A preliminary version of this paper appeared in Theory of Cryptography Conference, TCC '05, Lecture Notes in Computer Science Vol. ???, Joe Kilian ed, Springer Verlag, 2005. This is the full version.

Secure Computation of the Mean and Related Statistics

Eike Kiltz[*] Gregor Leander[†] John Malone-Lee[‡]

Nov 2004

Abstract

In recent years there has been massive progress in the development of technologies for storing and processing of data. If statistical analysis could be applied to such data when it is distributed between several organisations, there could be huge benefits. Unfortunately, in many cases, for legal or commercial reasons, this is not possible.

The idea of using the theory of multi-party computation to analyse efficient algorithms for *privacy preserving data-mining* was proposed by Pinkas and Lindell. The point is that algorithms developed in this way can be used to overcome the apparent impasse described above: the owners of data can, in effect, pool their data while ensuring that privacy is maintained.

Motivated by this, we describe how to securely compute the mean of an attribute value in a database that is shared between two parties. We also demonstrate that existing solutions in the literature that could be used to do this leak information, therefore underlining the importance of applying rigorous theoretical analysis rather than settling for ad hoc techniques.

[*]Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: `ekiltz@cs.ucsd.edu`. URL: `http://kiltz.net`. Supported in part by a DAAD postdoc fellowship.

[†] Fakultät für Mathematik, Ruhr-Universität Bochum, 44780 Bochum, Germany. E-Mail: `gregor.leander@ruhr-uni-bochum.de`

[‡] University of Bristol, Department of Computer Science, Woodland Road, Bristol, BS8 1UB, UK. E-Mail: `malone@cs.bris.ac.uk`

Contents

1 Introduction

In recent years there has been massive progress in the development of technologies for networking, storage and data processing. Such progress has allowed the creation of enormous databases storing unprecedented quantities of information. This possibility to store and process huge quantities of information throws up the question of privacy. The need for privacy may be a legal requirement, in the UK for example there are strict rules for any party that holds information about individuals [24]. It may also be motivated by commercial interests: a pharmaceutical company does not want the results of its trials to become available while products are still being developed based on these results. On the other hand, if it were not for such privacy considerations, there could be significant mutual benefit in pooling data for research purposes, whether it be scientific, economic or market research. This apparent impasse is the motivating factor for our work.

We consider a situation in which there are two parties, each owning a database. Suppose that there is some attribute present in both databases. We propose a protocol which the two parties can use to evaluate the mean of this attribute value for the union of their databases. This is done in such a way that, at the end of the protocol, the two parties learn the mean and nothing else. No trusted party is required.

1.1 Related work

The problem that we have described above is a case of *secure two-party computation.* This notion was first investigated by Yao who proposed a general solution [28]. The two-party case was subsequently generalised to the multi-party case [3, 6, 18]. Although these solutions are general, they may not be terribly efficient when used with huge inputs and complex algorithms. We are therefore interested in a tailor-made protocol for the problem in question.

In [20, 21] Pinkas and Lindell analysed an algorithm for data-mining in the model for secure two-party computation. This work has stimulated research in the cryptography community into tools for working securely with large, distributed databases [1, 15].

Several algorithms that could be used for two parties to compute the mean of their combined data have already been proposed [9, 11, 12]. None of these solutions have been analysed in the formal model of security for two-party computation; moreover, in the appendices of this paper we demonstrate that they leak information. Similar weaknesses are also found in related protocols proposed elsewhere [7, 9, 10, 11, 12, 13, 25, 26].

1.2 Outline

The paper proceeds as follows. In Section 2 we discuss the notion of secure two-party computation that we will be working with. We describe how our protocol works in Section 3 and Section 4: in Section 3 we assume the existence of oracles to compute the low-level functions required by our protocol and in Section 4 we give secure implementations of these oracles. We conclude Section 4 by comparing of our protocol with Yao's general solution for two-party computation applied to computing the mean. In the appendices we discuss other solutions that have been proposed and show why they are insecure.

2 Secure two-party computation: definitions and results

We define secure two-party computation following Goldreich [17]. An equivalent model and results may be found in [4]. Henceforth, all two-party protocols will involve the two parties P1 and P2.

We will consider *semi-honest* adversaries. A semi-honest adversary is an adversary that follows the instructions defined by the protocol; however, it might try to use the information that it obtains during the execution of the protocol to learn something about the input of the other party. Using techniques such as the GMW compiler of Canetti et al. [5], a protocol that is secure against semi-honest adversaries can be made secure against adversaries that attempt to deviate from the protocol.

2.1 Definitions

Using the notation of Pinkas and Lindell [20, 21], let $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a function. Denote the first element of $f(x_1, x_2)$ by $f_1(x_1, x_2)$ and the second by $f_2(x_1, x_2)$. Let π be a two-party protocol for computing f. The views of $\mathsf{P1}$ and $\mathsf{P2}$ during an execution of $\pi(x_1, x_2)$, denoted $\mathsf{view}_1^\pi(x_1, x_2)$ and $\mathsf{view}_2^\pi(x_1, x_2)$ respectively, are

$$\begin{aligned}\mathsf{view}_1^\pi(x_1, x_2) &:= (x_1, r_1, m_{1,1}, \ldots, m_{1,t}) \text{ and}\\ \mathsf{view}_2^\pi(x_1, x_2) &:= (x_2, r_2, m_{2,1}, \ldots, m_{2,t})\end{aligned}$$

where r_i denotes Pi's random input, and $m_{i,j}$ denotes the j-th message received by Pi. The outputs $\mathsf{P1}$ and $\mathsf{P2}$ during an execution of $\pi(x_1, x_2)$ are denoted $\mathsf{output}_1^\pi(x_1, x_2)$ and $\mathsf{output}_2^\pi(x_1, x_2)$ respectively. We define

$$\mathsf{output}^\pi(x_1, x_2) := (\mathsf{output}_1^\pi(x_1, x_2), \mathsf{output}_2^\pi(x_1, x_2)).$$

Definition 2.1 [**Privacy w.r.t. semi-honest behaviour**] We say that π *privately computes* a function f if there exist probabilistic, polynomial-time algorithms S_1 and S_2 such that

$$\{S_1(x_1, f_1(x_1, x_2)), f(x_1, x_2)\} \equiv \{\mathsf{view}_1^\pi(x_1, x_2), \mathsf{output}^\pi(x_1, x_2)\}, \text{ and} \tag{1}$$

$$\{S_2(x_2, f_2(x_1, x_2)), f(x_1, x_2)\} \equiv \{\mathsf{view}_2^\pi(x_1, x_2), \mathsf{output}^\pi(x_1, x_2)\} \tag{2}$$

where $\equiv$ denotes computational indistinguishability. ▌

Equations (1) and (2) state that the view of the parties can be simulated given access to the party's input and output only. Recall that the adversary here is semi-honest and therefore the view is exactly according to the protocol definition. Note that it is not sufficient for simulator S_i to generate a string indistinguishable from $\mathsf{view}_i(x_1, x_2)$: the joint distribution of the simulator's output and the functionality output $f(x_1, x_2)$ must be indistinguishable from $\{\mathsf{view}_i^\pi(x_1, x_2), \mathsf{output}^\pi(x_1, x_2)\}$. This is necessary for probabilistic functionalities [17].

2.1.1 A simpler formulation for deterministic functionalities

In the case that the functionality f is deterministic, it suffices to require that simulator S_i generates the view of party Pi, without considering the joint distribution with the output. That is, we can require that there exist S_1 and S_2 such that

$$\{S_1(x_1, f_1(x_1, x_2))\} \equiv \{\mathsf{view}_1^\pi(x_1, x_2)\}, \text{ and} \tag{3}$$

$$\{S_2(x_2, f_2(x_1, x_2))\} \equiv \{\mathsf{view}_2^\pi(x_1, x_2)\}. \tag{4}$$

The reason that this suffices is that when f is deterministic, $\mathsf{output}^\pi(x_1, x_2)$ must equal $f(x_1, x_2)$. See [17] for a more complete discussion.

2.1.2 Private Approximations

If we privately compute an approximation of a function, we may reveal more information than we would by computing the function itself. To capture this we use the framework of Feigenbaum et al. [14] for private approximations. We restrict ourself to the case of deterministic functions f.

We say that $\hat{f}$ is an *ε-approximation of f* if, for all inputs (x_1, x_2),

$$|f(x_1, x_2) - \hat{f}(x_1, x_2)| < \varepsilon.$$

Definition 2.2 We say that $\hat{f}$ is *functionally private with respect to f* if there exist a probabilistic, polynomial-time algorithm S such that

$$S(f(x_1, x_2)) \equiv \hat{f}(x_1, x_2)$$

where $\equiv$ denotes computational indistinguishability. ▮

Definition 2.3 Let f be a deterministic function. We say that π *privately computes an ε-approximation of function f* if π privately computes a (possibly randomised) function $\hat{f}$ such that $\hat{f}$ is functionally private with respect to f and $\hat{f}$ is an ε-approximation of f. ▮

2.2 Secure composition of two-party protocols

Before stating the composition theorem that we will use, we first need to define two notions: *oracle-aided protocols* and *reducibility of protocols.*

Definition 2.4 [**Oracle-aided protocols**] An *oracle-aided protocol* is a protocol augmented by two things: (1) pairs of oracle-tapes, one for each party; and (2) oracle-call steps. An oracle-call step proceeds by one party sending a special *oracle-request message* to the other party. Such a message is typically sent after the party has written a string, its *query*, to its write-only oracle-tape. In response the other party writes its own query to its write-only oracle-tape and responds to the requesting party with an *oracle-call message.* At this point the oracle is invoked and the result is that a string is written onto the read-only oracle-tape of each party. Note that these strings may not be the same. This pair of strings is the *oracle answer.*

In an oracle-aided protocol, oracle-call steps are only ever made sequentially, never in parallel. ▮

Definition 2.5 [**Reducibility of protocols**]

- An oracle-aided protocol *uses the oracle-functionality f* if its oracle answers according to f. That is, when the oracle is invoked with requesting party query x_1 and responding party query x_2, the oracle answers $f_1(x_1, x_2)$ to the requesting party and $f_2(x_1, x_2)$ to the responding party.
- An oracle-aided protocol using the oracle functionality f is said to privately compute a function g if there exist polynomial-time algorithms S_1 and S_2 that satisfy (1) and (2) (from Definition 2.1) respectively, where the corresponding views of the oracle-aided protocol g are defined in the natural manner.
- An oracle-aided protocol privately reduces g to f if it privately computes g when using the oracle-functionality f. If this is so, we say that g is *privately reducible to f.*

▮

Theorem 2.6 [**Composition theorem [17]**] Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then, the protocol g' derived from g by replacing oracle calls to f with the protocol for computing f privately computes g.

Theorem 2.6 above will greatly simplify our analysis. It allows us to first describe and analyse an oracle-aided protocol in Section 3 before separately discussing how to implement the low-level details in Section 4.

3 An oracle-aided protocol for computing the mean

In this section we describe oracle-aided protocol for computing an approximation of the mean. We first describe the oracles that we will use in Section 3.2 before discussing the actual protocol in Section 3.3. The notation that we will be using in the remainder of the paper is described below.

3.1 Notation

Let a be a real number. We denote by $\lfloor a \rfloor$ the largest integer $b \leq a$, by $\lceil a \rceil$ the smallest integer $b \geq a$, and by $\lceil a \rfloor$ the largest integer $b \leq a + 1/2$. We denote by $\mathsf{trunc}(a)$ the integer b such that $b = \lceil a \rceil$ if $a < 0$ and $b = \lfloor a \rfloor$ if $a \geq 0$; that is, $\mathsf{trunc}(a)$ rounds a towards 0.

Let p be a positive integer. All arithmetic modulo p is done centred around 0; that is $c \bmod p = c - \lceil c/p \rfloor p$.

3.2 The oracles

Much of our computation is going to occur in a suitably large finite field. Henceforth we denote this field $\mathbb{F}_p$. We will elaborate on what "suitably large" means in Section 4 when we discuss how to implement the oracles used by our protocol.

We will make use of various oracles for two-out-of-two sharing. These are described below. In all the definitions we assume that $\mathsf{P1}$ and $\mathsf{P2}$ input y_1 and y_2 to the oracle respectively, and we let f be a function of (y_1, y_2).

Oracle for additive shares of $f(y_1, y_2)$ over $\mathbb{F}_p$: The oracle chooses s_1 at random from $\mathbb{F}_p$, sends s_1 to $\mathsf{P1}$, and sends $s_2 = f(y_1, y_2) - s_1$ to $\mathsf{P2}$. Players now hold s_1 and s_2 such that $s_1 + s_2 = f(y_1, y_2)$.

Oracle for multiplicative shares of $f(y_1, y_2)(\neq 0)$ over $\mathbb{F}_p^*$: The oracle chooses s_1 at random from $\mathbb{F}_p^*$, sends s_1 to $\mathsf{P1}$, and sends $s_2 = f(y_1, y_2)/s_1$ to $\mathsf{P2}$. Players now hold s_1 and s_2 such that $s_1 s_2 = f(y_1, y_2)$.

Oracle for additive shares of $f(y_1, y_2)$ over the integers: This definition requires a little more care. First we assume that $f(y_1, y_2) \in [-A, A]$ for some A and we let ρ be a security parameter. Now, the oracle chooses s_1 at random from $[-A2^\rho, A2^\rho]$, sends s_1 to $\mathsf{P1}$, and sends $s_2 = f(y_1, y_2) - s_1$ to $\mathsf{P2}$. Players now hold s_1 and s_2 such that $s_1 + s_2 = f(y_1, y_2)$.

Note 3.1 *By defining an appropriate f, the oracles above can be used to convert shares of one type to shares of another type.*

The primitive that we will make the most use of is *oblivious polynomial evaluation* (OPE).

Oracle for OPE: One of the players takes the role of the *sender* and inputs a polynomial P of (public) degree l over $\mathbb{F}_p$ to the oracle. The second player, the *receiver*, inputs $z \in \mathbb{F}_p$ to the oracle. The oracle responds to the receiver with $P(z)$. The sender requires no response.

The idea of OPE was first considered in [22] where an efficient solution requiring $O(l)$ exponentiations and with a communication cost of $O(l \log p)$ was proposed. This method uses a 1-out-of-N oblivious transfer (OT) from [23]. This OT protocol requires $O(1)$ exponentiations. We note that all exponentiations can be carried out over a different – potentially smaller – finite field.

3.3 The protocol

Suppose that $\mathsf{P1}$ has n_1 entries in its databases and $\mathsf{P2}$ has n_2. Denote these $\{x_{1,1}, x_{1,2}, \ldots, x_{1,n_1}\}$ and $\{x_{2,1}, x_{2,2}, \ldots, x_{2,n_2}\}$ respectively. Let

$$x_1 = \sum_{i=1}^{n_1} x_{1,i} \text{ and } x_2 = \sum_{i=1}^{n_2} x_{2,i}.$$

Without loss of generality we will assume that x_1 and x_2 are integers; appropriate scaling can be applied otherwise. Our problem becomes computing

$$M = \frac{x_1 + x_2}{n_1 + n_2}$$

where $\mathsf{P1}$ knows (x_1, n_1) and $\mathsf{P2}$ knows (x_2, n_2). We will assume that there are some publicly known values N_1 and N_2 such that

$$-2^{N_1} \leq x_1 + x_2 \leq 2^{N_1}, \; 0 < n_1 + n_2 < 2^{N_2}.$$

We describe an oracle-aided protocol for privately computing an approximation

$$\hat{M} \approx \frac{x_1 + x_2}{n_1 + n_2}. \tag{5}$$

By adding random noise we show how our protocol can be modified to privately approximate the mean M in the framework of Feigenbaum et al. [14].

Let m be the closest power of 2 to $n_1 + n_2$, that is

$$2^{m-1} + 2^{m-2} \leq n_1 + n_2 < 2^m + 2^{m-1} \text{ and } m \leq N_2.$$

Let m_1 and m_2 be defined analogously for n_1 and n_2 respectively. Let

$$k = \max\{m_1, m_2\} + 1 \in \{m, m+1\}. \tag{6}$$

With k thus defined we have

$$n_1 + n_2 = 2^k(1 - \varepsilon) \text{ where } -\frac{1}{2} < \varepsilon \leq \frac{5}{8}.$$

We can express

$$\frac{1}{n_1 + n_2} = \frac{1}{2^k(1-\varepsilon)} = \frac{1}{2^k}\left(\sum_{i=0}^{\infty} \varepsilon^i\right) = \frac{1}{2^k}\left(\sum_{i=0}^{d} \varepsilon^i\right) + \frac{1}{2^k} R_d$$

where $|R_d| < \frac{8}{3}(\frac{5}{8})^{d+1} < 2^{-\frac{2}{3}d+1}$.

It follows that

$$\frac{2^{N_2 d + k}}{n_1 + n_2} = \sum_{i=0}^{d} (2^{N_2}\varepsilon)^i 2^{N_2(d-i)} + 2^{N_2 d} R_d. \tag{7}$$

Let

$$Z = \sum_{i=0}^{d} (2^{N_2}\varepsilon)^i 2^{N_2(d-i)}. \tag{8}$$

We are almost ready to describe our protocol, first we define some polynomials that it will use. For $i = 2, \ldots, N_2 + 1$, let $P_i(X)$ be a degree $N_2 - 1$ polynomial such that, for $X \in \{2, \ldots, N_2 + 1\}$

$$P_i(X) = \begin{cases} 1 & : \quad X = i \\ 0 & : \quad \text{otherwise.} \end{cases}$$

With the definitions above we can now describe the protocol. In the description, when we say that P1 or P2 "inputs" something we mean that it inputs it to an oracle that computes the required functionality. Without loss of generality we can assume $x_1 + x_2 \neq 0$ (we do this to allow field multiplication); the case $x_1 + x_2 = 0$ can be handled as a special case.

Note 3.2 *We assume that $\mathbb{F}_p$ is sufficiently large that whenever a conversion from the integers to $\mathbb{F}_p$ is necessary, for an oracle or a player, it can be done in the obvious way without having to do any modular reduction. We will see what this means for the value of p in Section 4.*

Protocol 3.3 Oracle-aided protocol to compute a private 2^{-t}-approximation $\hat{M}$ of $M = \frac{x_1+x_2}{n_1+n_2}$

Set $d = \lceil \frac{3}{2}(t + N_1 + 2) \rceil$.

1. P1 and P2 input n_1 and n_2 respectively. Oracle returns additive shares a_1^F, a_2^F of Z over $\mathbb{F}_p$.
2. P1 and P2 input a_1^F and a_2^F respectively. Oracle returns additive shares a_1^I, a_2^I of Z over $\mathbb{Z}$.
3. For $j = 2, \ldots, N_2 + 1$:
   - P1 computes $b_{1,j} = \lfloor a_1^I / 2^j \rfloor$
   - P2 computes $b_{2,j} = \lfloor a_2^I / 2^j \rfloor$
4. Parties locally convert their shares back into additive $\mathbb{F}_p$ shares $c_{i,j}$, where share $c_{i,j}$ is the $\mathbb{F}_p$ equivalent of integer share $b_{i,j}$.
5. P1 and P2 input m_1 and m_2 respectively. Oracle returns additive shares d_1, d_2 of k over $\mathbb{F}_p$.
6. P1 chooses e_1 at random from $\mathbb{F}_p$ and defines the polynomial
$$R_1(X) = \sum_{i=2}^{N_2+1} c_{1,i} P_i(d_1 + X) - e_1.$$
P1 runs an OPE protocol with P2 so that P2 learns $e_2 = R_1(d_2)$ and P1 learns nothing.
7. P2 chooses f_2 at random from $\mathbb{F}_p$ and defines the polynomial
$$R_2(X) = \sum_{i=2}^{N_2+1} c_{2,i} P_i(d_2 + X) - f_2.$$
P2 runs an OPE protocol with P1 so that P1 learns $f_1 = R_2(d_1)$ and P2 learns nothing.
8. P1 and P2 input $e_1 + f_1$ and $e_2 + f_2$ respectively. Oracle returns multiplicative shares g_1, g_2 of $(e_1 + f_1) + (e_2 + f_2)$ over $\mathbb{F}_p$.
9. P1 inputs x_1 and P2 inputs x_2. Oracle returns multiplicative shares h_1, h_2 of $x_1 + x_2$ over $\mathbb{F}_p$.
10. P1 computes $\hat{M}_1 = g_1 h_1 \cdot 2^{-N_2 d}$ and sends it to P2.

11. P2 computes $\hat{M}_2 = g_2 h_2 \cdot 2^{-N_2 d}$ and sends it to P1.
12. P1 computes and outputs $\hat{M} = \hat{M}_1 \hat{M}_2$.
13. P2 computes and outputs $\hat{M} = \hat{M}_1 \hat{M}_2$.

Theorem 3.4 Protocol 3.3 correctly computes an 2^{-t}-approximation of $M = \frac{x_1+x_2}{n_1+n_2}$.

Proof: By (7), (8) and by definition of the oracles used in steps 1 and 2, after step 2 of the protocol, P1 and P2 hold a_1^I and a_2^I respectively such that

$$\left| \frac{2^{N_2 d + k}}{n_1 + n_2} - (a_1^I + a_2^I) \right| \leq 2^{N_2 d} R_d.$$

Once the local computation takes place in step 3, P1 and P2 hold $b_{1,k}$ and $b_{2,k}$ such that

$$\left| \frac{2^{N_2 d}}{n_1 + n_2} - (b_{1,k} + b_{2,k}) \right| \leq 2^{N_2 d - k} R_d + 2 \leq 2^{N_2 d} R_d + 2.$$

Using the bound on the error term R_d we get

$$\left| \frac{2^{N_2 d}}{n_1 + n_2} - (b_{1,k} + b_{2,k}) \right| \leq 2^{d(N_2 - \frac{2}{3}) + 2}. \tag{9}$$

However, the players cannot identify k.

By definition of the oracle invoked at step 5 and the construction of the polynomials R_1 and R_2, after step 7 P1 and P2 hold (e_1, f_1) and (e_2, f_2) respectively such that

$$e_1 + e_2 = c_{1,k} \text{ and } f_1 + f_2 = c_{2,k}.$$

Moreover, by (9) and the properties of the conversion used at step 4, when e_1, e_2, f_1 and f_2 are considered as integers we have

$$\left| \frac{2^{N_2 d}}{n_1 + n_2} - ((e_1 + f_1) + (e_2 + f_2)) \right| \leq 2^{d(N_2 - \frac{2}{3}) + 2}. \tag{10}$$

Once the final steps have been executed, by (10) and by definition of N_1, when $\hat{M}$ is treated as an integer we have

$$\left| \left(\frac{x_1 + x_2}{n_1 + n_2} \right) 2^{N_2 d} - \hat{M} 2^{N_2 d} \right| \leq 2^{d(N_2 - \frac{2}{3}) + N_1 + 2}.$$

Therefore, once the factor of $2^{N_2 d}$ is removed from $\hat{M}$, we obtain an approximation such that

$$\left| \left(\frac{x_1 + x_2}{n_1 + n_2} \right) - \hat{M} \right| \leq 2^{-\frac{2}{3} d + N_1 + 2} \leq 2^{-t}. \tag{11}$$

This completes the proof. ▌

Theorem 3.5 Protocol 3.3 is private.

Proof: To prove privacy we must define simulators S_1 and S_2 as in Definition 2.1. We describe simulator S_1 as a list of steps 1-13 to output what comes into view of P1 at the appropriate step of Protocol 3.3. The description of S_2 is similar.

$S_1(x_1, \hat{M})$

1. Choose a_1^F at random from $\mathbb{F}_p$.
2. Choose a_1^I at random from $[-A2^\rho, A2^\rho]$.
3. Do nothing - local computation going on.
4. Do nothing - local computation going on.
5. Choose d_1 at random from $\mathbb{F}_p$.
6. Do nothing - P1 learns nothing from an oracle for OPE.
7. Choose f_1 at random from $\mathbb{F}_p$ - P2 would choose f_2 at random and so f_1 has the correct distribution.
8. Choose g_1 at random from $\mathbb{F}_p^*$.
9. Choose h_1 at random from $\mathbb{F}_p^*$.
10. Do nothing - local computation going on.
11. Compute $\hat{M}_2 = \hat{M}/(g_1 h_1) \bmod p$.
12. Do nothing - local computation going on.
13. Output $\hat{M}$.

▮

3.3.1 Achieving functional privacy

We sketch how our protocol can be modified to achieve functionally privacy with respect to the mean function. This is done by adding random noise before outputting the approximation (see also [14]).

Assume that Protocol 4.1 is being used to compute a 2^{-2t}-approximation of the mean M. The modified protocol proceeds as before as far as step 9. In the new step 10, P1 inputs g_1h_1 and P2 inputs g_2h_2 to an oracle that returns additive shares of $g_1h_1g_2h_2$ over $\mathbb{Z}$. Each player performs division of the resulting shares by $2^{N_2 d}$ locally. The players now have additive shares of $\hat{M}$. Let us denote these $\hat{M}_1'$ and $\hat{M}_2'$. If the players output their shares at this point, the result would be identical to that for Protocol 3.3; however, before P1 and P2 output $\hat{M}_1'$ and $\hat{M}_2'$ respectively, the players individually add uniform random noise in the range $[-2^{-t}, 2^{-t}]$ to their shares. Only once this random noise is added do the players individually output their shares with precision 2^{-2t}. Adding these shares gives an approximation $\hat{M}'$ of the mean M.

It is easy to verify that the modified protocol computes a $2^{-2t} + 2^{-t+1}$-approximation $\hat{M}'$ of M. It remains to show that the $\hat{M}'$ computed in the modified protocol is functionally private with respect to M: we require a simulator as described in Definition 2.2. Suppose that a simulator given $M = \frac{x_1+x_2}{n_1+n_2}$ adds uniform random noise R_1 and R_2 in the range $[-2^{-t}, 2^{-t}]$ and outputs $S(M) = M + R_1 + R_2$ with precision 2^{-2t}. It can be readily checked that the statistical difference between $S(M)$ and $\hat{M}$ is about 2^{-t}. This implies that the function computed by the modified protocol is functionally private with respect to the mean. The properties of the modified protocol give us the following theorem.

Theorem 3.6 There exists a protocol that privately computes an approximation of the mean.

3.4 The variance and standard deviation

Using the notation of Section 3.3, let

$$\tilde{x}_1 = \sum_{i=1}^{n_1} x_{1,i}^2,\ \tilde{x}_2 = \sum_{i=1}^{n_2} x_{2,i}^2 \text{ and } \tilde{M} = \frac{\tilde{x}_1 + \tilde{x}_2}{n_1 + n_2}.$$

It is easy to use the techniques of Protocol 3.3 to compute the sample variance

$$\sigma^2 = \frac{1}{n_1 + n_2}\left(\sum_{i=1}^{n_1}(x_{1,i} - M)^2 + \sum_{i=1}^{n_2}(x_{2,i} - M)^2\right) = \frac{\tilde{x}_1 + \tilde{x}_2}{n_1 + n_2} - M^2$$

as follows. One first computes multiplicative shares of M by following Protocol 3.3 until P1 and P2 hold $\hat{M}_1$ and $\hat{M}_2$ respectively. These shares are squared locally to give P1 and P2 multiplicative shares of M^2 which are then then converted to additive shares a_1 and a_2. The next step is to apply Protocol 3.3 replacing x_1 and x_2 with $\tilde{x}_1$ and $\tilde{x}_2$ respectively until P1 and P2 hold multiplicative shares of $\tilde{M}$ which are then converted to additive shares b_1 and b_2. Now, $b_1 - a_1$ and $b_2 - a_2$ is an additive sharing of the variance as required. The standard deviation is obtained by taking the non-negative square root.

4 Implementing the oracles

Here we describe how to implement the various oracles used by Protocol 3.3. The security of the resulting construction then follows from Theorem 2.6. These protocols all use an oblivious polynomial evaluation protocol OPE, for example that proposed in [22] could be used.

4.1 Conversion protocols

Protocol 4.1 (ATM) can be used for converting additive to multiplicative shares over $\mathbb{F}_p$ as required in steps 8 and 9 of Protocol 3.3.

Protocol 4.1 ATM(a_1, a_2) where $a_1 + a_2 = x(\neq 0)$

---

1. P1 chooses m_1 at random from $\mathbb{F}_p^*$ and constructs the polynomial $P(X) = m_1^{-1}a_1 + m_1^{-1}X$.
2. P1 runs OPE with P2 so that P2 learns $m_2 = P(a_2) = m_1^{-1}a_1 + m_1^{-1}a_2 = m_1^{-1}x$.

At the end the parties hold multiplicative shares m_1, m_2 of x.

Protocol 4.2 (MTA) can be used for converting multiplicative to additive shares over $\mathbb{F}_p$. It will be necessary for Protocol 4.4 that we describe shortly.

Protocol 4.2 MTA(m_1, m_2) where $m_1 m_2 = x$

1. P1 chooses a_1 at random from $\mathbb{F}_p^*$ and constructs the polynomial $P(X) = -a_1 + m_1 X$.
2. P1 runs OPE with P2 so that P2 learns $a_2 = P(m_2) = -a_1 + m_1 m_2 = x - a_1$.

At the end the parties hold additive shares a_1, a_2 of x.

Step 2 of Protocol 3.3 require a protocol to convert additive shares from $\mathbb{F}_p$ into additive shares from the integers. This is not as straightforward as it sounds. Suppose that $z = z_1^F + z_2^F$ over $\mathbb{F}_p$. The corresponding equation over the integers is $z = z_1^F + z_2^F - lp$ for some l. We therefore need to know l in order to make the conversion.

Suppose that our parties have shares (z_1^F, z_2^F) over $\mathbb{F}_p$ where

$$-2^{n-1} < z = z_1^F + z_2^F \bmod p < 2^{n-1}$$

for some n. If $p > 2^{\rho+n+4}$, where ρ is a security parameter (see Section 4.3), the parties can use Protocol 4.3 to compute additive shares z_1^I, z_2^I of z over the integers. Protocol 4.3 is taken from [2] and specialised to the two-party case.

Protocol 4.3 FTI(z_1^F, z_2^F) where $z = z_1^F + z_2^F \bmod p$

Let $t = \rho + n + 2$. P1 and P2 execute the following steps.

1. P2 reveals $a_2 = \mathsf{trunc}\left(\frac{z_2^F}{2^t}\right)$ to P1.
2. P1 computes $l = \lceil \frac{z_1^F + 2^t a_2}{p} \rfloor$.
3. P1 chooses an integer b_1 at random from $[-p2^\rho, p2^\rho]$ and reveals $b_2 = 0 - b_1$ to P2.
4. P1 sets $z_1^I = z_1^F + b_1 - lp$.
5. P2 sets $z_2^I = z_2^F + b_2$.

At the end the parties hold additive shares z_1^I and z_2^I of z over the integers.

4.2 Other protocols

Step 1 of Protocol 3.3 requires a protocol for sharing Z additively over $\mathbb{F}_p$. Before describing a protocol to do this we give a protocol, Protocol 4.4, for sharing $2^{N_2 \varepsilon}$.

Protocol 4.4 Sharing of $2^{N_2}\varepsilon$

1. The parties run ATM(n_1, n_2) to obtain multiplicative shares a_1, a_2 of $n_1 + n_2$.
2. The parties run Protocol 4.6 followed by ATM to obtain multiplicative shares b_1, b_2 of 2^k.
3. P1 computes $c_1 = 2^{N_2} a_1 b_1^{-1} \bmod p$, P2 computes $c_2 = a_2 b_2^{-1} \bmod p$.
4. The parties run MTA(c_1, c_2) to obtain additive shares d_1, d_2 of $c_1 c_2 \bmod p$.
5. P1 computes $e_1 = d_1 - 2^{N_2-1} \bmod p$ and P2 computes $e_2 = d_2 - 2^{N_2-1} \bmod p$.

At the end the parties hold additive shares e_1, e_2 of

$$2^{N_2}\varepsilon = (n_1 + n_2)2^{N_2-k} - 2^{N_2}.$$

We now have Protocol 4.5 for computing additive shares of Z over $\mathbb{F}_p$.

Protocol 4.5 Sharing of Z

1. The parties run Protocol 4.4 to obtain additive shares a_1, a_2 of $2^{N_2}\varepsilon$.
2. P1 chooses b_1 at random and defines the polynomial

$$P(X) = \sum_{i=0}^{d} (a_1 + X)^i 2^{N_2(d-i)} - b_1.$$

3. P1 runs OPE with P2 so that P2 learns $b_2 = P(a_2)$ and P1 learns nothing.

At the end the Parties hold additive shares b_1, b_2 of Z.

At Step 5 of Protocol 3.3 we require a protocol for obtaining an additive sharing of k over $\mathbb{F}_p$. Protocol 4.6 below does this; it requires a polynomials Q_a and Q_b that we define first.

We have $m_1, m_2 \in \{1, \ldots, N_2\}$ and so $m_1 - m_2 \in \{-(N_2 - 1), \ldots, (N_2 - 1)\} = S$; there are $2N_2 - 1$ possibilities. Let $Q_a(X)$ and $Q_b(X)$ be the polynomials of degree $|S| - 1 = 2N_2 - 2$ such that for $s \in S$,

$$Q_a(s) = \begin{cases} 0 & : \text{ if } s < 0 \\ 1 & : \text{ if } s \geq 0 \end{cases} \quad \text{and} \quad Q_b(s) = \begin{cases} 0 & : \text{ if } s \leq 0 \\ 1 & : \text{ if } s > 0 \end{cases} \tag{12}$$

Protocol 4.6 Sharing of k and 2^k

1. P1 chooses a_1 at random from $\mathbb{F}_p$ and defines the polynomial
$$P_1(X) = Q_a(m_1 - X)2^{m_1+1} - a_1.$$
2. P1 runs OPE with P2 so that P2 learns $a_2 = P_1(m_2)$ and P1 learns nothing.
3. P2 chooses b_2 at random from $\mathbb{F}_p$ and defines the polynomial
$$P_2(X) = Q_b(m_2 - X)2^{m_2+1} - b_2$$
4. P2 runs OPE with P1 so that P1 learns $b_1 = P_2(m_1)$ and P2 learns nothing.

At the end the parties hold additive shares $a_1 + b_1,\ a_2 + b_2$ of 2^k. By replacing 2^{m_1+1} and 2^{m_2+1} with $m_1 + 1$ and $m_2 + 1$ respectively, the same technique can be used for sharing k.

It is straightforward to prove the correctness and the privacy of the protocols in this section in the same manner as Theorem 3.4 and Theorem 3.5 for Protocol 3.3. The security of the protocol derived from Protocol 3.3 by replacing each oracle call with the appropriate protocol from this section then follows from Theorem 2.6.

4.3 Complexity

All that remains is to analyse the complexity of the final protocol and discuss the security parameters.

Our protocol clearly runs in a constant number of communication rounds between the two parties. The complexity of our protocol depends chiefly on the accuracy of the result; this corresponds to the length d of the Taylor expansion. After execution of Protocol 3.3, by (11) both parties end up with a real number $\hat{M}$ such that
$$\left|\frac{x_1 + x_2}{n_1 + n_2} - \hat{M}\right| \leq 2^{-\frac{2}{3}d+N_1+2} \leq 2^{-t}. \tag{13}$$

Let us consider the size of the finite field $\mathbb{F}_p$. For Protocol 3.3, we have to choose p to be sufficiently large so that no unwanted wrap-around (modulo p) can occur. The value Z that is computed in the first step satisfies the bound
$$0 < Z \leq \sum_{i=0}^{d} \left((5/8)2^{N_2}\right)^i 2^{N_2(d-i)} = 2^{N_2 d} \sum_{i=0}^{d} (5/8)^i \leq 2^{N_2 d+2}.$$
Consequently for Protocol 4.3 (FTI protocol) to work we need a prime $p > 2^{N_2 d+\rho+8}$, where ρ is a security parameter, typically chosen as $\rho = 80$.

In steps 10 and 11 we have to ensure that the value of $g_1h_1g_2h_2 \approx 2^{N_2 d}(x_1 + x_2)/(n_1 + n_2)$ does not exceed p. Now, we have the bound $|g_1h_1g_2h_2| \leq 2^{N_2 d+N_1}$. From these bounds on Z and $g_1h_1g_2h_2$, we conclude that the prime p must satisfy
$$\log p > N_2 d + \max\{\rho + 8, N_1\}.$$

Note 4.7 *By an improved FTI protocol (using an implicit representation of l in terms of shares) we can improve the requirement made to the prime p to $\log p > N_2 d + N_1$. This does not affect the asymptotic running time of the FTI protocol.*

The complexity of the protocol is clearly dominated by two operations: (1) Protocol 4.5 to compute shares of Z using OPE with a polynomial of degree d; and (2) step 5 (implemented by Protocol 4.6), and steps 6 and 7 of Protocol 3.3 using OPE with polynomials of degree $2N_2-2$ and N_2-1 respectively.

Assume $N = N_1 + N_2$. Using the OPE protocol from [22] this makes a computation cost of $O(N_2+d) = O(N+t)$ exponentiations and a communication cost of $O((N_2+d)\log p) = O((N+t)^2 N)$.

4.4 Comparison with the generic solution

In [28] Yao presents a constant-round protocol for privately computing any probabilistic polynomial-time functionality. We compare the complexity of our solution with the one obtained by using Yao's generic solution.

Assume the given probabilistic polynomial-time functionality is given as a binary circuit with N inputs and G gates. Without going into details, Yao's protocol requires a communication of $O(G)$ times the length of the output of a pseudo-random function (which we denote by β and is typically 80 bits long). The main computational overhead of the protocol is the computation of the N oblivious transfers plus the application of G pseudorandom functions.

In our case set $N = N_1 + N_2$. A circuit computing an 2^{-t}-approximation of $M = \frac{x_1+x_2}{n_1+n_2}$ should compute the Taylor series, namely $d = O(t + N)$ multiplications in $\mathbb{F}_p$. Assuming multiplication requires circuits of quadratic size [1] we estimate the number of gates as $G = O((t + N)N^2)$. This results in a communication cost of $O(\beta(t+N)\log^2 p) = O((t+N)^3 N^2 \beta)$ which is larger than the cost of our protocol by a factor of $(t + N)N\beta$. On the other hand, the number of oblivious transfers (and so the number of exponentiations) for the generic protocol is $O(N)$. If we also take into account the $O((N + t)N^2)$ applications of the pseudorandom generator, the computation cost remains much the same in both cases.

For comparison, note that there is a computation–communication tradeoff for oblivious transfer suggested in [23]. This can reduce the number of exponentiations by a factor of c at the price of increasing the communication by a factor of 2^c.

We also note that, using our techniques, all application of Yao's circuits in the log protocol from [20, 21] can be abandoned. This essentially leads to a slight improvement – a multiplicative factor β – in the communication cost of [20, 21].

4.4.1 Acknowledgement

We thank the anonymous TCC referees for the helpful comments.

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k^{th}-ranked element. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 40–55. Springer-Verlan, 2004.

[2] J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer-Verlag, 2002.

[1] There exist circuits of size $O(\log p \log^2 \log p)$ for multiplication; however, assuming quadratic circuits seems reasonable for realistic values of $\log p$. We note that this convention is also made in [2, 20, 21].

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In 20^{th} *ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, 1988.

[4] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[5] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party computation. In 34^{th} *ACM Symposium on Theory of Computing*, pages 494–503. ACM Press, 2002.

[6] D. Chanm, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In 20^{th} *ACM Symposium on Theory of Computing*, pages 11–19. ACM Press, 1988.

[7] W. Du. *A Study of Several Specific Secure Two-party Computation Problems.* PhD thesis, Department of Computer Science, Purdue University, 2001.

[8] W. Du and J. Atallah. Privacy-preserving cooperative scientific computations. In 14^{th} *IEEE Computer Security Foundations Workshop*, pages 273–282, 2001.

[9] W. Du and M. J. Atallah. Privacy-preserving cooperative statistical analysis. In *Annual Computer Security Applications Conference ACSAC*, pages 102–110, 2001.

[10] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *New Security Paradigms Workshop*, pages 11–20, 2001.

[11] W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In 4^{th} *SIAM International Conference on Data Mining*, 2004.

[12] W. Du and Z. Zahn. Building decision tree classifier on private data. In *Workshop on Privacy, Security, and Data Mining at The 2002 IEEE International Conference on Data Mining (ICDM)*, pages 1–8, 2002.

[13] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *New Security Paradigms Workshop*, pages 127–135, 2002.

[14] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer-Verlan, 2001. Full version on Cryptology ePrint Archive, Report 2001/024.

[15] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlan, 2004.

[16] J. von zur Gathen and J. Gerhard. *Modern computer algebra.* Cambridge University Press, New York, 2 edition, 2003.

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

[18] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game: A completeness theorem for protocols with honest majority. In 19^{th} *ACM Symposium on Theory of Computing*, pages 218–229. ACM Press, 1997.

[19] O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In *Advances in Cryptology - CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer-Verlag, 1987.

[20] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO 2000*, volume 1800 of *Lecture Notes in Computer Science*, pages 35–24. Springer-Verlag, 2000.

[21] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):117–206, 2002.

[22] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In 31st *ACM Symposium on Theory of Computing*, pages 245–254. ACM Press, 1999. Full version available at `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`.

[23] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In 12th *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, 2001.

[24] UK Government. Data protection act 1998. Available at `http://www.hmso.gov.uk/acts/acts1998/19980029.htm`.

[25] J. Vaidya and C. Clifton. Privacy preserving naive bayes classifier for vertically partitioned data. In 4th *SIAM International Conference on Data Mining*, 2004.

[26] J. S. Vaidya. *Privacy Preserving Data Mining over Vertically Partitioned Data.* PhD thesis, Department of Computer Science, Purdue University, 2004.

[27] X. Wang and V. Y. Pan. Acceleration of Euclidean algorithm and rational number reconstruction. *SIAM Journal on Computing*, 32(2):548–556, 2003.

[28] A. C. Yao. How to generate and exchange secrets. In 27th *Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Sc. Pr., 1986.

A Flaws in existing protocols

In this Section we review some protocols in the area of private information retrieval appearing or used in [7, 9, 10, 11, 12, 13, 25, 26]. We identify two general design techniques in protocols that lead to insecure solutions. First, to hide an (integer) number, multiplying by a random integer and publishing the resulting product is a bad idea as we outlined in Subsection A.1. This kind of "hiding by multiplication" technique only makes sense over a finite field where multiplication by a random field element again results in a random field element.

Second, as a tradeoff between security and efficiency, it could be tempting to design protocols that leak some secret information. However, when revealing some information about a secret value, one has to take extreme care not to apply the protocol to another dependent instance (sequential composition) that may result in leaking more information than initially intended. This is discussed further in Section A.3.

A.1 The log protocol of Du and Zhan [12]

As noted in [25], a protocol for computing the logarithm of a shared value may be used for computing the mean (and vice-versa) since

$$\log(M) = \log\left(\frac{x_1 + x_2}{n_1 + n_2}\right) = \log(x_1 + x_2) - \log(n_1 - n_2).$$

However, to compute M with accuracy 2^{-t} we need a protocol that computes the logarithm with accuracy $2^{-N_2 t}$ (and vice-versa). So this solution cannot be considered to be practical. Furthermore, it is not a priori clear if computing the two approximations of $\log(M)$ still leads to a protocol secure in the Feigenbaum et al. sense of Definition 2.2; it may reveal more information than computing the mean itself.

In this subsection we review the (binary) *log* protocol of Du and Zhan (Section 5.2 in [12]) and show why it is insecure.

Suppose P1 holds an integer x_1 and P2 holds an integer x_2. Both want to compute real numbers y_1 and y_2 such that $y_1 + y_2 = \log(x_1 + x_2)$. The protocol is described below (all computations are done over the integers or reals).

Protocol A.1 *Protocol to compute* $\log(x_1 + x_2)$

---

1. P2 takes a random integer value r (chosen from a sufficiently large interval) and sets $y_2 = -\log(r)$.
2. P2 runs OPE with P1 so that P1 learns $c = r(x_1 + x_2)$ and P2 learns nothing.
3. P1 sets $y_1 = \log(c)$.

At the end of the protocol P1 holds y_1, P2 holds y_2 such that $y_1 + y_2 = \log(x_1 + x_2)$.

In this protocol P1 learns something about x_2: P1 knows all the divisors of x_1 and so if $a|x_1$ and $a \not| r(x_1 + x_2)$ then it knows that $a \not| x_2$.

We note that Protocols 6 and 7 from [13] suffer from the same problem as the above protocol and hence leak secret information.

A provably secure protocol for computing logarithms is given by Lindell and Pinkas in [21].

A.2 The mean protocol of Du and Atallah [9]

In this Subsection we review the mean protocol from [9] (see also [7]) and show that it leaks information.

Protocol A.2 *Protocol to compute the mean* $M = \frac{x_1+x_2}{n_1+n_2}$.

---

1. P1 generates two random integer numbers r and s (chosen from a sufficiently large interval).
2. P1 runs OPE with P2 twice. The first time P2 learns $a = r(x_1 + x_2)$ the second time it learns $b = s(n_1 + n_2)$. P1 learns nothing.
3. P1 sends $t = s/r$ to P2.
4. P2 computes $t \cdot \frac{a}{b} = \frac{x_1+x_2}{n_1+n_2}$ and sends it to P1.

Clearly the protocol suffers from the same problem as log protocol in Section A.1. Furthermore, we show that in some cases P1 can completely determine x_2 and n_2.

Suppose that the random numbers r and s in the first step are integers chosen from the interval $[0, 2^k - 1]$. (The distribution of r and s is not specified in [9]. To properly hide $x_1 + x_2$ and $n_1 + n_2$,

r and s have to be chosen uniformly at random from a large enough interval. We can assume that r and s are integer values, otherwise appropriate scaling can be applied.) To properly hide $x_1 + x_2$ and $n_1 + n_2$ in Step 2, we assume that $k \approx 2N$ where $N = \max\{N_1, N_2\}$. Suppose now that $r > s$. The problem comes in Step 3 where the rational number $t = s/r$ must be transfered to P2. Using the well known technique of rational number approximation, we show that under some choice of parameters, t leaks r and s – a complete break of the protocol.

Transferring a rational number t typically is done by using fixed point arithmetic. Let t' be the fixed point representation of t, so t' equals t up to a (public) number d of digits: $t' = u'/2^d$ for an integer $0 \leq u' < 2^d$. To guarantee P2 a sufficiently good approximation $t' \cdot \frac{a}{b}$ of $t \cdot \frac{a}{b} = \frac{x_1+x_2}{n_1+n_2}$ in the last step, we need $d > k+2N+1$. Now, P2 runs an algorithm to compute a rational number approximation of t', that is to find (if it exists) the unique pair of coprime integers $r' < s'$ in the interval $[0, 2^k - 1]$ such that

$$|u'/2^d - s'/r'| < 1/2^{2k+1}. \tag{14}$$

Algorithms for this with complexity $O(d \log^3 d)$ may be found in [16, 27] for example.

Assume that r and s are coprime (by a theorem of Dirichlet this happens with asymptotic probability $6/\pi^2$). Now, $|u'/2^d - s/r| < 1/2^d \leq 1/2^{2k+1}$. Hence there must exist two integers r' and s' satisfying (14). Since such integers are unique we must have $r' = r$ and $s' = s$. Of course, given r and s, P2 can compute x_1 and n_1 – a complete break of the protocol.

We note that the mean protocol from Abdallah and Du is used in several places [10, 25, 26] ([25, 26] also discuss alternative solutions).

A.3 The matrix multiplication protocol of Du, Han, and Chen [11]

Here we review the matrix multiplication protocol from [11] and indicate why certain applications of it leak information. Such an application is used in [11].

We start explaining a multiplication protocol (Protocol 2 of [11]). On input of two $n \times n$ matrices $A = (A_{ij})$ and $B = (B_{ij})$ (for what follows we do not need to further specify where the elements of A and B are chosen from) it outputs shares V_1 and V_2 such that $V_1 + V_2 = AB$.

We vertically divide the $n \times n$ matrix M into two equal-sized sub-matrices M_l and M_r of size $n \times n/2$ (we assume n is even); we horizontally divide $M^I = M^{-1}$ into two equal-sized sub-matrices M_t^I and M_b^I of size $n/2 \times n$.

Protocol A.3 *MUL(A,B)*

1. Both players generate a random, public, invertible $n \times n$ matrix M.
2. P1 computes $A_1 = AM_l$, $A_2 = AM_r$ and sends A_2 to P2.
3. P2 computes $B_1 = M_t^I B$, $B_2 = M_b^I B$ and sends B_2 to P1.
4. P1 computes $V_1 = A_2 B_2$ and P2 computes $V_2 = A_1 B_1$.

At the end of the protocol P1 holds V_1, P2 holds V_2 such that $V_1 + V_2 = AB$.

It is easy to see that the indicated protocol correctly computes shares V_1 and V_2 of the matrix product AB. However, as also noted in [11], the protocol leaks information to both players: P2, for example, learns the $n/2 \times n$ matrix $A_2 = AM_r$ in Step 2, where M_r is a public matrix it knows. Intuitively (if M is properly chosen) this provides some information about the secret matrix A. To be more precise, since M_r is a $n \times n/2$ matrix, for each fixed column j of the matrix A, $\mathsf{P2}$ gets a system of $n/2$ linear equations with n unknown variables A_{ij}, $1 \leq i \leq n$. In total $\mathsf{P2}$ gets $n^2/2$ linear equations with n^2 unknown variables A_{ij}, $1 \leq i, j \leq n$. We note that in [11] the matrices M in the first step are chosen according to a more complex distribution. However, our simplification does not affect what follows.

If only applying the protocol once this may not be a problem. However, when using this protocol more than once one has to be extremely careful: Any composition of this protocol applied to *dependent instances* may lead to a complete break of the protocol.

This problem occurs in Section 4.5 of [11] when computing the inverse of a matrix. Protocol MUL is applied to two dependent matrices: to a (random) matrix Q *and* to its inverse Q^{-1}. Unfortunately this inversion protocol is a crucial building block for all main results in [11].

We now describe the inversion protocol. The setting is that $\mathsf{P1}$ holds matrix A, $\mathsf{P2}$ holds matrix B, and both want to securely compute shares V_1 and V_2 such that $V_1 + V_2 = (A + B)^{-1}$. This is done in two main steps. In the first step $\mathsf{P2}$ generates two random, invertible matrices, P and Q, and player 1 computes $C = P(A + B)Q$ using the multiplication protocol. $\mathsf{P2}$ knows nothing about C. In a second step $\mathsf{P1}$ locally computes $C^{-1} = Q^{-1}(A + B)^{-1}P^{-1}$ and both players run a protocol to compute matrices V_1 and V_2 such that $V_1 + V_2 = QC^{-1}P = (A + B)^{-1}$. We will show that after the execution of the protocol, $\mathsf{P1}$ can learn P and Q.

In the first step $\mathsf{P2}$ must reveal some information about Q, namely in the MUL protocol $\mathsf{P1}$ learns $M_b^I Q$ for some public $n/2 \times n$ matrix M_b^I.

In the second step (as an intermediate step), $\mathsf{P2}$ has to create shares of $Q = Q_1 + Q_2$ and send Q_1 to $\mathsf{P1}$. $\mathsf{P1}$ has to create shares of $C^{-1} = C_1 + C_2$ and send C_2 to $\mathsf{P2}$. Now both run a multiplication protocol to get shares W_1 and W_2 such that $W_1 + W_2 = C^{-1}Q = (C_1 + C_2)(Q_1 + Q_2) = C_1Q_1 + C_1Q_2 + C_2Q_2 + C_2Q_1$. To compute the shares W_1 and W_2, the players have to run the protocol MUL twice: on inputs (C_1, Q_2) and on inputs (C_2, Q_2). But at the point where the players run the multiplication protocol MUL on inputs (C_1, Q_2), $\mathsf{P1}$ learns $Q_2\hat{M}_r = (Q^{-1} - Q_1)\hat{M}_r$, where the $n \times n/2$ matrix $\hat{M}_r$ and the $n \times n$ matrix Q_1 are known to $\mathsf{P1}$.

So far in this inversion protocol $\mathsf{P1}$ has learnt

$$S := M_b^I Q \text{ and} \tag{15}$$

$$T := (Q^{-1} - Q_1)\hat{M}_r + Q_1\hat{M}_r = Q^{-1}\hat{M}_r \tag{16}$$

for a known $n \times n/2$ matrix M_b^I, and for a known $n/2 \times n$ matrix $\hat{M}_r$.

Let $Q = (Q_{ij})$ and $Q^{-1} = (Q_{ij}^{-1})$. Equation (15) provides $n^2/2$ linear equations in the unknown Q_{ij}, (16) provides $n^2/2$ linear equations in the unknown Q_{ij}^{-1}. Since Q^{-1} depends on Q, combining (15) and (16) we get n^2 (not necessarily linear) equations in the unknown Q_{ij}. If M and $\hat{M}$ where chosen at random, with high probability the equations are independent and hence knowledge of S and T provides enough information for $\mathsf{P1}$ to compute Q. For small n, the matrix Q can be computed efficiently. With a similar argument it also learns P, which enables him to compute $\mathsf{P2}$'s input B and hence to completely break the protocol.

To save the MUL protocol from [11], one could argue that when the matrix $\hat{M}$ is chosen properly (depending on Q, Q^{-1} and M), then one may hope that the resulting linear equations obtained by $\mathsf{P1}$ can be made dependent on the previous ones such that no new information about Q is released to $\mathsf{P1}$. However, in [11] the protocols using MUL or the inversion protocol as a sub-protocol become very complex (as do the dependencies) and so this is very likely to become impractical.

We note that the main results of [11] still hold when one replaces the MUL protocol by one that is provably secure. We suspect that the protocol proposed by Atallah and Du [8] (which itself builds on an idea by Goldreich and Vainish [19]) can be proved secure.