Implementation and Performance Evaluation of Privacy-Preserving Fair Reconciliation Protocols on Ordered Sets

Daniel A. Mayer, Susanne Wetzel
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030, USA
{mayer, swetzel}@stevens.edu

Dominik Teubert, Ulrike Meyer
RWTH Aachen
UMIC Research Center
Aachen, Germany
{dteubert, meyer}@itsec.rwth-aachen.de

ABSTRACT

Recently, new protocols were proposed which allow two parties to reconcile their ordered input sets in a fair and privacy-preserving manner. In this paper we present the design and implementation of these protocols on different platforms and extensively study their performance.

In particular, we present the design of a library for privacy-preserving reconciliation protocols and provide details on an efficient C++ implementation of this design. Furthermore, we present details on the implementation of a privacy-preserving iPhone application built on top of this library. The performance of both the library and the iPhone application are comprehensively analyzed. Our performance tests show that it is possible to efficiently implement private set intersection as a generic component on a desktop computer. Furthermore, the tests confirm the theoretically determined quadratic worst-case behavior of the privacy-preserving reconciliation protocols on the desktop as well as the iPhone platform. The main result of the performance analysis is that the protocols show linear runtime performance for average-case inputs. This is a significant improvement over the worst-case and is key for making these protocols highly viable for a wider range of applications in practice.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Design, Performance, Experimentation, Security

CODASPY'11, February 21–23, 2011, San Antonio, Texas, USA.

Keywords

Privacy, Ordered Sets, Private Set Intersection, Performance, Multi-Party Computation, Cryptographic Protocol, iPhone

1. INTRODUCTION

Secure multi-party protocols allow two or more parties to jointly compute a function of the inputs of each of the parties without requiring any of the parties to reveal their private input to anyone. Examples for such functions are computing the average of the inputs or determining the intersection of the input sets. In this context, Meyer et al. recently proposed protocols that allow two parties to reconcile their ordered input sets in a fair and privacy-preserving manner [24, 25]. In these protocols, the order of the private input set of a party is associated with the private preferences of that party with respect to the elements of its input sets. Fair reconciliation then means that the new protocols acknowledge the preferences of the parties in that they output only those common elements of the ordered input sets that maximize a common preference order. Applications of these protocols range from reconciling policies in Future Internet Architectures [29, 32] to determining common candidate time-slots while scheduling a meeting without involvement of a trusted server. In this paper we focus on the implementation and performance evaluation of these protocols. In particular, we describe the design and C++ implementation of a library for privacy-preserving reconciliation protocols on ordered sets. The library includes efficient implementations of the Paillier cryptosystem and the privacy-preserving set intersection protocol by Freedman et al. In addition, we describe the implementation of a proof-of-concept iPhone application *appoint* that is built on top of

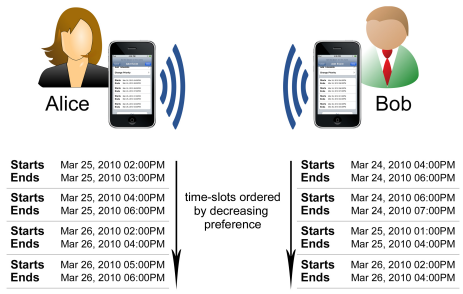


Figure 1: Illustration of *appoint*

the C++ library. *appoint* allows two parties to schedule a meeting, i.e., agree on a meeting time and date in a privacy-preserving manner. Each party has its own schedule which has a number of open time-slots available for potentially accommodating this meeting (see Figure 1). In addition, either party will typically prefer some time-slots over others. Using preference-maximizing reconciliation protocols for ordered sets, the two parties can agree on a time-slot which is open in both schedules and maximizes the parties' common preferences. In this context, it is in the interest of both parties to keep their individual schedules private while still being able to agree on a common meeting time. If one party published its entire schedule, it would allow the other party to possibly deduce sensitive information. Examples include work habits (e.g., the person never starts working before 10 am) or work load (evidenced by many open time-slots).

We provide a comprehensive performance evaluation of our protocols on a desktop computer as well as the iTouch[1]. The experiments on the desktop show that it is possible to efficiently implement private set intersection as a generic component. Furthermore, the tests confirm the theoretically determined quadratic worst-case behavior of the privacy-preserving reconciliation protocols [24]. The main result of the performance analysis on the desktop is that the protocols show linear runtime performance for average-case inputs. This is a significant improvement over the worst-case and is key for making these protocols highly viable for a wide range of applications in practice. The experiments on the iTouch further underline our protocols' practical relevance and show that smaller privacy-preserving applications are already viable on today's smartphones.

The remainder of this paper is organized as follows: Section 2 provides a brief overview on related work. In Section 3, the privacy-preserving tools and protocols are briefly reviewed. Section 4 presents the design and implementation of the library while Section 5.2 describes the implementation of application *appoint*. Finally, Section 6 focuses on the extensive performance evaluations. We close the paper with some remarks on future work.

2. RELATED WORK

Since Yao's seminal paper [35] on secure multi-party computation (SMPC), the field evolved from purely theoretical to practical. For example, recently, Pinkas et al. described the implementation of a two-party protocol using Yao's garbled circuits [30]. Other efforts of implementing general purpose SMPC protocols include [7–9, 22, 23].

Apart from general purpose SMPC, specialized protocols have been introduced to solve the multi-party computation problem for specific operations. One of these operations is private set intersection (PSI) which forms the foundation of the reconciliation protocols implemented and analyzed in this paper. Many PSI protocols have been suggested, e.g., [12–15, 19, 20]. For the performance analysis described in this paper we implemented the protocol suggested by Freedman et al. in [15], which is based on *oblivious polynomial evaluation*. Nevertheless, our overall implementation is designed to allow for a simple integration of any other PSI protocol in the future.

[1]The 3rd generation iPod touch (iTouch) is used for development and testing. It is based on the same hardware as the iPhone 3GS.

Voris et al. [34] previously performed an initial performance evaluation of the privacy-preserving reconciliation protocols introduced by Meyer et al. [24]. However, their work has shortcomings in that the analysis left open a number of crucial questions. In particular, the work in [34] does not provide explanations on the characteristics of the experimental results, e.g., why the experimental runtime is approaching a limit for larger set sizes. This especially is a problem since the experimental results are in disagreement with theoretical predictions. In addition, the experiments in [34] were not performed in a well-controlled test environment. That is, both, the length of the set elements as well as the number of elements in a set were changed at the same time. Furthermore, sets were generated at random (up to 1 million entries) but only 10 identical runs were averaged for each data point. Yet, the large number of possible set configurations would require significant averaging over different randomized sets in order to obtain a representative result. Overall, the experimental analysis done to date does not allow for any conclusions on the true practical behavior of the protocols introduced in [24].

The focus of this work is on analyzing the behavior of the recently revised protocols [25] in the worst and average case—with a particular focus on answering the open questions of [34] and presenting solid experimental evidence for the performance of the protocols. Furthermore, this work strives to provide a better understanding for the practical performance of the private reconciliation protocols and thus shows their suitability for a wide range of applications in practice. In particular, our proof-of-concept iPhone application underlines the protocols' practical relevance.

Today, SSL and PKI functions are part of any smartphone API (e.g., [2, 5]). Furthermore, the increased performance of mobile devices gives rise to more involved security- and privacy-related applications (e.g., [10, 16]). To the best of our knowledge, however, to date the field of secure multi-party computation on mobile devices is mostly unexplored. Our proof of concept iPhone application is a first step to investigate whether or not privacy-preserving applications are already viable on smartphones.

3. PRELIMINARIES

3.1 Homomorphic Encryption

The privacy-preserving tools, which are at the core of the privacy-preserving reconciliation protocols by Meyer et al., require an additively homomorphic and semantically secure public key cryptosystem [15].

DEFINITION 1 (HOMOMORPHIC ENCRYPTION). *An encryption scheme is said to be homomorphic if for given encryptions $E_k(m_1)$, $E_k(m_2)$ it holds for all encryption keys k that $E_k(m_1 \oplus m_2) = E_k(m_1) \otimes E_k(m_2)$ for some operators $\oplus$ in the plaintext space and $\otimes$ in the ciphertext space.*

It is important to note that the operation $\otimes$ can be performed without prior decryption.

DEFINITION 2 (SEMANTIC SECURITY). *In a semantically secure setting an adversary does not learn anything about the plaintext (besides its length) from given ciphertexts [18].*[2]

[2]Semantic security is often referred to as *indistinguishability of encryptions*.

In the following, we will use the Paillier cryptosystem [28]. It is a semantically secure public key cryptosystem based on the composite residuosity assumption preserving the group homomorphism of addition. That is, given two ciphertexts $E(m_1)$, $E(m_2)$ one can efficiently compute $E(m_1 + m_2)$. Similarly, $E(m_1 \cdot c)$ can be obtained given only $E(m_1)$ and a constant c.

3.2 Adversary Model

Following the work in [25], this paper focuses on a passive adversary who is honest but curious and is generally referred to as *semi-honest*. This can informally be described as a person, algorithm, or program which follows the protocol and performs all required computations. Yet, it might store intermediate results and do additional polynomial time computations with the goal of learning as much as possible. A formal definition is given in [17]. A protocol is said to be privacy-preserving in the semi-honest model if no party learns anything but what can be derived from the output of the protocol and its own private input. In general, semi-honest adversaries are contrasted by a malicious adversary who can show arbitrary behavior.

3.3 Oblivious Polynomial Evaluation

Freedman et al. (FNP) developed a set of tools to facilitate privacy-preserving set intersections. Their scheme requires a semantically secure homomorphic cryptosystem such as Paillier's [15]. FNP's construction assumes a chooser C and a sender S and allows these two parties to compute the intersections $X_C \cap X_S$ of their private sets X_C and X_S (chosen from the same domain). At the end of the protocol, C only learns which elements C and S have in common and S learns nothing. The technique used by FNP is oblivious polynomial evaluation. Oblivious means that S only sees encrypted coefficients and evaluates the polynomial without having access to its actual coefficients.

The protocol works as follows: First, party C generates a polynomial containing all her set elements c_i as roots:

$$p_C(x) = (x - c_1) \cdot (x - c_2) \ldots (x - c_n) = \sum_{i=0}^{n} \alpha_i \cdot x^i \quad (1)$$

Then, C encrypts all α_i using her public key and sends them to S. S chooses a random r_i for each s_i in his set and obliviously computes $p_C^{s_i} := E_C(r_i \cdot p_C(s_i) + s_i)$. Party S then sends all $p_C^{s_i}$ back to C. C uses her private key to decrypt the result. Note that $p_C^{s_i}$ decrypts to $s_i \in X_C$ if s_i is in X_C and to a random number otherwise. Thus, C can tell which elements of S are also in her set. In order for S to be able to also determine the intersection, the protocol is executed in the opposite direction. The protocol is privacy-preserving in the semi-honest model.

3.4 Privacy-Preserving Reconciliation

In the following we assume that party A's (B's) private input is a set S_A (S_B) of size n with elements a_i (b_i) for $1 \leq i \leq n$.

The $3PR^C$ protocols [24] introduce fairness into the reconciliation process. Each set element is associated with a preference, i.e., its position within the set—also referred to as *rank* of an element. The respective set is referred to as *ordered set*. Given such an ordered set, the $3PR^C$ protocols allow two parties A and B to reconcile their ordered sets (assumed to be from the same domain and of equal length) such that the result maximizes a *combined preference order* $\leq_{AB}$ under a *preference order composition scheme* C.

The $3PR^C$ protocols by Meyer et al. do not inherently enforce a specific preference order composition scheme. Instead, a preference order composition scheme C may be chosen based on a particular application. In [24], two preference order composition schemes were introduced which implement common notions of fairness: *Sum of Ranks* and *Minimum of Ranks*:

Definition 3 (Sum of Ranks).

$$\forall x, y \in S_A \cap S_B : \quad x \leq_{AB} y :\Leftrightarrow \\ rank_A(x) + rank_B(x) \leq rank_A(y) + rank_B(y) \quad (2)$$

where $rank_A(x)$ and $rank_A(y)$ ($rank_B(x)$ and $rank_B(y)$) correspond to the preference of x and y in set S_A (S_B).

Definition 4 (Minimum of Ranks).

$$\forall x, y \in S_A \cap S_B : \quad x \leq_{AB} y :\Leftrightarrow \\ min(rank_A(x), rank_B(x)) \leq min(rank_A(y), rank_B(y)) \quad (3)$$

where $rank_A(x)$ and $rank_A(y)$ ($rank_B(x)$ and $rank_B(y)$) correspond to the preference of x and y in set S_A (S_B).

The $3PR^C$ protocols [25] encompass multiple rounds. In each round, pairs of set elements are compared according to the respective combined preference order $\leq_{AB}$. For the sum of ranks composition scheme, the protocol includes up to $2n - 1$ rounds. In round $1 \leq i \leq 2n - 1$, all pairs of elements are compared where the sum of the ranks of these elements equals $2n + 1 - i$. For the minimum of ranks, the protocol includes at most n rounds. In round $1 \leq i \leq n$, the elements with rank i are compared with all elements having rank greater or equal to i. The comparisons in the $3PR^C$ protocols which were first introduced in [24] are based on the oblivious polynomial evaluation by Freedman et al. Specifically, comparing the two set elements a_i and b_j works as follows: Party A creates a polynomial $p_A(x) = (x - a_i) = \sum_{i=0}^{1} \alpha_i \cdot x^i$, encrypts the coefficients α_i with her public key and sends them to B. Party B obliviously evaluates the polynomial using b_j, i.e., computes $p_A^{b_j} := E_A(r_j \cdot p_A(b_j) + b_j)$ and sends the result to A. Party A now decrypts the result and checks whether it matches the corresponding a_i. If a match was found, she notifies B (through specific match confirmation operations), otherwise the protocol continues with a pair of set elements having an equal or lower combined preference. Note that this is a simplified description of the actual protocols [24]. In the following, we will refer to these protocols as $3PR^C$-*FNP-implicit* as the oblivious polynomial evaluation of the Freedman protocol is implicitly used in these protocols.

As described in [25], it is possible to generalize the original $3PR^C$ protocols in that the comparison may be carried out using an arbitrary PSI protocol. More specifically, it is possible to modularize the original protocols such that any PSI protocol may be explicitly used as a building block for the $3PR^C$ protocols. In the following, we distinguish between two variants of the modular protocols—$3PR^C$*-PSI-explicit-cached* and $3PR^C$*-PSI-explicit-non-cached*. These two protocols differ in that $3PR^C$*-PSI-explicit-cached* implements a caching mechanisms which allows the reusing of data. In particular, any set element may be part of comparisons at

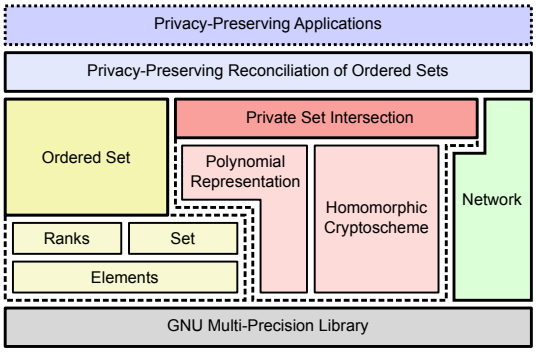


Figure 2: Building blocks of the PROS library.

various rounds. Instead of having to resend the corresponding data in each one of these rounds, the caching mechanism implemented in $3PR^C$*-PSI-explicit-cached* allows the eliminating of these additional data exchanges.

All the $3PR^C$ protocols (i.e., $3PR^C$*-FNP-implicit*, $3PR^C$*-PSI-explicit-non-cached*, and $3PR^C$*-PSI-explicit-cached*) were shown to be privacy-preserving in the semi-honest model [25], i.e., upon termination of the protocols, parties A and B have learned nothing but the set element that maximizes the combined preference order.

4. DESIGN AND IMPLEMENTATION OF THE PROS LIBRARY

The first main contribution of this paper is our new design of a modular library for *Privacy-Preserving Reconciliation of Ordered Sets* (PROS) which can be used to efficiently build privacy-preserving applications. Figure 2 illustrates the design of our PROS library. Each main component is coded in a different color and all interacting components use well-defined interfaces. Therefore, specific implementations for each component can be exchanged with other implementations–assuming they provide the same functionality—without requiring changes to any of the other components. The PROS protocols (blue) take ordered sets as inputs (yellow). An ordered set is a set combined with the ranks of its elements. Furthermore, the PROS protocols use PSI protocols (red). Currently, only Freedman's PSI, which operates on a polynomial representation of the data and uses a homomorphic cryptosystem, is implemented. The network layer (green) enables the communication between the different parties. All computations possibly involve long integers. This, in particular, applies to operations involving set elements, polynomials as well as cryptographic keys. The library builds on the well-known and efficient *GNU Multi-Precision Arithmetic Library* (GMP) [4]. In the following, we describe the different components in greater detail.

4.1 Ordered Sets

To allow for the flexibility of accommodating a multitude of sets containing different types of elements in the future, operations involving sets in our new library were implemented on an interface-defining virtual `ordered set` class. Internally, ordered sets are represented as a vector of integers and the position within the vector implicitly specifies the element's rank. The ordered set of integers is an implementation of the virtual `ordered set` class and provides all the necessary functions. New set element types, which are different from plain integers, can easily be integrated into our library. In Section 5.2 this will be demonstrated for our iTouch application *appoint* which enables the scheduling of a meeting in a privacy-preserving manner.

4.2 Homomorphic Cryptosystem

The efficiency of the implementation of the homomorphic cryptosystem has great influence on the overall performance of privacy-preserving protocols. Each round of these protocols typically involves several encryption/decryption operations as well as homomorphic operations (e.g., homomorphic addition or homomorphic multiplication by a constant). It is therefore crucial to develop efficient implementations of these operations. The cryptosystem is implemented as an individual class and it provides algorithms for key management and exchange, encryption/decryption, as well as homomorphic operations.

In particular, the Paillier cryptosystem was implemented following the original description in [28]. It provides homomorphic operations for addition and multiplication by a constant. To determine primes p and q for the Paillier cryptosystem, the *key generation* uses the Mersenne-Twister pseudo random generator (PRG) provided by the GMP library to first generate a random number of desired length. The actual primes are determined as the next prime greater than the random number generated by the PRG. The UNIX `/dev/random` device provides the seed for the PRG. GMP provides basic functions to perform modular arithmetic. More involved constructions, as required by the cryptosystem, were implemented from scratch.

Whenever possible, *performance enhancements* were introduced in our implementation of the Paillier cryptosystem. For example, all constants used during encryption and decryption are precomputed at the time of key generation and stored for later use. Furthermore, the decryption process involves exponentiation modulo n^2 which can be significantly accelerated by first performing the calculations modulo the square of the two primes and then combining the result using the *Chinese Remainder Theorem (CRT)* modulo n^2 afterward [28].

The addition of other homomorphic cryptosystems to the PROS library is part of future work.

4.3 Polynomial Representation

Polynomials and operations on them are defined by a virtual `polynomial` class, which, in particular, provides functions to efficiently multiply polynomials (e.g., as required by Equation 1 in Freedman's PSI).

Two classes implementing the `polynomial` interface were developed: `vectorpolynomial` and `flintpolynomial`. The latter is based on *FLINT: Fast Library for Number Theory* [3] and was introduced to leverage the efficient algorithms provided by this library. Each class handles the representation of as well as all operations on the polynomials. In `vectorpolynomial`, polynomials are represented as vector of their coefficients. Since the coefficients represent large integers, they are stored as `mpz_class` objects, the native data type of GMP. `flintpolynomial` stores polynomials in the native data types of *FLINT*, which are based on *GMP* as well.

4.4 Network

In order to enable the execution of the reconciliation protocol, a communication interface between the (two) parties

must be established. In our library, this was implemented through UNIX Sockets using the Transmission Control Protocol (TCP). Consequently, our protocols can be executed over basically any network used today.

A basic network interface was developed which allows sending and receiving integers as well as arbitrary byte-arrays over the network. This basic `network` class was extended to include a 32-bit header for all messages. This method allows for an efficient transmission of messages of arbitrary size.

In addition, the extended class provides functions to directly send `mpz_class` objects. GMP's export function is used to split long integers into blocks of 32 bits. These can be transmitted at once and re-imported into an `mpz_class` object, thus avoiding expensive string conversions.

When using FNP's PSI protocol, the individual amount of data transmitted in each round of the PROS protocols is small (bounded by n^2). To avoid transmission delays, the `TCP_NOWAIT` option is used when creating the TCP socket. This option disables *Nagle's algorithm*, a performance enhancement for TCP [21, pp. 815-816], and enables immediate transmission after each `send` system call.

4.5 Private Set Intersection

Freedman's PSI protocol (FNP-PSI) [15] forms the basis for the PROS protocols. Its main component is the *oblivious evaluation* of polynomials whose degree is equal to the size of the involved sets. A polynomial of degree n can be evaluated in $\Theta(n)$ using the *Horner Scheme* [11, p. 824]. Note that the evaluation has to be performed obliviously using the homomorphic operations of the Paillier cryptosystem. Using the homomorphic addition and multiplications by a constant, Horner's rule translates to $E\left(p(x_0)\right) = E\left(\sum_{i=0}^{n} a_i \cdot x_0^i\right) = E(a_0) \cdot (E(a_1) \cdot \ldots \cdot (E(a_{n-1}) \cdot E(a_n)^{x_0})^{x_0} \ldots)^{x_0}$

While Horner's scheme traditionally reduces the number of multiplications, the speed-up for the homomorphic operations is even greater since multiplications on plaintexts translate to (more expensive) exponentiations on ciphertexts.

In the implementation of the FNP protocol, the protocol obtains the basic polynomial root for each set element provided by the `set` class. *Alice* (in Freedman's protocol referred to as chooser C) multiplies all roots together using the functionality of the `polynomial` class such that they form the polynomial of Equation 1.

The coefficients of this polynomial are encrypted using Alice's public key. Then, they are sent from *Alice* to *Bob* (in Freedman's protocol referred to as server S) who uses Horner's rule to efficiently evaluate the polynomial in an oblivious fashion. The evaluation is repeated for all of Bob's set elements and the results are sent back to *Alice*. After decryption, *Alice* uses the `set` class to determine which of Bob's set elements are members of her own set.

Our PSI implementation is based on two new interface-defining virtual classes: `PSI` and *Private Matching* (`PM`). Concrete implementations of `PSI` provide two functionalities: one to `initialize` a set intersection and to learn the result and a second one to `respond` to a set intersection request. Each function takes the party's `set` as input. Our library uses *Standard Template Library* (STL) sets of `mpz_class` objects. PM was introduced into the library to handle degenerate sets of cardinality one. `PM` classes implement the same functions as `PSI` classes but take a single set element (`mpz_class`) as input.

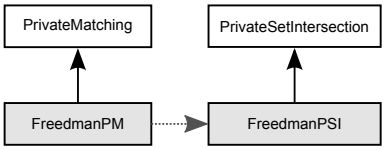


Figure 3: Class Hierarchy of the PSI and PM classes.

Freedman's PSI was implemented based on this design along with the corresponding `PM` implementation which transforms the integer inputs into sets of cardinality one and executes Freedman's PSI on those. Figure 3 illustrates the (simplified) class hierarchy. The PROS library will be extended by other PSI protocols in the future.

4.6 Privacy-Preserving Reconciliation of Ordered Sets

We have developed implementations for all three protocols, $3PR^C$*-FNP-implicit* and $3PR^C$*-PSI-explicit cached/non-cached* (see Section 3.4). All protocols are implemented as separate C++ classes, which makes the management of the network connection, the key management as well as the generation of the sets independent of the protocols themselves. Each protocol class provides member functions which execute the protocol as either *Alice* or *Bob*. A basic key exchange protocol was developed which enables the exchange of the public keys of the two parties (executing the PROS protocol) as well as populating each `crypto` class with the appropriate keys. The performance of all variants will later be analyzed in Section 6.

4.6.1 The $3PR^C$-FNP-implicit Implementation

$3PR^C$ and the preference order composition schemes were analyzed in detail and the parts which are independent of the composition scheme C were isolated and placed in a base class from which all $3PR^C$ protocols are derived. This setup allows for easy addition of new preference order composition schemes. The current implementation contains both composition schemes which were introduced in Section 3.4. The concrete protocol classes for *Sum of Ranks* and *Minimum of Ranks* provide the order in which the set elements are to be compared. The structure of implementing a preference order composition scheme consists of several loops which construct and evaluate polynomials of the correct set elements according to the specifics of the respective preference order composition scheme.[3]

4.6.2 The $3PR^C$-PSI-explicit Implementation

In order to facilitate that an underlying PSI protocol can be substituted without having to modify the generic code defining the order in which the PSIs are carried out, $3PR^C$*-PSI-explicit* was implemented based on the `PM` interface introduced in Section 4.5. Note, that only PM functionality is required to implement $3PR^C$*-PSI-explicit*.

This approach of directly executing one PM per comparison adds additional overhead compared to $3PR^C$*-FNP-implicit*. This is because each comparison causes the initi-

[3]It shall be noted that $3PR^C$*-FNP-implicit* does not directly call the Freedman protocol which is implemented as part of the PSI layer of the PROS library. Instead, it accesses the polynomial representation and the homomorphic cryptosystem directly.

ating party to send some data (i.e., a polynomial in the case of FNP's PSI) corresponding to its set element to the other party. In particular, this may result in the repeated sending of data for the same element of the ordered set. In order to remedy this shortcoming, a caching mechanism was introduced into the `PSI` and `PM` class designs. The implementation of the caching mechanism depends on the used PSI, since different data need to be stored for different PSI schemes. In the case of Freedman's PSI, this cache keeps track of which polynomials have been sent to or received from the other party. Received polynomials are stored locally thus avoiding repeated transmission. Generally, an identifier `id` uniquely identifies each cache entry. `Initialize` and `respond` functions take the `id` as an additional parameter. Before the actual set intersection is executed, `initialize` checks its cache. If the `id` is in the cache then the data is not sent again. Instead the function only receives the evaluation results from the other party, decrypts them, and returns the result. If the `id` is not in the cache, the data is created, encrypted, and sent as usual. Also, the new `id` is added to the cache. Similarly, the `respond` function checks if certain data was received in the past. If this is the case, the cached data is processed and the results are sent to the other party. If the `id` is not in the cache, the data is received as usual, added to the cache (with its `id`), and then processed.

The party calling `initialize` passes as `id` parameter the rank of its set element used in the current step. For the `respond` function, the caller passes the rank of the other party's element which is to be compared against in this step as `id`. This is necessary to determine if the other party's data corresponding to the element of current rank was received before. Since the preference order composition schemes clearly define which two elements in the respective sets are to be compared in each step of the reconciliation process, the information on `id` is readily available to both parties and the explicit use does not violate any privacy guarantees.

Whenever a concrete PSI, such as FNP, was used, we will substitute the *PSI* portion of the name (e.g., $3PR^C$-*FNP-explicit*).

5. appoint - A PROS APPLICATION

We have developed a proof-of-concept application *appoint* on top of the PROS library which allows the privacy-preserving negotiation of a meeting time while taking user preferences into account. In particular, the $3PR^C$-*FNP-implicit* protocol is being used to facilitate the negotiation. Since more and more people are using their mobile devices to handle their calendars, our application was implemented on Apple's iTouch platform.

Since the iTouch API is exclusively provided for Objective-C, applications for the iTouch are usually written in Objective-C as well. Our existing library, however, is C++ based. To combine these languages, the application as a whole was compiled as Objective-C++, which is a proper superset of C++, thus enabling the use of C++ code. Combining C++ and Objective-C introduces the additional challenge of interfacing two different class concepts.

In developing the app we have compiled the *GMP* library for the iTouch platform. Due to the modular design of our PROS library it was possible to leverage most of the code. The network component was identified as the only component that was platform dependent and required modification. Further, an iTouch GUI was added as part of the application to enable that a user user can transparently interact with the PROS library through a calendar-like interface. Since no modification to the rest of the library were required, the iTouch app can benefit directly from future improvements to the library. The existing C++ library interfaces with the new Objective-C code at two points: the network and the GUI. Both points will be discussed below. It is important to note, that our application runs on the genuine, i.e., non-*jail-breaked*, *iOS*.

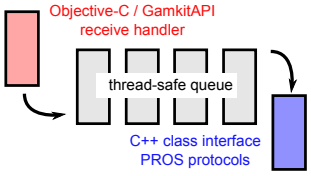


Figure 4: Flow of a received packet on the iTouch.

5.1 Bluetooth Network

One major design decision was the choice of the network technology. The iTouch provides Internet connectivity via 802.11b/g and GSM/UMTS as well as peer-to-peer connectivity over Bluetooth 2.1. When accessing the Internet via wireless LANs, a device usually is behind some network address translation (NAT) device and cannot be contacted directly by another party [21, Ch. 28]. A similar situation holds for the Internet access via GSM/UMTS. To enable communication of two mobile devices under these circumstances requires some form of relay server. The use of such a semi-trusted third party, which relays data while ensuring its integrity, would weaken the privacy guarantee of *appoint*. Furthermore, the use case of *appoint* suggests that both parties are in the same physical location (conferences, meetings, business dinners etc.). Therefore, the network component was implemented using peer-to-peer networking via Bluetooth.

The Bluetooth networking on the iTouch is abstracted through the *GameKit API*, which was originally built to support multiplayer games and therefore follows an event-driven programming paradigm. When using the GameKit API, incoming data packets are dispatched to a receive handler for processing. This asynchronous form of communication provides a challenge when executing a protocol with a well-defined order of execution, based on static send and receive calls. Our solution is built on a thread-safe queue which serves as a global data structure (cf. Figure 4). Data is pushed into the queue by the event-based receive handler and read-access is provided by a C++ class which corresponds to the network component of Figure 2.

5.2 Time-Slot Representation

In order to apply the PROS protocols to the *appoint* setting, it was necessary to represent the time-slots and the preferences associated with them as an ordered set. We encode each time-slot as a 32-bit integer $Z = S||D$ (set element) which is the concatenation of two integers S (s bits long) and D (d bits long) where $32 = s + d$. We choose $s = 16$ bits to uniquely specify the start time of the time-slot as the number of minutes since midnight of the day on which the reconciliation is performed. With this setup we can plan up to 45 days ahead. D is 16 bits long and encodes the duration of the event in hours. This allows the encoding of a duration from 1 up to 65,535 hours. The ordered set,

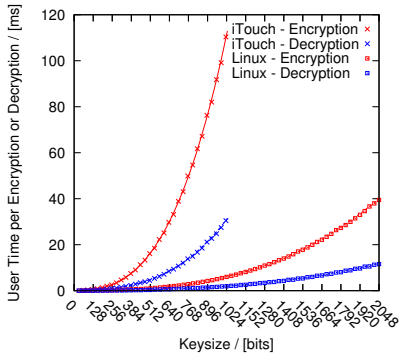


Figure 5: CPU time for encryption/decryption of 32-bit data.

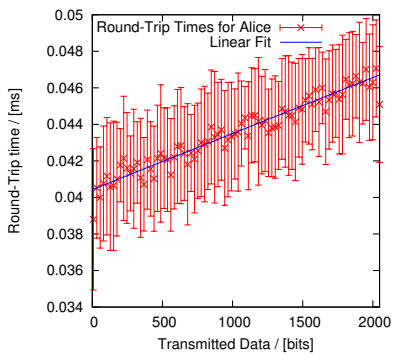


Figure 6: Network round-trip for data of reasonable size (Linux).

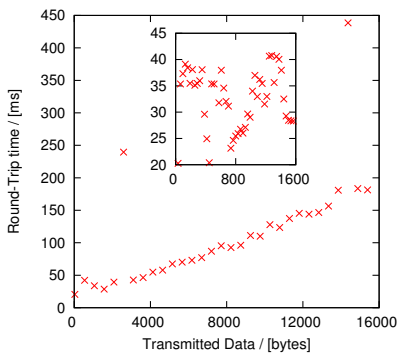


Figure 7: Network round-trip using Bluetooth on the iTouch.

which will be used by the PROS protocols, consists of these integers ordered by their preferences.

5.3 GUI Development

Interaction with smartphones using modern multi-touch displays is fundamentally different from that with desktop applications (e.g., [6, 26, 27]). In addition, the time an application remains open is very short. The user therefore expects a user interface (UI) which can be used quickly and intuitively.

We meet this challenge by not introducing new UI elements but instead using the ones provided natively through the *iOS SDK*. Furthermore, the look and feel of our application is built in the style of the native iTouch calendar application, thus simplifying the usage for iTouch users (see Appendix A for screenshots). For this, we leverage code provided by an open source project [1]. In addition, *appoint* uses the *Event Kit* framework, which was recently introduced in iOS 4, to access the local calendar on the iTouch and synchronize events with it.

When a user sets up his time-slots using the GUI, data is not directly passed to the protocols. Only once the negotiation is initiated, all time-slots are converted into integers with associated preferences as described in Section 5.2. The resulting ordered set is then passed to the PROS protocols. At the completion of the protocol, the one time-slot which maximizes the joint preference order is returned to the GUI. This value is then decoded and displayed to the user for acceptance and permanent storage in the calendar database.

6. PERFORMANCE EVALUATION

In the following, we provide a detailed analysis of our extensive performance tests. The tests were geared to gain a better understanding for the behavior of the protocols in practice and to explore their suitability for real-world applications. For this, we not only tested our Linux-based implementation of the PROS protocols and the FNP-PSI protocol, but also our proof-of-concept application *appoint*.

All Linux-based tests were performed on three dedicated machines having an identical configuration: AMD Athlon 64 Processor 3000+ at 1.8 GHz, 2 GB main memory, 64-bit Linux 2.6.32 (Ubuntu). For all experiments involving two parties the code for both parties was executed on the same host. While this does not represent the final setting in which the protocols might be utilized, it removes network latencies and simplifies the analysis and understanding of the data and the protocols. It is important to note that after one party has finished its computations and has sent off its data to the other party, it only waits to receive back results but does not perform any additional computation. Thus, the two processes do not compete for the CPU or cause timing conflicts.

appoint was tested using two iTouch devices. The iTouches have an ARM Cortex-A8 at 600MHz CPU, 256 MB of DRAM and are running iPhone OS 3.1.3. During the tests, each party was run on a separate iTouch device. The purpose of this section was not to compare the iTouch platform with the Linux platform but rather to show the general performance of the PROS protocols using Linux and the viability of mobile applications based on these protocols using a resource-limited platform such as the iTouch.

We start by presenting the results of testing the cryptographic operations and the network operations. Then, we present the performance results of the PROS protocols.

6.1 Cryptographic Performance

The performance of a cryptosystem is mainly influenced by two parameters: the key size and the length of the plaintext. The behavior of our implementation of the Paillier cryptosystem with respect to both was analyzed experimentally by measuring the CPU time. The time for each pair of parameters was averaged over 10,000 randomly generated inputs by measuring the time required to first encrypt all 10,000 plaintexts and then the time to decrypt all these ciphertexts. For the tests on the Linux platform, the cryptographic keys were varied between 32 bits and 2,048 bits in steps of 32 bits. In addition, for each key size, the plaintext length was varied from 8 bits to 256 bits in steps of 8 bits. On the iTouch platform, plaintexts of size 32 bits were investigated for keys ranging from 32 bits to 1,024 bits in steps of 4 bits. In the following, we present two cuts through the parameter space: varying the plaintext length for a fixed key size and vice versa. We directly compare the timings for the iTouch with those for the Linux platform to demonstrate the difference in computational power and to allow the reader to better relate the results presented below. Figure 5 shows how the timings for one encryption (decryption) changes with the key size for a fixed plaintext of 32 bits. Fitting both data sets for the Linux and iTouch platforms yields a cubic upper bound on the runtime. The time for one decryption is significantly smaller than the time for an encryption. This is due to the highly optimized implementation of the decryption algorithm using pre-computations and CRT. In addition, the operations performed during a

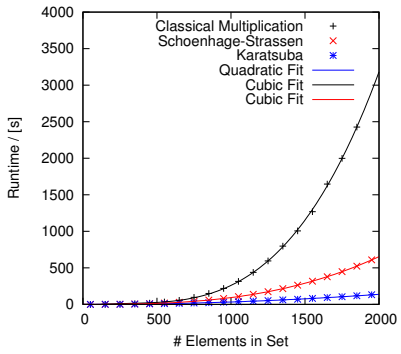
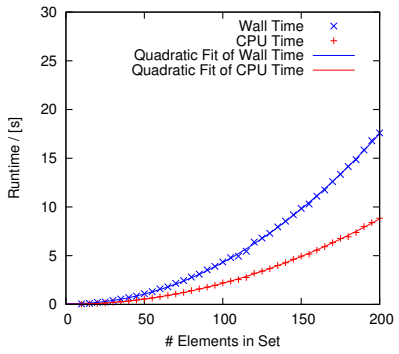
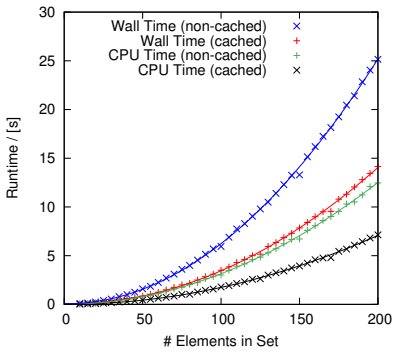


Figure 8: Wall time for the symmetric FNP-PSI protocol using 256-bit keys and 32-bit set elements.



(a) $3PR^C$*-FNP-implicit* $3PR^C$ implementation. Wall and CPU time are shown.



(b) $3PR^C$*-PSI-explicit* implementation. Wall and CPU time for the (non-) cached version are shown.

Figure 9: SOR worst-case runtime using 32-bit elements and 256-bit keys.

Paillier encryption are computationally more expensive than the operations as part of the decryption process. Due to a limit in space, the test results for varied plaintext sizes are presented in Appendix B.1.

6.2 Network Performance

To create a baseline for the network component used by the PROS and PSI protocols, the network implementation was tested for various sizes of the transmitted data. For the Linux platform, both the sender and the receiver were executed on the same machine. The data was transmitted internally using the loopback interface which almost completely eliminates network latency and thus allows an estimation of the overhead caused by the *send* and *receive* system calls.

Figure 6 shows the results for a series of tests. Each result was determined by first measuring the duration of 100,000 round-trip transmissions and then dividing the overall time by 100,000. This technique was necessary since the item for a single transmission is too short to be measured reliably. Each payload was randomly chosen and its size varied in steps of 8 bits between 8 bits and 2,048 bits. Since the data sent in each step of the protocol is bounded by the square of the modulus, the chosen range for the tests corresponds to reasonable key sizes. For each data size, the test was repeated 50 times and the average was plotted. The error bars show the standard deviation. The timing was performed using the wall time, i.e., the actual time which has passed—in contrast to CPU time, i.e., the time the process was actually allocated to the CPU. The broad scattering of the individual data can be explained by the process being preempted by the CPU during the measurement. The average behavior, however, is clearly linear. This was expected since—assuming constant overhead for executing the system call itself—doubling the data size results in double the transmission time.

Similar tests were performed for the Bluetooth network component for the iTouches. Figure 7 shows the results when varying the size of the data between 32 and 16,000 bytes in steps of 512 bytes averaging 100 round-trips. The inset shows an average of 32 ms per round-trip for data sizes which are commonly encountered during an execution of *appoint*.[4]

[4]To illustrate the general performance of the iTouch's Bluetooth performance, the range of tested data sizes was chosen wider deliberately.

6.3 PROS and PSI Protocol Performance

6.3.1 Set Generation

In order to allow for the testing of all $3PR^C$ protocols as well as the FNP-PSI protocol such that the parameters can be adjusted precisely, the sets must be prepared in a controlled manner. A C++ program was written to pre-generate suitable sets. Pre-generating sets also has the benefit that different protocols can be tested with the exact same input.

Note that for both preference order composition schemes the *worst-case* behavior of the respective reconciliation protocols is caused by a pair of sets containing a single matching element located at the end of each set [25]. The generating algorithm for the worst-case scenario ensures that the remaining elements in both sets are unique and do not match. Thus, the protocols find the single match only in the last step, resulting in the worst-case runtime of the protocols. Since FNP-PSI works in a single round which includes the comparison of all set elements, the number of matching elements is not expected to have an impact on the runtime of FNP-PSI.

To create a pair of sets for the *average case*, one has to make assumptions on what the average input for the protocols is. In order to perform tests which are as general as possible, but can also be translated to real-world inputs, the number of matching elements was specified as a fraction of the total number of elements in each set. For the performance evaluation it is assumed that all matching set elements are distributed equally at random across all possible preferences. This is achieved by adding all common elements to both sets, padding them using non-matching elements until the total number of elements is reached, and then randomly ordering both sets.

6.3.2 Symmetric FNP-PSI Protocol

In this paper we focused on symmetric settings in which both parties learn the result of the computation. We therefore analyzed the performance of executing FNP-PSI twice: once initiated by each party respectively. Recall that FNP-PSI creates a polynomial which contains all of Alice's set elements as roots. Bob evaluates this polynomial at all his

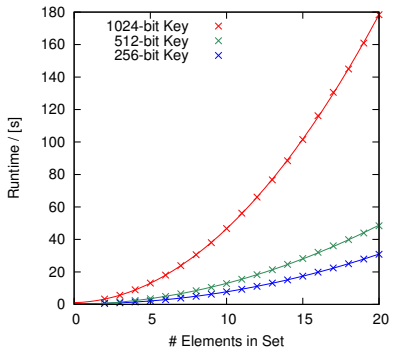


Figure 10: Runtime for the SOR scheme in the worst-case on the iTouch platform.

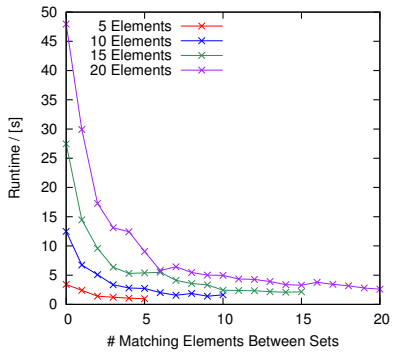


Figure 11: Runtime of the SOR scheme in the average case on the iTouch platform.

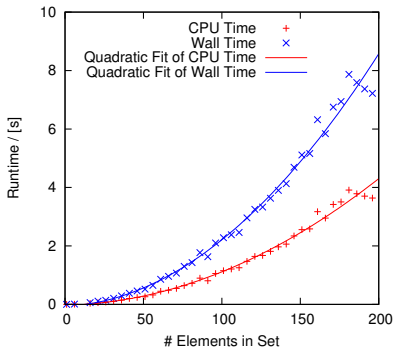


Figure 12: Runtime for the SOR scheme using $3PR^C$-*FNP-implicit* in the average-case (one match).

set elements and sends the results back. Since FNP-PSI computes the intersection of both sets and does not abort when a match is found, all steps need to be performed each time and the execution time is expected to be independent of the number and positions of the matching set elements.

The degree d of the polynomial introduced in Equation 1 grows linearly with the number of set elements resulting in a final degree of $O(n)$ [24]. Using classical multiplication (for polynomials), inserting an additional root requires $O(d)$ ring operations. Karatsuba's algorithm ($3d^{\log_2 3}$ operations) [33] and Schönhage and Strassen's FFT-based method ($O(d \log(d) \log\log(d))$ operations) [31,33] are two common and more efficient algorithms for multiplying polynomials. The runtime of FNP-PSI (as a function of the number of elements n) was analyzed using each multiplication algorithm. The tests were performed using the implementation provided by the FLINT library [3] for both advanced algorithms. For the tests, n was varied in steps of 100 between 50 elements and 2,000 elements averaging over 10 runs of the protocol. The key size was fixed at 256 bits and each set element had a length of 32 bits. Figure 8 shows the measured CPU time for one execution. While there is no significant difference in the runtime for small set sizes, the advantage of the advanced algorithms quickly becomes apparent for increasing n. For example for $n = 200$, using the advanced algorithms reduces the runtime by one third and for $n = 1,850$ the time required by Karatsuba's algorithm is only 5% of the classical implementation. In general, the complexity is reduced from $O(n^3)$ to $O(n^2)$.

6.3.3 Preference-Maximizing Protocols

Worst-Case.

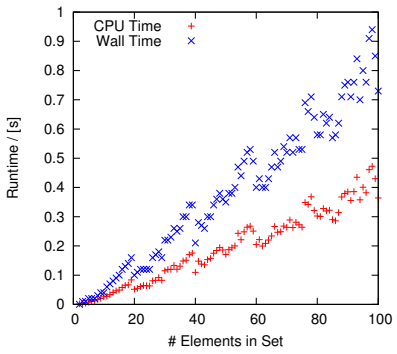
In [24], the worst-case performance of the $3PR^C$ protocol for both preference order composition schemes sum of ranks (SOR) and minimum of ranks (MOR) was theoretically determined as $O(n^2)$ given sets of size $O(n)$ as input. Figure 9 shows the experimental results for $3PR^C$-*FNP-implicit* and both $3PR^C$-*FNP-explicit* protocols using the SOR preference order composition scheme on the Linux platform. The graphs were obtained by varying the number of set elements, using the worst-case setting described above, in steps of 5 between 5 and 200 averaging over 20 executions of the protocol. The key size was fixed at 256 bits and each set element was 32 bits in length. For each run, both wall time and CPU time were considered. Note that the CPU time is consistently 50% of the wall time which indicates that the protocol is balanced amongst the parties. While one party is computing, the other one is waiting for the results. It is possible to fit the data points using a polynomial of degree two, which is in agreement with the quadratic behavior derived in [24]. Figure 9(a) shows the data for $3PR^C$-*FNP-implicit* and Figure 9(b) shows the $3PR^C$-*FNP-explicit* data. The positive effect of the caching mechanism is clearly visible. Both, wall time and CPU time are almost cut in half. The cached-variant of $3PR^C$-*FNP-explicit* is even more efficient than $3PR^C$-*FNP-implicit*. This is due to the fact that $3PR^C$-*FNP-implicit* uses the `vectorpolynomial` representation while $3PR^C$-*FNP-explicit* uses `flintpolynomial`. These results show, that the optimizations introduced by Meyer et al. are increasing the performance and that a naïve implementation as in $3PR^C$-*FNP-implicit-non-cached* suffers a significant performance loss. Using caching techniques, however, one can implement protocols that are both efficient and library.

appoint is solely based on $3PR^C$-*FNP-implicit*. Since the performance on the iTouches is significantly reduced with increasing key size, the worst-case tests were performed using 256-bit, 512-bit and 1,024-bit keys to investigate how the protocol performance develops. The number of set elements was varied between 2 and 20 in steps of 1. This corresponds to reasonable inputs for our application scenario, i.e., between two and twenty open time-slots. Figure 10 presents the results. For all key sizes the behavior is quadratic in the size of the set. However, as the tests of the cryptosystem indicated already, the performance is significantly reduced for large key sizes.

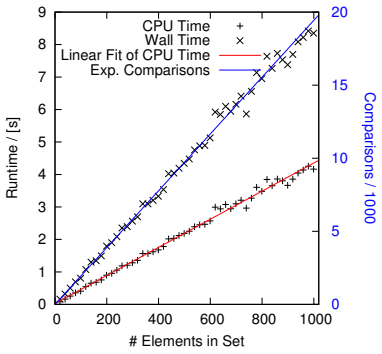
The results for the MOR preference order composition scheme are deferred to Appendix B.2.

Average-Case.

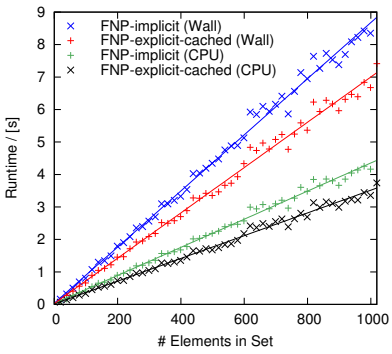
For the Linux platform, one set of experiments related to the average case was based on one matching element which was located at a random position in each ordered set. Given this random distribution, the average case is given by averaging the runtime over several independently chosen ordered sets. The cardinality was varied from 5 elements to 100 elements in steps of 5 elements and for each size the timings corresponding to 100 different ordered sets were averaged while using a fixed key. The resulting graph for $3PR^C$-*FNP-implicit* using the SOR scheme is shown in Figure 12. As in the worst-case scenario, the results show quadratic complexity with the CPU time being half of the wall time. However,



(a) Individual sections are visible using high resolution sample sizes.



(b) Long-range behavior. The expected number of comparisons is plotted on top of the CPU time (right axis).



(c) Long-range behavior using the largest point of each section using the $3PR^C$-*FNP-explicit* and $3PR^C$-*FNP-implicit* implementations.

Figure 13: Average-case runtime for the sum of ranks scheme using 32-bit set elements and 256-bit keys. Wall time and CPU time are shown. 100 runs were averaged and the fraction of matching elements is 5%.

the actual average runtime is about half of the one in the worst-case setting. This is due to the fact that a match, on average occurs after half the steps, i.e., when the elements in the middle of the ordered sets are being compared to each other.

Having only one matching set element is certainly a special case and thus additional experiments using multiple common elements were performed. Increasing the size of the ordered set while keeping the common elements constant does not constitute a good testing set. It is more representative to investigate how the runtime changes with ordered sets of different size where the sets have the same fraction of common elements. Figure 13(a) was obtained by creating ordered sets having 5% matching elements and varying their size n from 2 elements to 100 elements in steps of one element. The key size was kept constant at 256 bits and each set element had a length of 32 bits. For each data point, the timings corresponding to 100 different ordered sets were averaged while using a fixed key. Note that for $n < 20$ no single element matches since $0.05 \cdot 19 < 1$. Furthermore, an additional matching element is introduced for $n \in \{x \cdot 20 | x \in \mathbb{N}^+\}$. For the intervals in between, n increases but the number of matching elements remains constant. This structure is visible in Figure 13(a). The runtime drops down for $n = 20, 40, 60, \ldots$, i.e., whenever an additional matching element is introduced. The behavior in between appears quadratic for small n but cannot be clearly determined for the sections with larger n. This may be due to the fact that the number of possible combinations of matching elements increases with n and averaging 100 ordered sets may not be sufficient for reducing the error well-enough in order to obtain a clean graph.

For real-world applications, the overall behavior is often more important than the small-scale features described before. To remove the variations due to a new matching element, a set of tests was specifically crafted to only include the worst-case value of each section. For example, at $n = 20, 40, 60, \ldots$ a new matching element would be introduced and therefore ordered sets with $n = 19, 39, 59, \ldots$ were created. More precisely, the testing scenario varied n between 19 and 2,019 in steps of 20 and at each step 500 different ordered sets were considered to reduce the error in the resulting average value. The key size was kept at 256 bits and each set element was 32 bits long. The resulting graph in Figure 13(b) clearly shows that the runtime increases only linearly. This is a significant reduction over the worst-case behavior and constitutes a major result w.r.t. practical applications based on these protocols. It is possible to confirm this linear behavior by modeling the expected number of comparisons statistically, using the average-case input described above. The probability of finding a match after exactly i comparisons is given by $p(i)_{i>1} = \prod_{j=0}^{i-2} \frac{n^2-\alpha n-h}{n^2-j} \cdot \frac{\alpha n}{n^2-i+1}$ where α is the fraction of matching elements. Evaluating the expected value numerically yields the blue curve (right axis) in Figure 13(b). The plot shows that our experimental runtime is proportional to the theoretically expected number of comparisons.[5]

The same experiment was carried out for both $3PR^C$-*FNP-explicit* protocols. Figure 13(c) compares the $3PR^C$-*FNP-explicit-cached* with the $3PR^C$-*FNP-implicit*. The relation between the runtime of the two implementations is similar to what it is in the tests for the worst-case scenario: $3PR^C$-*FNP-explicit-cached* has a runtime which is slightly less than that of $3PR^C$-*FNP-implicit*.

For the iTouch platform, tests were performed using 512-bit keys for sets of size 5, 10, 15, 20. For each set size, the number of matching elements was varied from 1 to the number of elements in the set. The set generation process is similar to the one described above. However, because of the low number of elements in the sets, it was feasible to test each possible set configuration, instead of testing only those for a fixed fraction. For each data point, 50 different sets were averaged. Figure 11 shows the results for SOR. The runtime rapidly decreases when the number of matches increases. For 1 to 15 elements, the application is very practical with turn-around times of at most 27 seconds.

Appendix B.2 details the results for the MOR composition scheme, which are very similar to those of SOR discussed here.

[5]Due to space limitations, the Linux results for the MOR scheme in the average case are not included in this paper. However, it is important to note that the results are very similar to those for the SOR scheme.

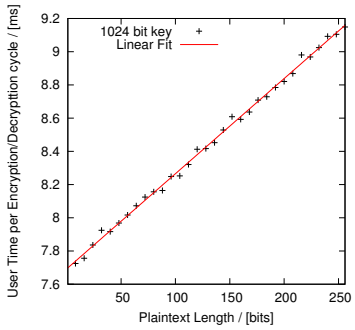


Figure 14: Time for encryption+decryption cycle as a function of plaintext size.

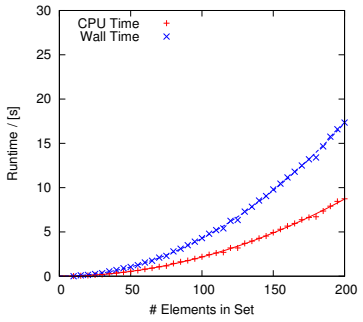


Figure 15: $3PR^C$-*FNP-implicit* $3PR^C$ implementation. MOR worst-case runtime using 32-bit set elements and 256-bit keys.

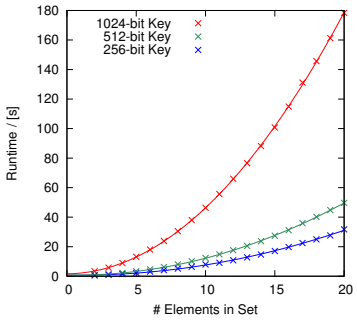


Figure 16: MOR worst-case runtime on the iTouch.

7. CURRENT AND FUTURE WORK

Directions for current and future work include the analysis of the protocols for LAN and WAN environments. In addition, we are investigating possible performance improvements for the encryption algorithm. We also plan to extend the library to include other homomorphic cryptosystems as well as leveraging our new, $3PR^C$-*PSI-explicit* implementation to evaluate the effect of using other privacy-preserving set intersection protocols.

Acknowledgment

In part, this work was supported by NSF Award CCF 1018616 and the UMIC Research Center, RWTH Aachen.

8. REFERENCES

[1] An Open-Source iPhone Calendar. `http://github.com/klazuka/Kal/`.

[2] Android API: java.security reference. `http://developer.android.com/reference/java/security/package-summary.html`.

[3] FLINT: Fast Library for Number Theory. `http://flintlib.org/`.

[4] GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`.

[5] iPhone API: Certificate, Key, and Trust Services API. `http://developer.apple.com/library/ios/#documentation/Security/Reference/certifkeytrustservices/Reference/`.

[6] iPhone Human Interface Guidelines. `http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html`.

[7] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A System for Secure Multi-Party Computation. In *15th ACM conference on Computer and communications security*, pages 257–266. ACM, 2008.

[8] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure Multiparty Computation Goes Live. In *Financial Crypto. and Data Security*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

[9] P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft. A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation. In *Financial Crypto. and Data Security*, volume 4107 of *LNCS*, pages 142–147. Springer, 2006.

[10] Y. Chen and W. Ku. Self-Encryption Scheme for Data Security in Mobile Devices. In *Consumer Comm. and Networking*, pages 850–854. IEEE, 2009.

[11] T. H. Cormen, C. E. Leiserson, R. R. L., and C. Stein. *Intro. to Algorithms*. MIT Press, third edition, 2010.

[12] E. D. Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-Preserving Policy-Based Information Transfer. In *Privacy Enhancing Technologies*, volume 5672 of *LNCS*, pages 164–184, 2009.

[13] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *Applied Cryptography and Network Security*, volume 5536 of *LNCS*, pages 125–142. Springer, 2009.

[14] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. In *Financial Cryptography'10*, volume 6052 of *LNCS*, pages 143–159, 2010.

[15] M. Freedman, K. Nissim, B. Pinkas, et al. Efficient Private Matching and Set Intersection. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004.

[16] P. Gasti and Y. Chen. Breaking and Fixing the Self Encryption Scheme for Data Security in Mobile Devices. In *Euromicro on Parallel, Distributed and Network-based Processing*, pages 624–630. IEEE, 2010.

[17] O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[18] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270–299, 1984.

[19] C. Hazay and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *Journal of Cryptology*, 23(3):1–35, 2008.

[20] C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC'10*, volume 6056 of *LNCS*, pages 312–331. Springer, 2010.

[21] C. M. Kozierok. *The TCP/IP Guide*. No Starch Press, 2005.

[22] Y. Lindell, B. Pinkas, and N. Smart. Implementing Two-Party Computation Efficiently With Security Against Malicious Adversaries. In *Security and*

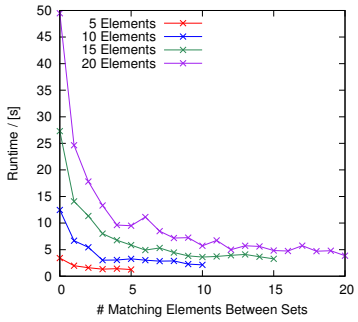


Figure 17: MOR average-case runtime on the iTouch.

Figure 18: Screenshots of the *appoint* GUI.

Cryptography for Networks, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[23] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—A Secure Two-Party Computation System. In *USENIX Security Symposium-Volume 13*, page 20. USENIX Association, 2004.

[24] U. Meyer, S. Wetzel, and S. Ioannidis. Distributed Privacy-Preserving Policy Reconciliation. In *IEEE International Conference on Communications, 2007. ICC'07*, pages 1342–1349. IEEE, 2007.

[25] U. Meyer, S. Wetzel, and S. Ioannidis. New Advances on Privacy-Preserving Policy Reconciliation. Cryptology ePrint Archive, Report 2010/064, 2010. `http://eprint.iacr.org/`.

[26] T. Moscovich. Multi-touch interaction. In *CHI '06: Human Factors in Computing Systems*, pages 1775–1778. ACM, 2006.

[27] C. Müller-Tomfelde, H. Benko, and D. Wigdor. Imprecision, Inaccuracy, and Frustration: The Tale of Touch Input. In *Tabletops*, Human-Computer Interaction Series, pages 249–275. Springer, 2010.

[28] P. Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[29] J. Pan, R. Jain, M. Bowman, X. Xu, S. Chen, and C. BUPT. Enhanced MILSA Architecture for Naming, Addressing, Routing and Security Issues in the Next Generation Internet. In *IEEE ICC*, 2009.

[30] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure Two-Party Computation is Practical. In *Advances in Cryptology–ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[31] A. Schönhage and V. Strassen. Schnelle Multiplikation Grosser Zahlen. *Computing*, 7(3):281–292, 1971.

[32] A. Seehra, J. Naous, M. Walfish, D. Mazieres, A. Nicolosi, and S. Shenker. A Policy Framework for the Future Internet. In *HotNets-VIII*. ACM, 2009.

[33] J. Von Zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2 edition, 2003.

[34] J. Voris, S. Ioannidis, S. Wetzel, and U. Meyer. Performance Evaluation of Privacy-preserving Policy Reconciliation Protocols. In *Policy'07*. IEEE, 2007.

[35] A. Yao. Protocols for Secure Computations. In *23rd Annual IEEE Symposium on Foundations of Computer Science*, volume 23, pages 160–164. IEEE, 1982.

APPENDIX

A. appoint GUI

Figure 18 presents some screenshots of the *appoint* GUI. The leftmost one shows the main calendar screen which lists already scheduled appointments. Right next to it, the interface to enter a time-slot is illustrated. In the third screenshot one can see the list of time-slots—ordered by preference. Finally, the rightmost screenshot shows the result of the reconciliation process.

B. ADDITIONAL PERFORMANCE DATA

B.1 Crypto Performance as Function of Plaintext Size

Figure 14 shows that keeping the key size constant at 1,024 bits and varying plaintext sizes from 8 to 256 bits yields a linear runtime behavior.

B.2 MOR Worst-Case Performance

Figures 15 and 16 show the worst-case results for the MOR scheme on the Linux platform and on the iTouch respectively. The Linux timings were obtained by varying the number of set elements in steps of 5 between 5 and 200 averaging over 20 different sets. The key size was fixed at 256 bits and each set element was 32 bits in length. For the iTouch, keys of sizes 256-bit, 512-bit, and 1,024-bit were used. The number of set elements was varied between 2 and 20 in steps of 1 averaging timings for 50 different sets.

The timings for the MOR scheme are almost identical to the ones for SOR. This is due to the fact that in the worst-case the same number of comparisons is required in both cases. The tests show that the larger number of rounds required by SOR does not significantly impact the performance.

B.3 MOR Average-Case Performance

In Figure 17 the average-case performance using the MOR scheme on the iTouch platform is shown. The tests were performed based on the same parameters as the SOR tests: using 512-bit keys for sets of size 5, 10, 15, and 20. For each set size, the number of matching elements was varied from 1 to the number of elements in the set. The behavior is very similar to Figure 11.