α, the lower bound of the one-sided CI is higher than $N_{probe}p_V$. Hence, the false negative probability (the probability that a probing party is not identified as such) is also very small.

Only if a dataset is recognised as valid, A and B will return it to the other party. Otherwise A and B will abort the protocol, and the probing party cannot learn anything.

4) Intersection cardinality computation: Since the encryption is commutative any items that are in both sets will have the same ciphertexts in both double-encrypted datasets. Without sampling A and B compute the intersection cardinality by counting the number of equal ciphertexts in both double-encrypted sets.

With sampled datasets A and B compute an estimate of the intersection cardinality. Let $C = |D_A \cap D_B|$ denote the intersection cardinality of the datasets. Since the probability that an element of the intersection $D_A \cap D_B$ is in the sample is p_D, the expected value of the intersection size of the samples $\left|\tilde{D_A} \cap \tilde{D_B}\right|$ is:

$$\mathbb{E}\left(\left|\tilde{D_A} \cap \tilde{D_B}\right|\right) = C \cdot p_D ,$$

and hence $\hat{C}$ is an unbiased estimator for C:

$$\mathbb{E}\left(\hat{C}\right) = \mathbb{E}\left(\left|\tilde{D_A} \cap \tilde{D_B}\right|\right) \frac{1}{p_D} = C$$

If $\hat{C}$ is large we can approximate it as Normal-distributed and construct a two-sided CI around $\mathbb{E}\left(\left|\tilde{D_A} \cap \tilde{D_B}\right|\right)$ according to Equation 2:[6]

$$\Pr\left(\hat{C} \cdot p_D - z_{\alpha/2}\sigma_{\hat{C}} \leq C \cdot p_D \leq \hat{C} \cdot p_D + z_{\alpha/2}\sigma_{\hat{C}}\right) = 1 - \alpha , \quad (3)$$

where $\sigma^2_{\hat{C}} = Cp_D(1 - p_D) \approx \hat{C}p_D(1 - p_D)$. We can transform Equation 3 into a CI for the actual intersection size C:

$$\Pr\left(\hat{C} - z_{\alpha/2}\sigma_{\hat{C}}/p_D \leq C \leq \hat{C} + z_{\alpha/2}\sigma_{\hat{C}}/p_D\right) = 1 - \alpha . \quad (4)$$

C. Multiple parties

Extending the scheme to more than two parties is relatively straight-forward. We propose that the parties form a one-directional ring topology. Figure 2 shows an example for three parties. To counter possible collusion the ring topology could be generated randomly, for example using a secure multi-party computation for random ordering [31]. We assume that each party knows the complete topology, which means it also knows all other parties.

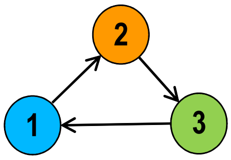

Figure 2. Ring topology for three parties – encrypted and permuted datasets are only passed in one direction

During steps 2 and 3 each party receives valid sets or datasets from their right neighbour, encrypts and permutes the sets and forwards them to their left neighbour. Without probing detection, after all encryption is done in step 3 each party will have their own dataset encrypted by all parties. Each party then sends their own fully-encrypted dataset to all other parties (or a subset of parties interested in the intersection). Then all interested parties can perform the intersection cardinality computation.

With probing detection, each party checks for probing before returning a fully-encrypted dataset to the original owner (each party knows when a dataset is fully-encrypted since the number of parties is known). Each party broadcasts the result of the probing detection to all other parties, i.e. whether it detected a probing attack or not. After the broadcasts are received, all parties send the fully-encrypted datasets to all interested parties not identified as probers. This means probers will never get any fully-encrypted datasets (not even their own) and the probing attack fails. However, all interested honest parties can perform the intersection cardinality computation (extending the approach described in Section IV-B4 to multiple sources is straightforward).

Vaidya et al. [11] proposed an optimisation of the protocol for three or more parties. Instead of a ring the parties form a binary tree, and the intersections are carried out in hierarchical fashion. The technique is a significant performance improvement, especially if any two-party intersections are much smaller then the sizes of the datasets, and also reduces information disclosure, as not all parties can learn the intersection sizes of all combinations of datasets. Vaidya's optimised approach works with our proposed dataset sampling. Unfortunately, it does not work in the case of population estimation with CR, which requires that the interested parties must learn the intersection sizes of *all* combinations of datasets.

[6] Since we do not know C we must use its estimate $\hat{C}$ to get an estimated standard deviation.

D. Complexity

Since every party needs to encrypt and permute the datasets of all parties (and possibly the valid sets of all parties) the total computational complexity of the scheme is $O\left(k^2N\right)$ overall, where where k is the number of parties and N is the size of the datasets. However, for a single party the complexity is linear $O(kN)$. Since every party has to forward every other parties dataset, the communication complexity is also $O\left(k^2N\right)$ overall and $O(kN)$ for each party.

The sampling of the data only requires one hash function computation, one modulo operation and one comparison per dataset item (neglecting the addition with the salt).[7] Fast hash functions only require few multiplication operations (plus some add, shift and logical operations which are much faster than multiplications). For example, the Murmur hash function [32] performs only three multiplications plus three multiplications per 4 bytes of input data (a total of six multiplications in the case of 4-byte input data, such as IPv4 addresses). The encryption requires one modular exponentiation per dataset (or valid set) item. Raising a number to an exponent of K bits requires K–$2K$ multiplications and typically K is a few hundred bits.[8]

E. Security

The two-party scheme is computationally-secure in the presence of semi-honest adversaries [14] and it is easy to see that the security extends to the multi-party case. To protect against third parties, we propose to use properly configured SSL/TLS [29], [30] for all communication.

There are no security issues in step 1. In step 2 only the valid space is exchanged. No party can learn anything about another party's IP addresses. An attacker could mount a probing attack in this phase, and if successful could then perform a known plaintext attack to recover another party's secret encryption key. However, the encryption is resistant to known plaintext attacks, provided a secure key size and modulus are used. Without probing an attacker cannot perform a known plaintext attack due to the random permutation of items performed with each encryption.

In step 3 all parties exchange the encrypted and permuted item lists. Since the encryption is secure, no party can decrypt another party's dataset. Since all datasets are permuted by each party, no party knows the mapping between data items and ciphertexts for any encrypted dataset. Again, plaintext attacks are only possible if probing can be performed. If probing is prevented the only information the parties can learn are the sizes of the intersections of the different datasets and the sizes of the datasets. All encrypted datasets and keys used could be deleted after the set intersection cardinality has been computed. This helps to prevent potential attacks later on, e.g. if at some stage an attacker somehow gains access to keys and encrypted datasets.

The sampling we propose increases the practical security of the approach. Even if an attacker could somehow steal the encryption keys, the attacker would only get access to subsets of sampled items and not whole datasets.

F. Minimum sample rate

We can determine the minimum sample rate needed, so that the relative error of the intersection cardinality estimate is not higher than a target maximum relative error $\Delta\epsilon_{max}$ with a given confidence level $1-\alpha$ if we have an estimate of the intersection cardinality. Based on Equation 4 we define:

$$\Delta\epsilon_{max} = \frac{\hat{C}-C}{C} \geq \frac{-z_{\alpha/2}\cdot\sigma}{Cp_D} = \frac{-z_{\alpha/2}\cdot\sqrt{Cp_D(1-p_D)}}{Cp_D},$$

which we can rewrite as:

$$\Delta\epsilon^2_{max} \geq \frac{z^2_{\alpha/2}\cdot p_D(1-p_D)}{Cp^2_D}.$$

Solving the quadratic equation for p_D yields:

$$p_D \geq \frac{2z^2_{\alpha/2}}{2\left(\Delta\epsilon^2_{max}C + z^2_{\alpha/2}\right)}.$$

Then r and R must be chosen so that $r/R \geq p_D$. In practice we do not know C, but we may have a rough idea of its expected value. If we use a lower bound for C then the computed p_D will be sufficient to keep the relative error below $\Delta\epsilon_{max}$. If multiple different intersection cardinalities shall be computed from more than two datasets, we need to use the minimum of all expected intersection cardinalities to determine the sample rate.

If we cannot predict C, a possible strategy would be to compute the intersection size(s) with a low sample rate to obtain an estimate of C and then if necessary recompute the intersection size(s) with a sample rate adjusted to achieve the desired precision. This is still more efficient than computing the intersection sizes(s) without sampling, assuming the adjusted sample rate is small.

[7] If we restrict the sample rates to $\frac{1}{2^e}$ with $e = [0, 1, \ldots, N]$ we could replace the slower modulo operation with a much faster shift operation.

[8] Usually Montgomery multiplication [33] is used to implement $x^y \bmod M$, which requires K–$2K$ plus 2 Montgomery steps.

V. Prototype Implementation

First we present an overview of our prototype implementation. Then we present some examples demonstrating the use of the prototype.

A. Overview

We implemented a prototype in Python, which we named SeFaSI (Secure Fast Set Intersection) [16]. The encryption is based on the RSA functions of the PyCrypto library [34]. As hash function for sampling we use the Murmur hash [32] Python library. The Murmur hash function is very fast and has a good distribution that passes all the usual tests [32]. Note, that both libraries are based on underlying C/C++ libraries. Using Python has the advantage of enabling fast prototyping and keeping the source code small and readable. The last point is very important, as it allows all parties to easily verify the security of the implementation, i.e. the encryption and permutation code.[9] Our implementation consists of several tools that implement the basic functions, such as sampling, encryption, set intersection cardinality computation, and an "umbrella" tool that implements the PSIC protocol steps based on the basic tools.

The disadvantage of a Python implementation is slower execution speed compared to compiled languages, such as C/C++. Even if a library is coded in C/C++, such as the used encryption/hash libraries, the Python to C/C++ call conversion adds significant overhead in case a function executes relatively fast and is called many times (which is the case for the sampling). We also implemented a fast version of the sampling, encryption, and set intersection cardinality computation tools in C. The C tools use the modular exponentiation function of libopenssl and the C version of the Murmur hash.

The random permutation is implemented based on GNU sort, which is used after the encryption. A lexicographical sort of the ciphertexts is equivalent to a random permutation of the plaintexts, since the ciphertexts are essentially random bit strings. Performance-wise this approach is not ideal since sorting has complexity $O(Nlog(N))$, while shuffling (e.g. Fisher–Yates) has complexity $O(N)$. However, the sorting with GNU sort is still almost two magnitudes faster than the encryption, GNU sort is readily available and it performs external sorting (it uses constant memory regardless of the size of the input data).

Also, having sorted ciphertexts has the advantage that the computation of the intersection size of two or more encrypted and permuted datasets can be done with $O(N)$ complexity (assuming $N \gg k$). Our intersection algorithm moves k pointers through the k datasets. In each step the algorithm checks whether the current k ciphertext values of items are the same. If yes, then the size of the intersection is increased by one. Then, the pointers for the datasets with the currently smallest ciphertext values are advanced and so on.

With the "umbrella" tool the parties have to perform the process manually. For example, each time a party receives an encrypted dataset, it must run the tool to encrypt and permute the dataset, and and then it must pass on the dataset to the next party. The prototype also includes a tool that performs the set intersection cardinality computation with multiple geographically-distributed parties almost automatically (but currently only without probing detection).

The results we will present in Section VI show that the performance of our prototype is sufficient for practical use, even if the datasets contain up to 1–2 billion entries. However, further improvements are future work, such as a scalable shuffling implementation or multi-threading support that would allow to parallelise the sampling or encryption on CPUs with multiple cores.

B. Example use

SeFaSI consists of a number of separate tools. The README file in the distribution [16] provides many examples on how to use the different tools. Here we illustrate the use of SeFaSI in the scenario where multiple geographically-distributed parties want to compute the intersection cardinality without using probing detection.

SeFaSI uses two configuration files. The *public* configuration file specifies the sampling parameters, the probing detection parameters and public encryption parameters (the modulus). The *private* configuration file contains the private key and the private sampling salt for the sampling of the valid set.

1) Step 1: SeFaSI configuration: All parties need to agree on the public configuration. One approach is to use SeFaSI to generate a default public configuration file with the command:

```
sefasi_main.py -a config -n test
```

This command will generate the two files test.pub.cfg (public configuration) and test.priv.cfg (private configuration). The public configuration file can be modified based on the requirements and distributed to all parties. Then all parties need to generate a private configuration file with the following command:

[9] Assuming there is trust in the used libraries.

```
sefasi_main.py -a config -c test.pub.cfg -n
test
```

2) Step 2: Host setup: All parties need to agree on the list of participating hosts (peers) and their host names or IP addresses. Each party must create two accounts on their host:

- A “main” account that is used to run SeFaSI and has access to the public *and* private configuration.
- A “sefasi” account that other parties can access to upload data files (acting as “inbox”) and has no access to the private configuration.

Each party must generate a key pair for the "main" account for use with SSH. For example, by running the following command an RSA key pair is created:

```
ssh-keygen -t rsa
```

Each party must add all parties’ public RSA keys to the .ssh/authorized_keys file of the “sefasi” user (including their own public RSA key). Each party can access each other party’s “sefasi” user, hence the “sefasi” user *must not* have access to the SeFaSI private configuration or any other private data, such as the unencrypted data files.

3) Step 3: Initial encryption and permutation: Each party must perform the initial sampling and encryption of their dataset. For example, if the dataset is `data.txt` the following command can be used:

```
sefasi_main.py -f -a init_enc -c
test.pub.cfg -p test.priv.cfg data.txt
```

This command will generate the files data.txt.encperm (encrypted and permuted dataset) and data.txt.size (contains the true size of the dataset as number).

4) Step 4: Run the PSIC protocol: Each party then can start the PSIC protocol. In the following command we assume the list of participating hosts is "192.168.1.1 192.168.1.2 192.168.1.3" (host names can be specified instead of IP addresses), the IP address of the local host is 192.168.1.2, and the PREFIX directory /home/main/sefasi is used to store temporary files and the results in different sub directories:

```
sefasi_peer.sh test.pub.cfg test.priv.cfg
"192.168.1.1 192.168.1.2 192.168.1.3"
192.168.1.2 /home/main/sefasi
```

`sefasi_peer.sh` will check the reachability of all peers with SSH, create working directories below PREFIX and then wait for the user to put the .encperm and .size files generated in step 3 into the appropriate directory (by default PREFIX/own). The files can also be copied into this directory before starting the process.

Table I

Speed of item sampling depending on sample rate with 4-byte items

Sample rate [%]	Items/second
1	5,450,000
5	5,300,000
10	5,200,000
50	4,350,000

`sefasi_peer.sh` will pass the encrypted and permuted local dataset to the left neighbour.[10] Once an encrypted dataset is received from the right neighbour, `sefasi_peer.sh` will encrypt and permute the dataset and pass it to the left neighbour. The process will continue until all datasets have been encrypted by all peers. Fully-encrypted datasets will then be sent to all peers. After all fully-encrypted datasets have been received, `sefasi_peer.sh` will compute the intersection cardinalities between all datasets.

VI. Performance

This section describes the performance evaluation of our prototype system. Our performance measurements were done on a PC with Intel i7 2.8 GHz CPU and 24 GB RAM running FreeBSD 9.0 and Python 2.7.3. Note that in all experiments we used a single CPU core only.[11]

A. Sampling speed

Table I shows the number of 4-byte item values (e.g. IPv4 addresses) sampled per second depending on the sample rate (the higher the sample rate is the higher the file write overhead) measured for the C implementation (the average of 10 runs). Even with 50% sample rate our prototype can sample about 4,35 million items per second, so it would take about 4 minutes to sample a dataset with 1 billion 4-byte items. The Python sampling implementation is roughly one magnitude slower, for example at 50% sample rate the Python tool can only sample 420,000 items per second.

B. Encryption and permutation speed

Table II shows the encryption and permutation speed depending on the key size and the modulus size for

[10] The list of hosts is viewed as ring. Datasets other than the fully-encrypted datasets are only passed to the left neighbour, which is the host in the list before the local host. For example, here the left neighbour of 192.168.1.1 is 192.168.1.3, the left neighbour of 192.168.1.2 is 192.168.1.1 and so on.

[11] We attached each tested process to a particular CPU core with the tool `cpuset`.

Table II
Speed of encryption and permutation depending on key size and modulus size with 4-byte items and encrypted 4-byte items

Key/Modulus size [bits]	Items/second	Encr. items/second
128/128	29,000	28,000
160/256	19,000	18,500
160/512	10,000	10,000
160/1024[a]	3,600	3,500
224/2048[b]	1,100	1,000

[a] NIST 2010 (Legacy) [24]
[b] NIST secure until 2030 [24]

Table III
Speed of set intersection cardinality computation depending on key size and modulus size with encrypted items

Datasets	Overlap [%]	Items/second
2	100	2,180,000
3	100	1,500,000
4	100	1,150,000
5	100	950,000
2	0	1,650,000
3	0	880,000
4	0	540,000
5	0	360,000

both 4-byte input items (e.g. IPv4 addresses) and already encrypted items (encrypted with the same key and modulus size) for the C implementation (the average of 10 runs). As expected, the rate decreases with increasing key and modulus lengths. While it takes longer to encrypt larger input (e.g. ciphertexts of already encrypted IPv4 addresses), the difference is relatively small. Note that the rates are dominated by the encryption, which takes 97–98% of the time. Only 2–3% of the time is used by the permutation (GNU sort). The Python implementation actually has a very similar performance. The encryption function is very slow, so the Python call overhead is negligible.

With an insecure 64 bit key and 64 bit modulus it would take 9–10 hours to encrypt end permute 1 billion items. With 160 bit key and 1024 bit modulus (NIST 2010 Legacy [24]) it would take over 3 days to encrypt and permute 1 billion items; however, with a sample rate of 10% the time reduces to only 8 hours (including the sampling).

C. Set intersection cardinality speed

Table III shows the set intersection cardinality speed depending on the number of datasets and the overlap (the average of 10 runs). All datasets have roughly the same size. An overlap of 100% represents the best case (highest performance) and an overlap of 0% represents the worst case (lowest performance). Computing the set intersection cardinality of five sets with roughly 1 billion items each would take between 18 minutes and 50 minutes depending on the overlap. However, with a sample rate of 10% the time would reduce to between 2 and 5 minutes. The C implementation is roughly 20 times faster than the Python implementation, for example the Python implementation only achieves 110,000 items per second for two datasets and 100% overlap.

D. Dataset size exchanged

If the items are 4 bytes long, such as for IPv4 addresses, then each unencrypted set is of size $4N_i$ bytes (uncompressed). With our sampled PSIC approach the dataset size depends on the encryption modulus and the sample rate. Table IV illustrates the sizes for different example dataset sizes, encryption moduli and sample rates.

E. Cardinality estimate sampling error

We now analyse the error caused by sampling and investigate whether the empirical error is consistent with the theoretical predictions. Figure 3 shows the relative errors of the estimates depending on different true cardinality and different sample rates as boxplots (100 runs for each setting). The relative error decreases with increasing sample rate and increasing true cardinality. In Section VI-F we analyse in more detail how the error affects actual CR population estimates. Here we only note that with the datasets used for CR in [6] the smallest cardinality of all combinations of datasets is larger than 2,000 with all sources or larger than 55,000 if the smallest source is removed (or pooled with a larger source).

We also analysed the percentage of experiments where the true value lies within the 95% confidence interval (CI) estimated with Equation 4. If our derived CI is correct, then for a large number of runs the measured percentages should always approach 95%. For the settings in Figure 3 in almost all cases the measurements are within ±3% of the correct 95% value, and some noise is expected since we only carried out 100 runs per setting. We conclude that our derived CI appears to be correct.

F. CR estimate sampling errors

We also evaluated the impact of the sampling on the pooled Lincoln-Petersen (L-P) approach and log-linear

Table IV
Sizes of encrypted and sampled datasets for example values of number of 4-byte items, encryption moduli and sample rates

Number of 4-byte items [M]	Encryption modulus [bits]	Sample rate [%]	Size [GB]
100	128	100	1.6
100	512	100	6.4
100	1024	100	12.8
1000	128	100	16
1000	512	100	64
1000	1024	100	128
100	128	10	0.16
100	512	10	0.64
100	1024	10	1.28
1000	128	10	1.6
1000	512	10	6.4
1000	1024	10	12.8

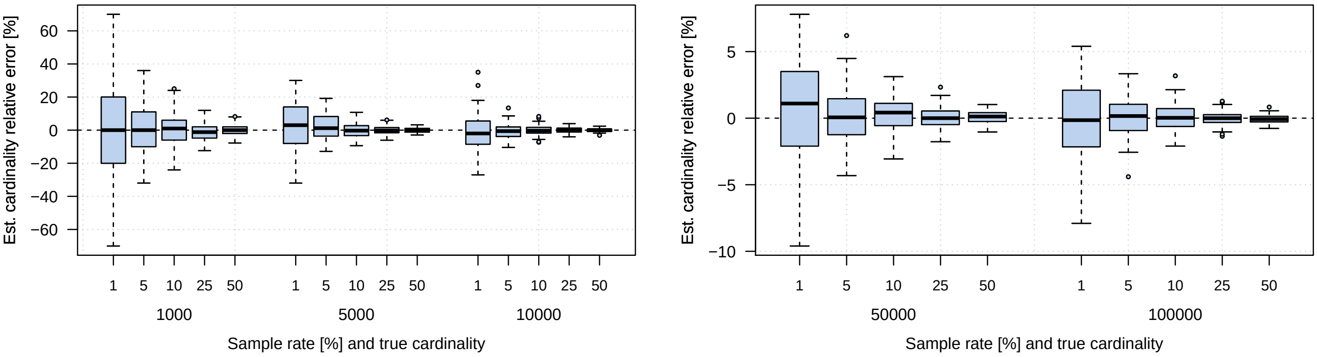

Figure 3. Relative error of cardinality estimates depending on true cardinality size and sample rate

models (LLMs) used to estimate the actively used IPv4 space based on several IPv4 address data sources in [6]. We repeatedly sampled all data sources from [6], each time with a different sample salt, and then computed the CR estimates from the sampled data sources. Note that for LLMs we did not identify the "best" model from the unsampled data and then consistently used this best model. Instead the best model is selected each time from the sampled data. This results in larger errors but reflects the more realistic case where we never have the unsampled data and hence do not know the best model based on the unsampled data.

Figure 4 shows the relative errors for lower and upper bounds of L-P and LLM estimates depending on different sample rates, treating the estimates for the unsampled data sources as true values.[12] For each sample rate we carried out 100 runs (with different sample salts). The results show that the relative errors are very small for pooled L-P even with sample rates as low as 0.1%. The relative errors for LLMs are much larger, but still do not exceed 5%. Also, with LLMs for sample rates of 1% or higher the error is usually no more than 1–2%.

For pooled L-P the datasets are always aggregated into two sets and all intersection cardinalities between any two pooled datasets are relatively large, so the relative errors caused by sampling are small. For LLMs the intersection cardinality for any combination of datasets can be used potentially depending on which model is selected [6]. If some cardinalities used by the selected model are small their relative error due to the sampling is large, and this

[12] The lower and upper bounds for pooled L-P are the minimum and maximum estimates over all combinations of pooling all data sources into two sets [6]. The lower and upper bounds for LLMs are based on the profile likelihood "confidence interval" [6].

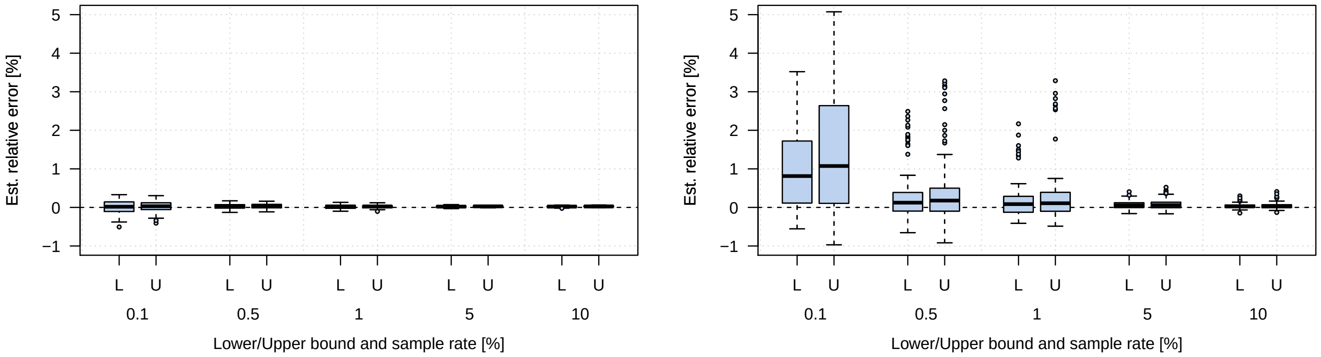

Figure 4. Relative error of (L)ower and (U)pper bound of CR estimates depending on the sample rate for pooled L-P (left) and LLMs (right)

can significantly increase the error of the CR estimate. As mentioned before, with all the data sources from [6], the smallest cardinality is larger than 2,000.

We could restrict the LLM to not use cardinalities that are very small and unreliable. However, the larger errors for LLMs are mainly caused by model inconsistencies, since the model is selected independently for each set of sampled datasets. For sample rates of 5% and 10% the different selected models are quite consistent; there are only a few small differences in the selected parameters that represent higher level data source interactions (e.g. interactions between four or more data sources). For a sample rate of 1% models for different samples start to diverge. For a sample rate of 0.1% different models sometimes even differ in a few two-source interactions. However, even with a sample rate of 0.1% the relative error is no larger than 5%.

Compared to the "true" LLM based on the complete/unsampled data, the LLMs based on the sampled data are biased towards a larger number of model parameters. Since we have mainly (apparently) positively correlated data sources in [6], increasing the number of model parameters usually results in higher population estimates.[13] Hence the error is biased towards overestimating the "true" population, which is clearly visible in Figure 4(right).

VII. Conclusions and Future Work

A private set intersection cardinality (PSIC) protocol allows using capture-recapture (CR) techniques to estimate population sizes with datasets from multiple parties while ensuring the privacy of the data of each party. Previously proposed PSIC protocols are not well suited for the use with CR, some only work for two parties, others work only for more than two parties, and all of them do not scale well for large datasets.

We developed a new PSIC technique that is a tradeoff between performance and precision of the estimated set intersection size. It significantly improves the scalability of existing approaches at the cost of a small sampling error. Based on an implemented prototype we demonstrated that our algorithm scales well with datasets of up to at least 1–2 billion items with acceptable sampling errors. We also showed that the impact of the sampling on the CR estimates is tolerable. Even with sample rates as low as 0.1% the relative error in the CR estimates is under 5%, and with sample rates of 1% or higher the relative error in the CR estimates is not over 1–2%. Our PSIC technique also includes a novel approach to reliably prevent probing attacks – attacks where one party creates a dataset with mostly invalid items, to test whether one (or a few) valid items are in other parties' datasets.

Acknowledgements

This research was supported under Australian Research Council's Linkage Projects funding scheme (project LP110100240) in conjunction with APNIC Pty Ltd and by Australian Research Council grant FT0991594.

References

[1] C. G. J. Petersen, "The Yearly Immigration of Young Plaice into the Limfjord from the German Sea," *Rept. Danish Biol. Sta.*, vol. 6, pp. 1–77, 1895.

[2] F. C. Lincoln, "Calculating Waterfowl Abundance on the Basis of Banding Returns," *U.S. Dept . Agric. Circ.*, vol. 118, pp. 1–4, 1930.

[3] E. B. Hook, R. R. Regal, "Capture-Recapture Methods in Epidemiology: Methods and Limitations," *Epidemiol. Rev.*, vol. 17, no. 2, pp. 243–264, 1995.

[13] Positively correlated data sources lead to population underestimates with L-P. LLMs correct for this with positive model parameters (coefficients) that increase the estimate. The more positive parameters are present in the selected "best" model, the higher is the estimate.

[4] A. Chao, P. K. Tsay, S. H. Lin, W. Y. Shau, D. Y. Chao, "The Applications of Capture-Recapture Models to Epidemiological Data," *Statistics in Medicine*, vol. 20, pp. 3123–3157, October 2001.

[5] M. Roughan, J. Tuke, O. Maennel, "Bigfoot, Sasquatch, the Yeti and other missing links: what we don't know about the AS graph," in *ACM Internet Measurement Conference (IMC)*, pp. 325–330, October 2008.

[6] S. Zander, L. L. H. Andrew, G. Armitage, G. Huston, "Estimating IPv4 Address Space Usage with Capture-Recapture," in *(accepted at) 7th IEEE Workshop on Network Measurements (WNM) in conjunction with IEEE LCN*, October 2013.

[7] G. Huston, "IPv4 Address Report." http://www.potaroo.net/tools/ipv4/index.html.

[8] R. Agrawal, A. Evfimievski, R. Srikant, "Information Sharing Across Private Databases," in *ACM SIGMOD International Conference on Management of Data*, pp. 86–97, 2003.

[9] M. Freedman, K. Nissim, B. Pinkas, "Efficient private matching and set intersection," in *Eurocrypt*, 2004.

[10] L. Kissner, D. Song, "Privacy-preserving set operations," in *Advances in Cryptology (CRYPTO)*, pp. 241–257, 2005.

[11] J. Vaidya, C. Clifton, "Secure Set Intersection Cardinality with Application to Association Rule Mining," *J. Comput. Secur.*, vol. 13, pp. 593–622, July 2005.

[12] Y. Sang, H. Shen, "Efficient and Secure Protocols for PrivacyPreserving Set Operations," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, pp. 1–35, November 2009.

[13] M. Burkhart, M. Strasser, D. Many, X. Dimitropoulos, "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics," in *19th USENIX Security Symposium*, August 2010.

[14] E. De Cristofaro, P. Gasti, G. Tsudik, "Fast and Private Computation of Cardinality of Set Intersection and Union," in *11th International Conference on Cryptology and Network Security (CANS)*, 2012.

[15] A. Z. Broder, "On the Resemblance and Containment of Documents," in *Compression and Complexity of Sequences 1997*, pp. 21–29, June 1997.

[16] S. Zander, "Secure Fast Set Intersection (SeFaSI) Implementation," 2013. http://caia.swin.edu.au/sting/sefasi.

[17] S. Hohenberger, S. Weis, "Honest-verifier private disjointness testing without random oracles," in *Workshop on Privacy Enhacing Technologies (PET)*, 2006.

[18] L. Kissner, D. Song, "Privacy-Preserving Set Operations." Technical Report CMU-CS-05-113, Carnegie Mellon University, June 2006.

[19] C. Blundo, E. Cristofaro, P Gasti, "EsPRESSo: Efficient Privacy-Preserving Evaluation of Sample Set Similarity," in *Data Privacy Management and Autonomous Spontaneous Security*, pp. 89–103, 2013.

[20] R. Li, C. Wu, "An Unconditionally Secure Protocol for Multi-Party Set Intersection," in *5th International Conference on Applied Cryptography and Network Security (ACNS)*, pp. 226–236, 2007.

[21] T. B. Pedersen, Y. Saygm and E. Savas, "Secret sharing vs. Encryption-based Techniques For Privacy Preserving Data Mining." Joint UNECE/Eurostat work session on statistical data confidentiality, Manchester, United Kingdom, December 2007.

[22] S. Pohlig, M. Hellman, "An improved algorithm for computing logarithms over and its cryptographic significance," *IEEE Transactions on Information Theory*, vol. 24, pp. 106–110, January 1978.

[23] A. Shamir, R. Rivest, L. Adleman, "Mental Poker." MIT/LCS/TM-125, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1979.

[24] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, "Recommendation for Key Management." Special Publication 800-57 Part 1 Rev. 3, NIST, July 2012.

[25] N. G. Duffield, M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Trans. Netw.*, vol. 9, pp. 280–292, June 2001.

[26] W. Feller, "On the normal approximation to the binomial distribution," *The Annals of Mathematical Statistics*, vol. 16, no. 4, pp. 319–329, 1945.

[27] T. Kleinjung, "Discrete logarithms in GF(p) – 160 digits," February 2007. https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind0702&L=NMBRTHRY&P=R45&D=0&I=-3&T=0.

[28] University of Oregon Route Views Project. http://www.routeviews.org/.

[29] A. Freier, P. Karlton, P. Kocher, "The secure sockets layer (ssl) protocol version 3.0." RFC 6101 (Historic), August 2011. http://www.ietf.org/rfc/rfc6101.txt.

[30] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246 (Proposed Standard), August 2008. http://www.ietf.org/rfc/rfc5246.txt.

[31] L. Sweeney, M. Shamos, "A Multiparty Computation for Randomly Ordering Players and Making Random Selections," Tech. Rep. CMU-ISRI-04-126, Carnegie Mellon University, School of Computer Science, July 2004. http://reports-archive.adm.cs.cmu.edu/anon/isri2004/CMU-ISRI-04-126.pdf.

[32] A. Appleby, "MurmurHash," 2011. https://sites.google.com/site/murmurhash/.

[33] P. Montgomery, "Modular Mupltiplication Without Trial Division," *Math. Computation*, vol. 44, pp. 519–521, 1985.

[34] D. C. Litzenberger, "PyCrypto – The Python Cryptography Toolkit." https://www.dlitz.net/software/pycrypto/.