An Experimental Study of Encrypted Polynomial Arithmetics for Private Set Operations

Myungsun Kim and Benjamin Z. Kim

Abstract: **Recent studies on the performance of private set operations have examined the use of homomorphic public-key encryption and the technique of representing sets as polynomials in a cryptographic model. These polynomial-based solutions require intensive polynomial arithmetics that exhibit quadratic computational complexity. In the effort to develop practical algorithms for private set operations, various well-known techniques such as Karatsuba's algorithm or the fast Fourier transform (FFT) can be used to reduce the complexity of polynomial computations. The FFT appears to be the obvious best choice; however, the use of FFTs in the implementation of polynomial-based set operations may lead to certain subtle technical problems. These problems become particularly serious when *the cardinality of the sets is dynamic*. Furthermore, our experiment shows that Karatsuba's algorithm delivers higher performance than the FFT in our application, provided that a reasonable response time is important. In addition, our experimental implementation demonstrates the *heuristic bound* on cardinality for which Karatsuba's algorithm outperforms the FFT. This value can be used to determine which optimization techniques and which settings are superior choices in the deployment of private set operation schemes.**

Index Terms: **Polynomial arithmetic, private set operations, efficient computation**

I. INTRODUCTION

DATA aggregation and sharing have played a crucial role in various applications that are used as part of distributed network services (*e.g.,* cloud services and ad hoc networks). In terms of ensuring secure data aggregation and sharing, the largest breakthrough has been the development of efficient privacy-preserving techniques and protocols for computation over sets or multisets by mutually distrusting parties, in which no party should gain any information about the other parties' private input sets other than the resulting set. We call this the *private set operation* (PSO) problem.

Extensive studies have been performed in the attempt to solve this fundamental problem in various settings. The first results were based on general multiparty computation (MPC) [1], [2]. In this approach, users share the values of each input and cooperatively evaluate a function represented by a circuit. Thus, general MPC solutions tend to exhibit high computation and communication complexity because the depth of the circuit that represents the function dominates the complexity. As a typical example, an MPC solution for set intersection that is secure against a malicious adversary requires $O(n^3 k \log^{\nu} k)$ communication complexity for n users whose input set has cardinality k for a constant ν.

This high complexity is the primary reason that a special-purpose protocol that is more efficient than the corresponding general-purpose MPC solution must be developed. As a first attempt in this line of research, Freedman *et al.* proposed the first efficient privacy-preserving set intersection operation using polynomial representation [3]. Broadly speaking, polynomial representation refers to the representation of sets (or multisets) as the roots of a polynomial. Later, we will describe this technique in more detail. Kissner and Song [4] subsequently showed that polynomial representation allows for more diverse set operations even while allowing for a stronger adversary, also known as a malicious adversary. This result was followed by a series of papers (*e.g.*, [5], [6], [7], [8]). One common property of the subsequently developed schemes is that the cost of the polynomial arithmetic operations determines their overall performance.

Hereafter, for the purposes of consistency and convenience in terminology, we will use the term "private" rather than the term "privacy-preserving," following the state-of-the-art convention (*e.g.*, [9]). Accordingly, the reader may interpret private set operations as privacy-preserving set operations if he or she is more familiar with the latter.

Now, we will describe the specific details of the problem on which we focus in this work. The reader is advised to consult Appendix A while reading this section if necessary.

A. Problem Statement

Consider a real-world application using a private set intersection scheme. For example, to determine which airline passengers should appear on a 'do-not-fly' list, an airline company must utilize an application to calculate the intersection of its private passenger list and the do-not-fly list issued by a government. We searched for any previous work reporting a real implementation of such an application. However, to the best of our knowledge, there are no studies that address any such real implementation based on polynomial representation.

In this situation, one of the most promising approaches is to develop a proof-of-concept (PoC) implementation of such an application. Indeed, such an implementation is the major contribution of this paper. Our primary interest in realizing this cryptographic application is to investigate which components of these applications act as performance bottlenecks, thereby resulting in performance problems, and what is the optimal method of avoiding such performance bottlenecks.

Regardless of the details of the implementations, existing polynomial-based set operation schemes (*e.g.*, [3], [4], [6], [7]) rely on the following three polynomial arithmetics. Here, $R[x]$

M. Kim is with the Department of Information Security, College of Information Technology, The University of Suwon, 17 Wauan-gil, Bongdam-eup, Hwaseong-si, Gyeonggi-do 18323, South Korea, email: msunkim@suwon.ac.kr, and is the corresponding author.

B. Kim is with the Paradigm Shift Academy, Daejeon 34052, South Korea, benjaminzkim@gmail.com.

is a ring of polynomials in x with coefficients in a ring R.

1. *Polynomial expansion.* Given a polynomial $\varphi(x) = \prod_{i=1}^{k}(x - \alpha_i) \in R[x]$ in a linear factored form, compute the expansion coefficients $\varphi[i] \in R$ for each term of φ:

$$\varphi(x) = \sum_{i=0}^{k} \varphi[i]x^i = \varphi[k]x^k + \cdots + \varphi[1]x + \varphi[0].$$

2. *Polynomial multiplication.* Given two polynomials in $R[x]$, namely, $\varphi(x) = \sum_{i=0}^{k_1} \varphi[i]x^i$ and $\gamma(x) = \sum_{j=0}^{k_2} \gamma[j]x^j$, compute

$$\psi(x) = \varphi(x) \cdot \gamma(x) = \sum_{\ell=0}^{k_1+k_2} \psi[\ell]x^\ell,$$

where $\psi[\ell] = \sum_{\ell=i+j} \varphi[i] \cdot \gamma[j]$ for $0 \le i \le k_1$ and $0 \le j \le k_2$.

3. *Polynomial multipoint evaluation.* Given a polynomial $\varphi(x) \in R[x]$ and k points $\langle \alpha_1, \alpha_2, \ldots, \alpha_k \rangle$, compute

$$\langle \varphi(\alpha_1), \varphi(\alpha_2), \ldots, \varphi(\alpha_k) \rangle .$$

There are several useful techniques that can improve the performance of polynomial arithmetics such as those discussed above. For example, FFT-based polynomial multiplication has a computation complexity of $O(k \log k)$ rather than $O(k^2)$ for polynomials of degree k [10].

In our implementations, we also routinely apply known optimization techniques to address the observed performance problems. Surprisingly, we experience several interesting difficulties in the course of deploying these well-known techniques. We begin with a basic constraint that results from the proof of protocol security.

1. We have two immediate options from among widely used encryption schemes.[1] One option is Paillier's cryptosystem [11], and the other is an additive variant of El Gamal encryption [**?**]. Paillier encryption uses a plaintext space $\mathbb{Z}_N$ for an RSA modulus N, whereas the additive El Gamal encryption scheme uses a subgroup $\mathbb{G}_q$ with prime order q of $\mathbb{Z}_p^\times$ for a large prime p. Here, $q = (p-1)/\delta$ is prime and δ is a small positive integer. The problem we face is that proving the security of polynomial-based PSO protocols requires a simulator[2] that must be able to factor polynomials to find their roots. However, factoring a polynomial of degree k over $\mathbb{Z}_N$ is believed to be hard when the factorization of N is unknown; however it can be performed in $O(k^2 \log k \log p)$ time over $\mathbb{Z}_p$.

Thus, we must exclude additively homomorphic encryption schemes that operate on groups of unknown order; this is our first observation.

2. Second, consider PSO schemes with polynomial representation. Each user first converts his or her private set Φ into a corresponding polynomial φ represented by coefficients. Then, for the purpose of fast multiplication, each user should compute the discrete Fourier transform (DFT) and proceed with the remaining steps using the DFT output. At this point, all polynomials will be in the form of point representations. After all steps are completed, the users should re-convert their resulting polynomials into the coefficient representation. Of course, FFT allows DFT to be computed quickly (*i.e.*, $O(k \log^2 k)$ field operations), assuming a primitive m-th root of unity $\omega \in R$, for a positive integer m.

Here, the first question for our application is the following: "is ω always available?" Another important question arises in the case in which both the elements and cardinality of the sets are *dynamic* and thus the PSO module must be frequently invoked: does the FFT remain the best choice in light of the total processing time? We note that the FFT requires two heavy conversion routines, as described above.

3. Let us re-examine the first question. Unless the underlying coefficient ring R contains certain primitive roots of unity, an FFT-based technique cannot be directly applied. In our case, wherein the underlying working group is $\mathbb{G}_q$ and we therefore must rely heavily on FFTs, we should determine whether $\mathbb{G}_q$ has a primitive m-th root of unity, *i.e.*, whether $\mathbb{G}_q$ supports the FFT.

The commonly used method of addressing the above issue is to choose a Fourier prime q such that $q = v2^t + 1$ for two positive integers v and t. At this point, the problem we must address is that we need to modify the key generation algorithm of the additive variant of El Gamal to sample from the set of primes of the form $v2^t + 1$. Specifically, we take $q = v2^t + 1$ for a random v and then perform a primality test routine until we obtain a Fourier prime. We can obtain such a prime within $O(\log q)$ trials, which does not appear to pose a significant problem. However, considering the degree of the polynomial product, we should have $2^t = 2k$. This means that before implementing the El Gamal variant, we should know the value k.

In summary, using the FFT requires that we modify the key generation portion of the El Gamal approach and that we know the cardinality of all sets before implementing the PSO module. Sometimes, however, we do not have access to such information regarding the input sets, whereas at other times, we must implement the PSO module using the El Gamal encryption scheme in the form of a library.

Throughout this paper, when we refer to a *special-purpose setting*, we mean a specific environment required by a particular optimization technique such as the FFT. We refer to the opposite case as a *general-purpose setting*.

Within the context of PSO, the above considerations suggest a reasonable likelihood that FFT is not always the best choice. Therefore, a natural question arises: what is the cardinality of the input sets for which the second-best technique will run faster than the best technique in a special-purpose setting? Once this cardinality value is determined, we can immediately identify the following items.

- In a general setting, the maximum cardinality of sets that the second best technique can handle as efficiently as the best technique.
- The class of optimization techniques that are best suited to our target implementation. More specifically, even if the target implementation allows us to use a special-purpose setting, de-

[1] Of course, other encryption schemes with additive homomorphism can also be regarded as candidates.

[2] The fundamental role of a simulator is to generate a sequence of fake communication transcripts that are indistinguishable from real ones in a computational sense. See [13] for details.

pending on the expected cardinality, the second-best techniques can provide various advantages in deployment because they are easy to understand and, thus, to implement.

B. Our Results

Based on these considerations, we can now describe the scope of this work.

To provide an understanding of our goals and the basics of PSO using polynomial representation, we first review Kissner and Song's private set intersection (PSI) scheme [4]. We choose their PSI scheme because their scheme is one of the most efficient for this purpose and covers various set operations, including intersection and union.

Then, we describe the details of our implementation of a '`do-not-fly`' list application using Kissner and Song's PSI protocol between two parties (*e.g.*, a client and a server). Broadly speaking, our implementation consists of both non-cryptographic components, such as the user interface and networking code, and cryptographic components, including the encryption/decryption algorithms and set intersection operations. We benchmark the processing delay of each sub-module and observe that the network processing latency, which is caused by the PSI routine, is the major bottleneck for the overall performance. More specifically, the computation delay in the PSI routine results in a long network delay.

To improve the processing time of computations for PSI operations, we then apply various optimization techniques to the `do-not-fly` application. We choose three representative optimization techniques: the Karatsuba algorithm [14], the Toom-Cook 3-way multiplication (Toom-3) algorithm [15], and the FFT [16]. More specifically, in the case in which a ring R satisfies a special condition for the FFT technique, we measure the processing times of the optimized application using each technique while varying the number of elements in the sets. Based on these experiments, we attempt to determine the maximum number of elements for which performance comparable to that of the FFT can be achieved. In addition, we attempt to verify the maximum cardinality of sets for which the Karatsuba algorithm can ensure practical performance of the do-not-fly list application. Moreover, the efficiency gains of Toom-3 over the Karatsuba algorithm confirm our expectations. Thus, we consider the Karatsuba algorithm as the primary rival of the FFT, but we believe that the reader can easily estimate the performance advantages of Toom-3 relative to the Karatsuba technique.

We remark that we treat the PSI protocol (*e.g.*, Kissner and Song's PSI) as a block box. Namely, we attempt to improve the execution time of the application without modifying any specifications of the core PSO protocols. Hence, our implementation is equally applicable for any other PSI (more generally, PSO) scheme.

C. Recent Results

The naïve implementation of PSO schemes based on polynomial representation incurs a quadratic computation complexity [4], [5], [6], [7] of the degree of the polynomial. As previously mentioned, to the best of our knowledge, there is no related work that presents a real implementation of a PSO application using polynomial representation. One possible reason for this lack is that the underlying public-key encryption must be carefully chosen. Specifically, 'encrypting a polynomial' means encrypting its coefficients. Thus, because it must support polynomial arithmetics on encrypted coefficients, the underlying public-key encryption should exhibit certain specific properties. We call such a technique *homomorphic* public-key encryption, and we will discuss the details in a later section.

One exception has been presented by Mohassel [17]. In the cited work, Mohassel proposed fast algorithms for the computation of encrypted polynomials using the FFT algorithm. Unfortunately, the author did not present any experimental results to support his arguments. Furthermore, to the best of our knowledge, no PSO research or application based on his results has been published in the open literature.

This paper is organized as follows. We first recall the basic concept of homomorphic encryption and polynomial representation in Section II. This section also introduces various useful definitions and notational conventions. In Section III, we describe practical topics related to implementation for our application program. We divide these topics into two categories: special and general settings. We discuss the details of implementing the Karatsuba and Toom-3 algorithms in PSI protocols to reduce their computational complexity. Then, Section IV analyzes a series of experiments as evidence of our arguments. Section V presents the lessons learned from these experiments in various contexts. We conclude the paper in Section VI.

II. BACKGROUND

In this section, we describe the concept of polynomial representation in §II-B and the cryptographic tool of additively homomorphic public-key encryption in §II-A. We provide a high-level description of polynomial-based PSO schemes based on Kissner and Song's ideas, but we will omit the detailed theory of adversary models, including an adversary's capabilities and the semi-honest and malicious models. We recommend that the reader refer to [13] for the details of these adversary models. We begin by establishing some basic notation that is used throughout the paper.

Notation. As mentioned above, the set R denotes the plaintext domain of the encryption scheme, which will be described later. Let $R[x]$ be a ring of polynomials with coefficients from R. For $\varphi, \gamma \in R[x]$ of degree k, we write $\varphi(x) = \sum_{i=0}^{k} \varphi[i]x^i$, where $\varphi[i]$ denotes the coefficient of x^i in the polynomial φ. For notational convenience, we simply use $\varphi + \gamma$ to denote the addition of φ and use γ and $\varphi \cdot \gamma$ to denote the multiplication of φ and γ.

For an integer $\ell \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, \ell\}$. We further assume that all sets have the same cardinality k unless explicitly stated otherwise.

A. Additively Homomorphic Public-Key Encryption

A public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{E}, \mathsf{D})$ consists of the following algorithms:

- KG is a randomized algorithm that takes a security parameter as input and outputs a secret key sk and a public key pk; pk defines a plaintext space and a ciphertext space.
- E is also a randomized algorithm; it takes pk and a plaintext α in the plaintext space as inputs and outputs a ciphertext c in

the ciphertext space. This process is usually randomized using a randomizer r in a suitable random space. We represent this process as $c = \mathsf{E}_{pk}(\alpha; r)$.

- D takes sk and c as inputs and outputs the plaintext α.

A public-key encryption scheme $\mathcal{E}$ that has *additively homomorphic* properties provides an operation in the ciphertext space that is used to compute the encrypted sum of two plaintexts given only the corresponding ciphertexts. More formally, in an additively homomorphic cryptosystem, there is an operation $\oplus$ such that

$$\mathsf{E}_{pk}(\alpha + \beta) := \mathsf{E}_{pk}(\alpha) \oplus \mathsf{E}_{pk}(\beta),$$

which can be efficiently computed given only $\mathsf{E}_{pk}(\alpha), \mathsf{E}_{pk}(\beta)$, and pk. It is also possible to perform efficient scalar multiplication with a scalar value s through repeated application of the $\oplus$ operation, which is denoted by $\otimes$. In other words, given a known constant s and a ciphertext of $\mathsf{E}_{pk}(\alpha)$, we can efficiently compute the encryption of $s \cdot \alpha$, denoted by

$$\mathsf{E}_{pk}(s \cdot \alpha) := s \otimes \mathsf{E}_{pk}(\alpha) = \underbrace{\mathsf{E}_{pk}(\alpha) \oplus \cdots \oplus \mathsf{E}_{pk}(\alpha)}_{s \text{ times}}.$$

In addition, we require the cryptosystem to provide semantic security [18]. In other words, given two ciphertexts, it must be infeasible for an adversary to find any meaningful relationship between the corresponding plaintexts based only on the ciphertexts.

If three or more users participate in PSO operations, then to prevent a small group of users from compromising privacy, PSO protocols require that the homomorphic public-key cryptosystem must support secure (t, n)-threshold decryption, where n is the number of users and $t \leq n$. In a (t, n)-threshold version of an additively homomorphic cryptosystem, the private key sk is shared among the n users, with each user u_i holding a private share $sk_{i \in [n]}$. Using the private key share sk_i, a user u_i can compute a partial decryption of a ciphertext. To successfully decrypt a given ciphertext, t of the n key shares are required to compute the plaintext by combining t partial decryptions of the ciphertext.

Instantiations. As concrete instances of additively homomorphic cryptosystems, there are two popular candidates: the additive El Gamal cryptosystem [19] and Paillier's cryptosystem [11]. Because of the technical concerns outlined above, we use the additive El Gamal encryption scheme in our experiment, with the following specifications.

- *Key Generation.* The scheme's key generation algorithm takes the security parameter as input and first generates a group description $(\mathbb{G}_q, g, q, p)$, where g is the generator of a cyclic group $\mathbb{G}_q$ of prime order q for a safe prime $p = 2q-1$. Next, it chooses a random integer $x \in \mathbb{Z}_q$ and sets both a public key $y = g^x$ and a secret key x. Hence, $\mathbb{G}_q$ is the plaintext space, and the ciphertext space is $\mathbb{Z}_p \times \mathbb{Z}_p$. Notably, $\mathbb{G}_q$ is the set of all quadratic residues mod p.
- *Encryption.* Given a plaintext message $\alpha \in \mathbb{G}_q$, the encryption algorithm computes the El Gamal ciphertext c as follows:

$$c = (g^r, g^\alpha y^r),$$

where r is a randomizer.

- *Decryption.* Given a ciphertext c, the decryption algorithm parses it as (c_1, c_2) and outputs $c_2 \cdot c_1^{-x}$.

The additive variant of El Gamal encryption is equivalent to the standard El Gamal encryption scheme except for the computation of the El Gamal ciphertext c.

A major drawback of additive El Gamal encryption is that decryption can be performed efficiently only as long as the messages are small. For most PSO applications, however, it is sufficient to determine whether a ciphertext encrypts the plaintext $\alpha = 0$; otherwise, there is no need to decrypt the ciphertext. For example, in our PSI implementation, it suffices for a client to test whether ciphertexts are encryptions of '0' without fully decrypting them.

B. Polynomial Representation

The essential approach represented by the operations proposed by Freedman *et al.* [3] and extended by Kissner and Song is to encode the elements of (multi-)sets as the roots of an encrypted polynomial and to compute the results of several set operations using homomorphism.

The private input multiset $\Phi = \{\alpha_1, \ldots, \alpha_k\}$ of a given user is encoded by defining a polynomial $\varphi(x) = \prod_{i=1}^{k}(x - \alpha_i)$ and then encrypting the coefficients of the resulting polynomial. The result of the multiset operations can now be computed solely by manipulating the encrypted polynomials. The sum of two polynomials can be computed via homomorphic addition of their coefficients. We denote this operation by $f \oplus g$, where f and g are encrypted polynomials. Furthermore, the encrypted result g of the polynomial multiplication of an encrypted polynomial f and a plaintext polynomial φ can be computed via homomorphic addition and scalar multiplication:

$$g[\ell] = \sum_{i=0}^{\ell} \varphi[\ell - i] \otimes f[i], \quad 0 \leq \ell \leq \deg(\varphi) + \deg(f).$$

We denote this operation by $\varphi \otimes f$.

Polynomial-based set operations. Using polynomial addition and multiplication, PSOs can be achieved as follows.

- *Union.* For an encrypted polynomial f and a plaintext polynomial φ representing two sets Φ and Ψ of cardinality k, a polynomial representation of the union $\Phi \cup \Psi$ can be computed via homomorphic polynomial multiplication:

$$\varphi \otimes f.$$

This product contains all roots of f and φ in the corresponding summed multiplicity. Furthermore, from the decryption of $\varphi \otimes f$, one cannot learn any more information than can be gained from $\Phi \cup \Psi$, as proven in [4, Theorem 1].

- *Intersection.* For two encrypted polynomials f and g representing two sets Φ and Ψ of equal size k, their intersection can be computed using the following operations:

$$f \otimes \gamma_1 \oplus g \otimes \gamma_2.$$

Here, γ_1 and γ_2 are random polynomials of degree $\deg(f)$. The roots of the resulting polynomial include those that are common

to f and g (with minimum multiplicity), but it is possible that some random roots are introduced to the polynomial. To fix this problem, we use a filtering step by which only the elements in the original set Φ (and Ψ) are evaluated so that we can get the elements of $\Phi \cap \Psi$. Again, from the decryption of the resulting polynomial, one cannot learn more than can be learned from $\Phi \cap \Psi$. This is proven in [4, Lemma 2 & Theorem 3] for same-cardinality sets.

Kissner and Song's PSI scheme. We describe this PSI protocol graphically.

	Client (u_1)	Server (u_2)
	$\Phi = \{\alpha_1, \ldots, \alpha_k\}$	$\Psi = \{\beta_1, \ldots, \beta_k\}$
1.	Setup	
2.	$\phi(x) \leftarrow \prod_{i=1}^{k}(x - \alpha_i)$	$\psi(x) \leftarrow \prod_{i=1}^{k}(x - \beta_i)$
3.	$f(x) := \mathsf{E}_{pk}(\phi(x))$	$g(x) := \mathsf{E}_{pk}(\psi(x))$
4.	Send $f(x)$ to u_2	
5.		For $\gamma_1, \gamma_2 \in R[x]$, $h \leftarrow \gamma_1 \otimes f \oplus \gamma_2 \otimes g$
6.		Send $h(x)$ to u_1
7.	$\theta(x) := \mathsf{D}_{sk}(h(x))$	
8.	$\forall \alpha_i \in \Phi, \theta(\alpha_i)$	

In Step 8, the client can learn $\Phi \cap \Psi$ by collecting those $\alpha \in \Phi$ such that $0 = \theta(\alpha)$.

Remark 1: We give a description of PSI protocol based on Kissner and Song's two-party protocol in the semi-honest model while we have several choices to describe PSO protocols (e.g., [3], [8], [20]). The underlying reasons of this choice are two-folds: one is the natiral extension to multi-party setting, and the other is an easy extension to malicious security.

We would like to remark that malicious security would require us to incorporate several additional features. These additional features include zero-knowledge proofs, such as a proof of plaintext knowledge and a proof of correct multiplication. However, because we would like to focus on polynomial arithmetics, we defer the treatment of a maliciously secure implementation and its performance to a future work. See [4] for an in-depth discussion of correctness and security.

III. OUR `do-not-fly` IMPLEMENTATION

This section presents the implementation details of our `do-not-fly` application based on Kissner and Song's PSI protocol. We discuss various design choices that may affect the overall performance and present our prototype implementation. Recall that our primary goal in this work is to identify the cut-off cardinalities of sets for which different algorithms are preferred, which are strongly related to the performances of the Karatsuba, Toom-3, and FFT algorithms.

A. Important Design Settings

We now identify and discuss several factors that significantly affect the overall performance of the `do-not-fly` implementation. We begin with straightforward issues and then turn to some less trivial strategies. We remark that for the sake of generality, we assume below that the participating users do not perform any pre-computations.

- *PoC implementation.* The `do-not-fly` application is implemented in C++ using the NTL library for large-integer arithmetic [**?**] and OpenSSL for hash function invocations. Our implementation is essentially single threaded; therefore, we utilize only one of the four cores available on the machine. We present the details of our testbed in Section IV.
- *A special-purpose setting.* As mentioned above, certain optimization techniques require special settings to perform properly. Recall that we must choose a particular plaintext domain $\mathbb{G}_q$ with a Fourier prime q of the form $v2^t + 1$ such that $2^t = 2k$. Here, we present an example result in Table 1, which lists the average processing times for additive El Gamal key generation in this special setting. See §IV for detailed results.

Table 1. Benchmarking of additive El Gamal key generation

	$t = 10$	$t = 11$	$t = 12$	$t = 13$
KG$(\cdot)$ [sec]	92.452	143.784	94.630	112.158

- *A general-purpose setting.* In this case, we need only find a group description $(\mathbb{G}, g, q, p)$ to ensure semantic security in the standard El Gamal encryption scheme. On the same machine, 31.73 seconds on average are required for the execution of the same algorithm.
- *Data format.* We encode an encrypted polynomial as a list of pairs of the coefficient and the degree of the indeterminate for each term. In other words, we declare an encrypted polynomial $f(x) := \langle \mathsf{E}_{pk}(\varphi[0]), \ldots \mathsf{E}_{pk}(\varphi[k]) \rangle$ as follows:

```
class EncryptedPolynomial {
...
public:
    int        degree;
    Ciphertext coeff;
...
};
```

Hence, the length of an encrypted monic polynomial of degree k is $260k$ bytes.

Note that the objective of this study is not to improve the performance of the naïve implementation using various optimization techniques. Rather, we first implement the `do-not-fly` application by employing the FFT algorithm. Note that this implementation should rely on a set of parameters that is customized for the FFT algorithm. We measure the processing times for the main components to identify the major bottlenecks. Next, to support more general settings, we also implement applications employing the Karatsuba and Toom-3 algorithms.

B. FFT-based Implementations

Here, we report on the implementation of the `do-not-fly` application using the FFT algorithm. Because q is of the form $v2^t + 1$ with $t = \log k + 1$, we should repeatedly run the `do-not-fly` application using different parameters $(\mathbb{G}, q, g, p)$ and key pairs (y, x) in accordance with the cardinality k of the sets. For example, Table 2 reports several such parameters for $t = 15$. These values were generated by the C++ code presented in Figure 1.

Our implementation is based on Mohassel's algorithms, as proposed in [17]. The author presents fast algorithms for com-

Table 2. Parameters used in our FFT-based implementation for $t = 15$

p :	176821459277259303421583865232327491029329315986787879588887348471015046336109568378039781839096440152274
	845047474468286445305145839919932602232825753095006134906679011561330242249939544160932857164809408073246
	223104610036502719936317526629926476710326459202569858995845954642968927072476545462888595589052419
q :	884107296386296517107919326161637455146646579933939397944436742355075231680547841890198909195482200761374
	723414322265257291995996630111641287654750306745333950578066512112496977208046642858240470225237340366231
	115523050182513599681587633149632383551632296012849294979229773214844635362382727314442977945262O9
v :	863386031627242692488202466954724077291647050716737693305114006206128155938035001845897372261213086681029
	516833371427179908716532421484045909339969497534209643099018611139307823486032930473049666250459378576 47
	573752978693860937189050422997687874562140914075048139628154075405121714221076882143010720649 67
g :	112771027101018250729779708193195388338056472597126063321957170418708226388342317384362332080456458862447
	580298138650511193305319013082227362074073006041816626042191898512953934217265990541977999942194466866441
	610516750820287592776564917612174854259653020871144260785973996582504721408379343188453629073594298

```
...
w = pow(2, t);
do {
    do {
        RandomLen(v, 1024 - t - 1);
        q = w * v + 1;          // q = v2^t+1
    } while (!ProbPrime(q, 80));
    p = 2 * q + 1;
} while (!ProbPrime(p, 80));

while (found == false) {
    do {
        RandomBnd(g, p);     // g: generator
    } while (g < 2 || g > p-2);
    PowerMod(x, g, 2, p);
    PowerMod(y, g, q, p);
    if (x != 1 && y != 1)
        found = true;
}

do {
    RandomBnd(x, q);     // x: secret key
} while (x < 2 || x > q-2);
PowerMod(y, g, x, p);     // y: public key
...
```

Fig. 1. Additive El Gamal key generation code

putations using encrypted polynomials. Indeed, his algorithms rely heavily on the FFT algorithm, and thus, we can directly utilize these algorithms without additional modification. The underlying reason for the applicability of the FFT to encrypted polynomials is that the FFT algorithm consists only of scalar multiplications and additions.

We generate all set elements using the NTL pseudorandom function, $\texttt{RandomBits}(\cdot, \lambda)$, which outputs a positive integer smaller than 2^λ. We set $\lambda = 1024$; thus, our implementation can handle strings of up to 128 characters (not bytes) in length. However, we encode the set elements into the coefficients in $\mathbb{G}_q$ using OpenSSL $\texttt{SHA1}(\cdots)$ invocations and a few additional computations.

C. General-Purpose Implementations

Now, we consider a general setting rather than customized settings, as above. As explained in §I-A, if a well-engineered crypto-library does not allow us to invoke the working group $\mathbb{G}_q$ created in §III-B, then we can no longer use FFT-based optimization algorithms. In this situation, we must adopt different optimization techniques. In this paper, we apply two techniques: the Karatsuba and Toom-3 algorithms. Their computational complexities are higher than that of the FFT, and thus, we call them second-best solutions. It is well known that the former algorithm has $O(k^{1.585})$ complexity and that the latter algorithm has $O(k^{1.465})$ complexity in degree k.

Although our FFT-based implementations are based on Mohassel's algorithms, our general implementations are based on the blueprint described in the following sections. We omit detailed specifications due to page limitations. For clarity and concreteness, our description refers to Karatsuba's algorithm.

C.1 Encrypted Polynomial Multiplication

We show that Karatsuba's algorithm functions well on encrypted polynomials that represent multisets and therefore allows us to achieve sub-quadratic computation complexity for multisets of cardinality k.

To this end, we first assume that a random polynomial can be multiplied by a given encrypted polynomial. We denote the random polynomial by $\gamma(x) = \sum_{i=0}^{k} \gamma[i]x^i \in R[x]$ and the encrypted polynomial by $f = \mathsf{E}_{pk}(\varphi)$, where $\mathsf{E}_{pk}(\varphi) := \sum_{i=0}^{k} \mathsf{E}_{pk}(\varphi[i])x^i$.

Algorithm 1 illustrates an algorithmic workflow using the Karatsuba approach for polynomial multiplication on ciphertexts, where for two ciphertexts a and b, $a-b$ means $a \oplus (-1) \otimes b$. If $\psi = \gamma \cdot \phi$, then we obtain the encryption of polynomial ψ of degree $2k$. The algorithm produces a list of $2k+1$ encrypted coefficients, *i.e.*,

$$\langle \mathsf{E}_{pk}(\psi[0]), \mathsf{E}_{pk}(\psi[1]), \ldots, \mathsf{E}_{pk}(\psi[2k]) \rangle .$$

Next, we argue and prove that Algorithm 1 has $O(k^{\log_2 3})$ computational complexity. For further details on the asymptotic complexity of polynomial multiplication, see [22].

Proposition 1: Algorithm 1 has an asymptotic complexity of $O(k^{\log_2 3})$.

Proof: To simplify the analysis of the complexity of Algorithm 1, we assume that the number of coefficients k is some power of a positive integer, *i.e.*, $k = \lambda^m$.

We see that the number of h_i is k and that the number of h_{st} is equal to the number of all possible pairs of k coefficients, *i.e.*, $\frac{k(k-1)}{2}$. Therefore, the total number of multiplications is

$$\left(\frac{1}{2}\lambda^2 + \frac{1}{2}\lambda\right)^m = \left(\frac{1}{2}\lambda^2 + \frac{1}{2}\lambda\right)^{\log_\lambda k} = k^{\log_\lambda\left(\frac{1}{2}\lambda^2 + \frac{1}{2}\lambda\right)}.$$

Algorithm 1 Encrypted polynomial multiplication

Input: γ and $\langle f_i = \mathsf{E}_{pk}(\varphi[i])\rangle_{i=0}^{k}$
Output: $\langle \mathsf{E}_{pk}(\psi[i])\rangle_{i=0}^{2k}$ such that $\psi = \gamma \cdot \varphi$
1: **for** $i = 0$ **to** k **do**
2: $\quad h_i \leftarrow \gamma \otimes f_i$
3: **end for**
4: **for** $i = 1$ **to** $2k-1$ **do**
5: $\quad$ **for** s, t such that $s + t = i$ and $t > s \geq 0$ **do**
6: $\quad\quad h_{st} \leftarrow (\gamma[s] + \gamma[t]) \otimes f_s \oplus f_t$
7: $\quad$ **end for**
8: **end for**
9: $g_0 \leftarrow h_0$ and $g_{2k} \leftarrow h_k$
10: **for** $i = 1$ **to** $2k-1$ **do**
11: $\quad$ **if** i is odd **then**
12: $\quad\quad g_i \leftarrow \bigoplus_{s+t=i} h_{st} - \bigoplus_{s+t=i} (h_s \oplus h_t)$
13: $\quad$ **else**
14: $\quad\quad g_i \leftarrow \bigoplus_{s+t=i} h_{st} - \bigoplus_{s+t=i} (h_s \oplus h_t) \oplus h_{i/2}$
15: $\quad$ **end if**
16: **end for**
17: **return** $\langle g_i \rangle_{i=0}^{2k}$

If $\lambda = 2$, then we obtain a complexity of $k^{\log_2 3}$. This completes the proof. □

C.2 Encrypted Multipoint Polynomial Evaluation

Although Horner's method [23] is clearly beneficial in reducing the number of multiplications, the asymptotic complexity remains $O(k^2)$ given k points.

To overcome this barrier, we use a divide-and-conquer strategy such as the Karatsuba approach. Let f be an encrypted polynomial, and let φ be a polynomial representation of a multiset $\Phi = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$, *i.e.*, $\varphi(x) = \prod_{i=1}^{k}(x - \alpha_i)$. Let us assume that f is the resulting polynomial in encrypted form and that $\deg(f) = 2k$ for simplicity.

We begin by dividing φ into ν polynomials, denoted by $(\varphi_1, \varphi_2, \ldots, \varphi_\nu)$, of degree $\eta = k/\nu$ and then apply the division algorithm theorem. By the division algorithm, there exist some $q, r \in R[x]$ such that $f = q \cdot \varphi - r$ and $r = 0$ or $\deg(r) < \deg(\varphi)$. In addition, it is straightforward to confirm that $f(\alpha) = r(\alpha)$, where α is a root of φ. Now, we present an efficient algorithm for multipoint polynomial evaluation in PSO protocols.

Proposition 2: Algorithm 2 can be used to evaluate an encrypted polynomial at k points using at most $O(k^{1.28})$ multiplications.

Proof: For simplicity, we assume $k = 2^m$ for an integer $m \geq 0$. As stated above, let $f(x) = \sum_{\ell=0}^{2k} f[\ell]x^\ell$ and let $\varphi(x) = \prod_{i=1}^{k}(x - \alpha_i)$. The division algorithm theorem states that there exist some $q, r \in R[x]$ such that $f = q \cdot \varphi + r$ and $r = 0$ or $\deg(r) < \deg(\varphi)$. Furthermore, we see that $f(\alpha_i) = r_j(\alpha_i)$ for all $1 \leq i \leq k$ and $1 \leq j \leq \nu$.

In Line 1, we then need to expand each polynomial φ_j. Namely, we must compute $\varphi_j(x) = (x - \alpha_{i_1}) \cdot (x - \alpha_{i_2}) \cdots (x - \alpha_{i_\eta})$. By Lemma 1, Algorithm 2 ensures that this computation has an asymptotic complexity of $\eta^{\log_2 3}$.

Algorithm 2 Encrypted multipoint polynomial evaluation

Input: $\Phi = \{\alpha_1, \ldots, \alpha_k\}$ and $f(x) = \sum_{\ell=0}^{2k} f[\ell]x^\ell$
Output: $\langle f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_k)\rangle$
/* ν is the number of sub-polynomials $(\varphi_1, \ldots, \varphi_\nu)$ */
1: Fix the value ν /* $\eta = \deg(\varphi_j)$ */
2: Set $\eta \leftarrow k/\nu$
3: $\varphi \leftarrow \prod_{j=1}^{\nu} \varphi_j(x)$
4: **for** $j = 1$ **to** ν **do**
5: $\quad$ Compute $r_j = f \oplus (-1) \otimes (\varphi_j \otimes q_j)$
6: $\quad f(\alpha_i) \leftarrow r_j(\alpha_i)$
7: **end for**
8: **return** $\langle f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_k)\rangle$

Finally, we must compute a remainder polynomial, r_j, which requires us to perform polynomial division. When we divide f by φ_j, if we use the Karatsuba algorithm for polynomial division, as in Jebelean's integer division algorithm [24], then line 5 requires at most $\nu\eta^{\log_2 3 - 1}$ multiplications. As an alternative, we can use Mohassel's division algorithm [17], which is of $O((2k)^{\log_2 3})$ complexity. In this case, we must apply Karatsuba multiplication in place of FFT multiplication.

Then, we evaluate the polynomial r_j at α_i. If we use Horner's rule, then we need only perform at most η^2 multiplications. Because we must repeat such a polynomial evaluation ν times, the total computational complexity amounts to $\nu(\nu\eta^{\log_2 3 - 1} + \eta^2)$. If we let $\eta = k^\epsilon$ for some real number $\epsilon > 0$, then we have $\nu = k^{1-\epsilon}$ such that $\nu(\nu\eta^{\log_2 3 - 1} + \eta^2) = k^{1-\epsilon}(k^{1+\epsilon(\log_2 3 - 2)} + k^{2\epsilon})$. If we let $\epsilon = 0.28$, then we have $O(k^{1.28})$ complexity in terms of multiplication. Thus, our claim holds. □

Polynomial expansion. Polynomial expansion can be easier than polynomial multiplication because it does not require homomorphic operations on ciphertexts. In other words, polynomial expansion is directly applied to represent an input multiset Φ as a polynomial φ. The main idea is to apply the Karatsuba strategy to expand $\varphi(x)$. We omit the details of the polynomial expansion algorithm and the theory of its asymptotic complexity. The reader can find these details in [10, §8.1].

Lemma 1 ([10]) Polynomial expansion based on Karatsuba's algorithm can be performed with $O(k^{\log_2 3})$ field operations.

Remark 2: *Further performance-enhancing options.* There are several less obvious considerations that can help us to further optimize the performance of our `do-not-fly` application. Although we discuss them below, we defer their implementation to the next version of the PoC implementation because these optimizations apply to the case of specific settings, whereas this paper focuses on general PSO scenarios.

- *Parallelism.* When the bottleneck lies in the computation of polynomial arithmetics, the thread that is responsible for receiving will push the computations into the buffer faster than the thread that is performing the computations can pull them. Karatsuba's algorithm forces a given polynomial to be split into sub-polynomials of lesser degree. When multiple cores are used, the arithmetics for multiple sub-polynomials in the buffer can be performed in parallel.

• *Use of UDP-based Protocols.* If we find the bottleneck to lie in transmission, then we can improve the response time by, for example, using UDP instead of TCP or by choosing socket options that are optimized for the transmission of many tiny packets.
• *Point-based Polynomial Representation.* Representing a polynomial as point representation enables to more efficiently carry out polynomial arithmetics rather than coefficient representation. In this literature this approach was introduced by Cheon et al. in [20].

IV. EVALUATION RESULTS

This section presents a detailed performance evaluation of our `do-not-fly` implementation, and the various lessons learned are summarized in Section V. The reader may first wish to briefly review the next section.

Experimental environments. Our experiments were performed on two different machines: A PSI server was run on an iMac machine with an Intel Core i5 3.5 GHz processor and 16 GB of main memory that was running the Mac OS X 10.10.1 operating system. Our PSI client was run on a Windows 7 PC with an Intel Core i5-3570 operating at 3.40 GHz with 4 GB of main memory. The two systems were connected through a 100 Mbps Ethernet LAN. All methods were implemented using GCC compiler version 4.2.1. In our experiments, we used Shoup's NTL library [**?**]. We measured the average running times. Finally, we varied the cardinality of the sets from 2^7 to 2^{20}.

A. Total Execution Time in a Special-purpose Setting

We first present the total execution times for the `do-not-fly` application running in a *customized* setting. Because we based the implementation on the FFT algorithm, the experimental environments were fixed as described in §III-B. The cardinality of the sets used in our experiments ranged from small $(2^7 \sim 2^8)$ to moderate and large $(2^{15} \sim 2^{20})$, and their elements were all 160-bit $\mathbb{G}_q$ values. Note that whenever the cardinality of the sets was varied, we were required to create a new working group $\mathbb{G}_q$ without increasing the bit length of the set elements.

The time was measured as the difference between the system time at which the application started and the system time at which the application ended. More specifically, we say that the application "started" when the client program initiated its execution (*i.e.*, it invoked the key generation routine), and the application "ended" when the client program output the resulting set, if any. We required the server to begin its computations only after the client had opened a connection on the server's listening socket. Upon establishing the connection, the client program sent to the server a description of the system parameters $(\mathbb{G}_q, q, g, p, y)$. However, we did not include the time spent by the client in sending the parameters or the time spent by the server in waiting for an incoming connection.

Table 3 summarizes the experimental results we obtained in this setting while varying the cardinality of the input sets for each implementation. Here, the total execution time is the sum of the additive El Gamal key generation time, the running times of the client and server PSI programs, and the transmission times between the server side and the client side. Furthermore, the values given are averages over 100 randomly generated sets (for our purposes, it did not matter if the intersection was empty). Unfortunately, we failed to measure the execution times for certain sets of cardinality higher than 2^{17} (in the Karatsuba-based case) or 2^{18} (in the FFT-based case). Moreover, we terminated our application after waiting for approximately 72 hours because we set the maximum delay time to three days. We use a dash ('-') to indicate such a situation.

Table 3. Total execution times in a customized setting (seconds)

k	Karatsuba-based	FFT-based
2^7	98.769	105.105
2^8	240.920	255.564
2^9	256.322	366.618
2^{10}	317.130	1,088.117
2^{11}	1,001.345	1,332.306
2^{12}	2,651.158	2,881.258
2^{13}	7,743.503	5,786.149
2^{14}	23,006.365	15,031.275
2^{15}	68,184.805	33,778.607
2^{16}	203,750.917	76,351.447
2^{17}	608,286.039	170,566.976
2^{18}	-	339,970.680
2^{19}	-	-
2^{20}	-	-

We first investigated the key generation times for our additive El Gamal encryption scheme, which are reported in Figure 2. Interestingly, we observed that the running time of this scheme appears to be independent of the cardinality k. If k is relatively small (*e.g.*, $k = 2^7$ or 2^8), then the key generation time dominates the total execution time in this setting.

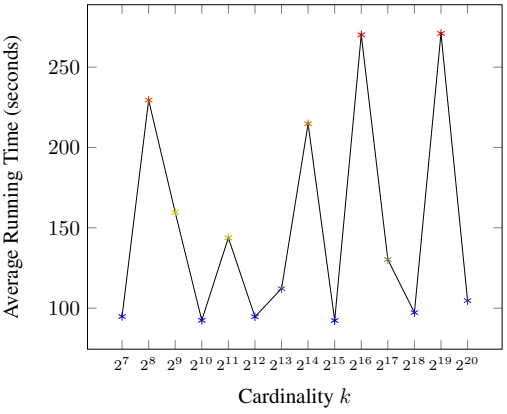

Fig. 2. Average key generation times

Next, we examined the effect of the PSI main program on the total execution time. The key generation time adversely affected the total execution time, but its effect was restricted to the case of small cardinalities. This implies that as the set cardinality k increases, the total execution time begins to strongly depend on the running time of the PSI module. This hypothesis is justified by our observation that the transmission time between users also has no significant impact on the total execution time. Figures. 3

and 4 clearly illustrate how each implementation affects the total execution time of the `do-not-fly` application.

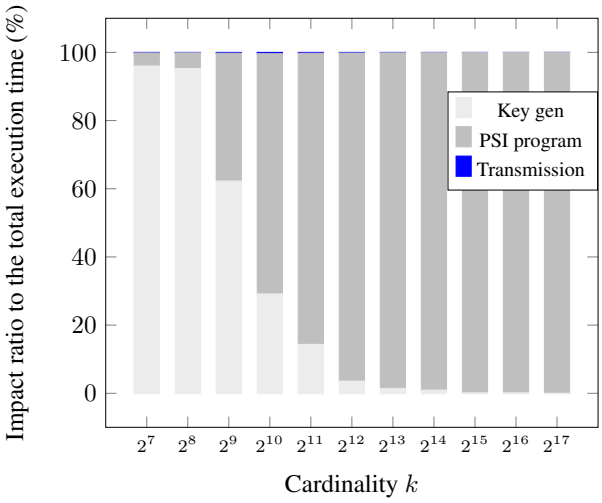

Fig. 3. Impact ratio of our Karatsuba-based PSI program

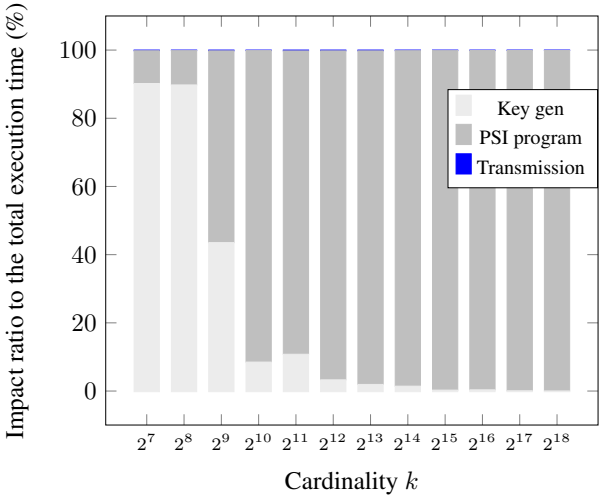

Fig. 4. Impact ratio of our FFT-based PSI program

We can more precisely explain the above graphs by referring to Table 4, which reports the running times observed for the PSI programs implemented using two different techniques. Moreover, Table 5 provides evidence that the communication time between users makes an insignificant contribution to the overall performance.

Finding a Cut-off. We can now discuss the identification of a cut-off cardinality between the conditions for which the Karatsuba-based application is preferred and those for which the FFT-based application is preferred. Table 3 shows that this cut-off cardinality lies between 2^{12} and 2^{13}.

To find a more precise cut-off, we first generated a new Fourier prime $q = v2^t + 1$ such that $2^t \geq 2k$ and proceeded with the remaining steps as before. We then repeatedly ran two different instances of the application while increasing the cardinalities from $k = 2^{12}$ in increments of 10. Table 6 shows that

Table 4. Running times of PSI programs (seconds)

k	Karatsuba-based	FFT-based
2^7	3.81492	10.10092
2^8	11.18280	25.72680
2^9	96.11820	206.21420
2^{10}	223.47640	994.06140
2^{11}	855.15600	1,185.31600
2^{12}	2,551.71800	2,780.21400
2^{13}	7,621.72500	5,661.16500
2^{14}	22,772.38200	14,790.87900
2^{15}	68,054.00600	33,634.98200
2^{16}	203,403.82700	75,978.70500

Table 5. Transmission times among users (seconds)

k	Karatsuba-based	FFT-based
2^7	0.147	0.211
2^8	0.311	0.430
2^9	0.581	0.769
2^{10}	1.192	1.724
2^{11}	2.425	3.216
2^{12}	4.781	6.119
2^{13}	9.411	12.811
2^{14}	17.739	26.004

the cut-off value was 4326. However, we observed that the cut-off did not remain stable at 4326; in different tests, we observed different cut-offs between 4316 and 4346.

Table 6. Cut-off cardinality (seconds)

k	Karatsuba-based	FFT-based
4096	2,581.008	2,891.355
4106	2,600.411	2,896.564
4116	2,623.392	2,907.618
4126	2,651.338	2,914.117
4136	2,683.304	2,912.306
4146	2,711.771	2,921.258
	⋮	
4316	2,968.003	2,971.149
4326	3,019.365	2,981.445
4336	3,062.805	2,998.607
4346	3,113.920	3,011.051
4356	3,150.717	3,010.577

Remark 3: We also tested the performance of the application implemented using Toom-3. We observed that our implementation using the Toom-3 algorithm was $2.17 \sim \mathbf{4.13}$-times faster than the Karatsuba-based application, depending on k. We note that the former algorithm is clearly faster in both a theoretical and practical sense compared with the latter but is substantially more difficult to properly implement in C++. Because the evaluation results for our Toom-3-based application satisfied the theoretical expectations, we decided to omit explicit analysis of its timing results and comparisons with the other two implementations.

B. Experiments in a General-Purpose Setting

The only difference between a special-purpose setting and a general-purpose setting lies in the key generation algorithm. As mentioned above, an FFT-based algorithm may not perform correctly when a general-purpose setting is used. Thus, in such a setting, we can only utilize the Karatsuba or Toom-3 technique. Thus, to optimize the performance in this case, we must focus on the running time of the key generation algorithm, which is 31.728 seconds on average.

V. LESSONS LEARNED

When performing a series of experiments such as those discussed above, much can be learned from the unanticipated results that arise in real-world environments. In this section, we describe some of those results and the lessons learned from them.

A. Two Hidden Multiplications

Even though Karatsuba-based multiplication is obviously faster than the naïve algorithm, our implementation is slower than our theoretical expectations. For example, the case of $k = 2^7$ is approximately 2 seconds slower than expected. This performance degradation results from multiplications hidden in the theoretical analysis. Specifically, Karatsuba's algorithm requires two additional subtractions. In turn, these subtractions require us to multiply several ciphertexts by -1. Fortunately, this multiplication can be far more efficiently implemented. We analyze the additional multiplication cost in the proposition presented below.

Proposition 3: Let pk and $\mathsf{E}_{pk}(\cdot)$ be defined as above, let $r \in R[x]$ be a random polynomial, and let $f = \mathsf{E}_{pk}(\varphi)$ be an encrypted polynomial of $\varphi \in R[x]$, assuming that $\deg(r) = \deg(f) = k$ and $k = 2^m$ for some integer $m \geq 0$. Then, Algorithm 1 (for computing $r \otimes f$) requires us to perform $\left(\lceil k^{\log_2 3} \rceil - 1\right)$ multiplications of -1 with ciphertexts.

Proof: Let $r(x) = r_0 + r_1 x^{k/2}$ and $f(x) = f_0 + f_1 x^{k/2}$, where $r_0, r_1 \in R[x]$ and f_0 and f_1 are encrypted polynomials of degree less than $k/2$. Then, Algorithm 1 will compute $r \otimes f$ by recursively calling

$$\begin{aligned} & r_0 \otimes f_0 + \\ \big((r_0 + r_1) \otimes (f_0 \oplus f_1) & \underbrace{-r_0 \otimes f_0}_{(i)} \underbrace{-r_1 \otimes f_1}_{(ii)}\big) x^{k/2} + \\ & r_1 \otimes f_1 x^k . \end{aligned} \tag{1}$$

Our implementation requires that in Equation (1), the two terms (*i*) a nd (*ii*) must be re-written as follows:

$$\begin{aligned} -r_0 \otimes f_0 &= (-1) \otimes (r_0 \otimes f_0) , \\ -r_1 \otimes f_1 &= (-1) \otimes (r_1 \otimes f_1) . \end{aligned}$$

Hence, if $\mathsf{M}(\mathsf{k})$ is the number of multiplications of encrypted coefficients by -1 that are required to compute Equation (1), then $\mathsf{M}(\mathsf{k}) = 3\mathsf{M}(\mathsf{k}/2) + 2$. Through simple calculations, we find that $\mathsf{M}(\mathsf{k}) = 3^{\mathsf{m}} - 1$ and $\mathsf{M}(\mathsf{k}) = 3^{\log_2 \mathsf{k}} - 1 = \mathsf{k}^{\log_2 3} - 1$. □

For the remainder of this section, we attempt to provide answers to the questions posed in the Introduction section.

B. Discussion

Using homomorphic encryption, as in the El Gamal and Paillier cryptosystems, allows us to achieve strong privacy because it enables us to compute directly over encrypted data. More specifically, additive El Gamal encryption makes it possible to add two ciphertexts and to multiply a ciphertext by a constant without decryption. Instead, we must pay the price of the running time incurred by heavy computations. For this reason, in the literature, sets composed of between 128 and 8192 elements have generally been treated as large sets (*e.g.*, [25][§7.2] and [26][§5.1]).

Thus, it is important to know the *heuristic* bound of cardinality such that real-world PSO applications will be feasible from a practical and rational economic point of view.

To this end, we consider running the `do-not-fly` application over sets of cardinality 2^{14}. According to our experiments (which were, of course, not fully optimized), a client must wait for approximately 4 hours to receive the resulting intersection even when the application uses the FFT technique. These results lead to the following findings.

Lessons 1: Certain users, of course, may accept the 4-hour processing delay if they can achieve strong privacy. However, it is likely that many other users will not be willing to endure such long processing delays. We argue that for general users to expect a reasonable response time, the *heuristic upper bound of cardinality* should be at most 2^{11}, regardless of the underlying optimization techniques, when polynomial representation is employed. According to Table 4, even an FFT-based PSI program requires approximately an hour of processing time for sets of 2^{12} elements.

Lessons 2: Our fine-tuned experiments reported in Table 6 show that the cut-off cardinality between the Karatsuba-based and FFT-based implementations ranges from 4316 to 4346. Because $4316 > 2^{12}$, even if we increase the heuristic upper bound to 2^{12}, for most general users, *Karatsuba's algorithm is a better choice* from a practical point of view. In summary, the FFT-based implementation is faster than the Karatsuba-based implementation for input sets of more than 2^{15} elements. Unfortunately, the heavy computation costs incurred for operations over ciphertexts completely offset the gains from using the FFT. Therefore, for practically meaningful PSO applications using polynomial representation, Karatsuba's algorithm (more precisely, the Toom-3 algorithm) is a better choice as an optimization tool.

Lessons 3: Polynomials are obviously an attractive tool for use as mathematical primitives. However, they incur much heavier computation workloads compared with other tools, such as those used by De Cristofaro *et al.* [26] and Pinkas *et al.* [9]. Our conjecture is that this is one of reasons that polynomial-based PSO applications may not appeal to practitioners in PSO fields.

VI. SUMMARY AND FUTURE WORK

We reviewed polynomial-based PSO schemes and implemented a `do-not-fly` application based on Kissner and Song's

scheme, which demonstrates superior performance compared with previous PSO schemes. We developed the application using three representative optimization techniques: the Karatsuba, Toom-3, and FFT algorithms. First, our experimental study indicated the existence of a heuristic upper bound on the number of elements in the input sets, and this bound carries a practical meaning for the `do-not-fly` application. Second, we attempted to identify a cut-off cardinality between the conditions for which the Karatsuba- and FFT-based implementations are preferred. Finally, we presented several implications of our series of experiments.

In future work, we will provide a full report on performance of set union protocols. We will choose a further variety of set union protocols and look into testing efficiency and practical running times of tested protocol by applying point representation instead of coefficient representation.

REFERENCES

[1] A. Yao, "Protocols for secure computations" in *FOCS*, pp. 160–164, 1982.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *STOC*, pp. 1–10, 1988.

[3] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set-intersection," in *Advances in Cryptology-Eurocrypt* (C. Cachin and J. Camenisch, eds.), LNCS 3027, pp. 1–19, 2004.

[4] L. Kissner and D. Song, "Privacy-preserving set operations," in *Advances in Cryptology-Crypto* (V. Shoup, ed.), LNCS 3621, pp. 241–257, 2005.

[5] Y. Sang, H. Shen, Y. Tan, and N. Xiong, "Efficient protocols for privacy preserving matching against distributed datasets," in *ICICS* (P. Ning, S. Qing, and N. Li, eds.), LNCS 4307, pp. 210–227, 2006.

[6] Y. Sang and H. Shen, "Privacy preserving set intersection protocol secure against malicious behaviors," in *PDCAT* (D. Munro, H. Shen, Q. Sheng, H. Detmold, K. Falkner, C. Izu, P. Coddington, B. Alexander, and S.-Q. Zheng, eds.), pp. 46–468, 2007.

[7] Y. Sang and H. Shen, "Privacy preserving set intersection based on bilinear groups," in *ACSC* (G. Dobbie and B. Mans, eds.), CRPIT 74, pp. 47–54, 2008.

[8] M. Kim, H. T. Lee, and J. H. Cheon, "Mutual private set intersection with linear complexity," in *WISA* (S. Jung and M. Yung, eds.), LNCS 7115, pp. 219–231, 2011.

[9] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *USENIX Security*, pp. 797–812, 2014.

[10] J. von zur Gathen and J. Gerhard, *Modern computer algebra*. Cambridge, University Press, 2003.

[11] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology-Eurocrypt* (J. Stern, ed.), LNCS 1592, pp. 223–238, 1999.

[12] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology-Crypto* (G. Blakley and D. Chaum, eds.), LNCS 196, pp. 10–18, 1984.

[13] O. Goldreich, *The foundations of cryptography: Volume 2–Basic Applications*. Cambridge University Press, 2004.

[14] A. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics-Doklady*, Vol. 7, pp. 595–596, 1963.

[15] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Mathematics Doklady*, Vol. 4, pp. 714–716, 1963.

[16] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, Vol. 19, No. 90, pp. 297–301, 1965.

[17] P. Mohassel, "Fast computation on encrypted polynomials and applications," in *CANS* (D. Lin, G. Tsudik, and X. Wang, eds.), LNCS 7092, pp. 234–254, 2011.

[18] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, 1984.

[19] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung, "Multi-autority secret-ballot elections with linear work," in *Advances in Cryptology-Eurocrypt* (U. Maurer, ed.), LNCS 1070, pp. 72–83, 1996.

[20] J. H. Cheon, S. Jarecki, and J. H. Seo, "Multi-party privacy-preserving set intersection with quasi-linear complexity," *IEICE Transactions*, Vol. 95-A, No. 8, pp. 1366–1378, 2012.

[21] V. Shoup, "NTL: A library for doing number theory version 9.7." available at http://www.shoup.net/ntl/.

[22] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations," *IACR Cryptology ePrint Archive*, Vol. 2006, No. 224, 2006.

[23] W. G. Horner, "A new method of solving numerical equations of all orders, by continuous approximation," *Philosophical Transactions Royal Society of London*, Vol. 109, pp. 308–335, 1819.

[24] T. Jebelean, "Practical integer division with Karatsuba complexity," in *ISSAC* (B. Char, P. Wang, and W. Küchlin, eds.), pp. 339–341, 1997.

[25] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?," in *NDSS*, 2012.

[26] E. De Cristofaro and G. Tsudik, "Experimenting with fast private set intersection," in *TRUST*, LNCS 7344, pp. 55–73, 2012.

Appendix

I. ALGEBRAIC BACKGROUND

An efficient polynomial multiplication algorithm requires that the coefficient ring contain certain roots of unity. We recall that an element a of a ring R is a zero divisor if there exists a nonzero $b \in R$ such that $ab = 0$. In particular, 0 is a zero divisor (unless R is the trivial ring $\{0\}$). The reader should note that in many algebra texts, 0 is not considered to be a zero divisor.

Definition 1: Let R be a ring, $m \in \mathbb{N}$ such that $m \geq 1$, and $\omega \in R$.

1. ω is an m-th root of unity if $\omega^m = 1$.
2. ω is a primitive m-th root of unity (or a root of unity of order m) if it is an m-th root of unity, $m \in R$ is a unit in R, and $\omega^{m/t} - 1$ is not a zero divisor for any prime divisor t of m.

In the following, using the same notation as above, we identify a polynomial $\varphi = \sum_{j=0}^{m-1} \varphi[j]x^j \in R[x]$ of degree less than m with the coefficient vector $(\varphi[0], \ldots, \varphi[m-1]) \in (R)^n$.

Definition 2 (DFT) The R-linear map

$$\mathrm{DFT}_\omega : \begin{cases} (R)^m & \longrightarrow & (R)^m \\ \varphi & \mapsto & \big(\varphi(\omega^0), \varphi(\omega^1), \ldots, \varphi(\omega^{m-1})\big) \end{cases}$$

that evaluates a polynomial at the powers of ω is called the Discrete Fourier Transform (DFT).

The Discrete Fourier Transform is a special multipoint evaluation at the powers $1 = \omega^0, \omega, \ldots, \omega^{m-1}$ of a primitive m-th root of unity ω. The Fast Fourier Transform (FFT) is a method of quickly computing the DFT. It was (re)discovered by Cooley and Tukey in 1965 [16]. The inverse DFT can thus also be computed quickly. See [10][§8.2] for more details on the DFT and FFT.