Journal of Official Statistics, Vol. 27, No. 4, 2011, pp. 669–691

# Secure Multiple Linear Regression Based on Homomorphic Encryption

*Rob Hall*[1]*, Stephen E. Fienberg*[1]*, and Yuval Nardi*[2]

We consider the problem of linear regression where the data are split up and held by different parties. We conceptualize the existence of a single combined database containing all of the information for the individuals in the separate databases and for the union of the variables. We propose an approach that gives full statistical calculation on this combined database without actually combining information sources. We focus on computing linear regression and ridge regression estimates, as well as certain goodness of fit statistics. We make use of homomorphic encryption in constructing a protocol for regression analysis which adheres to the definitions of security laid out in the cryptography literature. Our approach provides only the final result of the calculations, in contrast with other methods that share intermediate values and thus present an opportunity for compromise of privacy. We perform an experiment on a dataset extracted from the Current Population Survey, with 51,016 cases and 22 covariates, to show that our approach is practical for moderate-sized problems.

*Key words:* Combining data sources; confidentiality; homomorphic encryption; privacy-preserving statistical calculation; secure multi-party computation.

## 1. Introduction

Preserving the confidentiality of individually identifiable information in statistical databases has a long statistical tradition, although within the world of official statistics efforts to do so have focused on perturbing the data through some form of data masking in order to allow for the calculation of simple summary statistics rather than on accurate statistical inference for more elaborate statistical models. There is an active debate in the statistical literature on how to best achieve confidentiality while also allowing for useful statistical analyses; see e.g., Duncan et al. (2001).

Here we consider the problem of multiple regression calculations where the data are divided among several parties, each of whom is unwilling to reveal their data. This problem occurs for example where the data consist of health insurance billings and records, and the parties are health insurance agencies. In this case, there are legal barriers which prevent the release of data, however a regression performed on the union of the parties' data may have better properties than a regression on either (incomplete) data set. Similar problems arise when parties have done surveys on the same set of individuals at

[1] Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, U.S.A. Emails: rjhall@cs.cmu.edu and fienberg@stat.cmu.edu
[2] Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel, Email: ynardi@ie.technion.ac.il
**Acknowledgments:** This research was partially supported by Army contract DAAD19-02-1-3-0389 to Cylab, and NSF Grant BCS0941518 to the Department of Statistics, both at Carnegie Mellon University.

different times, and wish to implement regressions on the union of variables in the multiple surveys, but are unwilling to share the data.

It has long been argued that a superior way to protect the confidentiality of statistical data is to restrict access to those data and to simply provide results from statistical modeling such as regression coefficients and standard errors.[3] For recent examples of this line of work which also include a focus on regression diagnostics, see O'Keefe and Good (2007; 2008; 2009) and Reiter (2003). If we deem the regression coefficients to be "too sensitive to release" then we can also consider output perturbation in the spirit of the recent cryptographic literature on $\epsilon$-differential privacy; see e.g., Dwork (2008) and Nissim (2008). By and large, the cited literature has been focused more on privacy protection and less on the utility and accuracy of the resulting statistical releases, or it treats the resulting analysis problem far more generically as one of learning from noisy data; see e.g., Chen et al. (2009).

Other approaches include the use of data masking such as the addition of normally distributed noise to the underlying data; cf. Duncan and Pearson (1991) and the recent review by Duncan and Stokes (2009). This leads, in the case of regression analysis, to inference using measurement error models that have a long history in statistics, eg., see Fuller (1993). Duncan et al. (2001a, b) and Trottini et al. (2004) discuss the risk-utility tradeoffs associated with such an approach. We contrast this focused statistical use of masked data with suggestion in the datamining literature of a fundamentally different sort, e.g., Du et al. (2004). As an alternative to the traditional additive noise approach to data masking, Ting et al. (2008) suggest using a more subtle perturbation method for protecting confidential continuous microdata—Random Orthogonal Matrix Masking (ROMM)—which preserves the sufficient statistics for multivariate normal distributions, and thus is statistically defensible, but serves the function of more traditional data masking, and they illustrate ROMM in the context of multiple regression analysis.

In this article we consider a related but different problem in privacy protection, associated with statistical calculations across multiple databases, studied in the context of regression analysis by Karr et al. (2005; 2006; 2009). In our setting, there are two distinct issues of privacy—that of the individuals whose data are in the different data bases and that of the owners of the databases. For the former, if data are merged across sources, privacy protection for the linked individual files goes well beyond the privacy issues that are considered with regard to the data within the individual sources. Even if there were not an issue associated with individual privacy protection, the database owners might not wish or be able to share their data with others directly.

There has been much recent work on the problem of securely estimating regression coefficients. A survey of earlier techniques is to be found in Vaidya et al. (2005) (Chapter 5). Those techniques hinge on so-called secure matrix multiplication subprotocols. Essentially since estimating regression coefficients boils down to matrix products, a secure regression protocol can be made by composing secure matrix products. However the protocols they give for computing the matrix products have certain drawbacks which may limit their practicality. First, some rely on the existence of a trusted third party (or at

[3] For example, see http://www.ssa.gov/policy/docs/rsnotes/rsn2009-01.html and http://hrsonline.isr.umich.edu/index.php?p=resappguide

least a partially trusted third party). Such a trusted party may not exist in practice. In fact the field of cryptographic protocols (Goldreich 2004) was born to get around this difficulty. The other matrix product protocols they give do not meet the requirements for security under the stringent but well-accepted definitions from the cryptography literature, which we explain below in Section 2. These authors achieve a much weaker definition of security which may imply that more private information is being leaked than is absolutely necessary. We note, however, that all of the protocols they present are easy to implement and efficient to run. Therefore such protocols are advantageous if maintaining strict security is not part of the goal.

The method for secure regression due to Karr et al. (2005; 2006; 2009) is based similarly on a method for secure matrix products. These authors stop, however, after securely computing the full data covariance matrix, at which point it is shared by all (as is the response vector). While this makes the protocol computationally very attractive, and allows parties to locally compute a wide variety of diagnostics, it also presents a compromise with respect to data privacy. Revealing the data covariance matrix is not necessary for computing the coefficient vector, and hence causes the leakage of more private information than is strictly necessary. As an example, we explore the unavoidable leakage due to computing the coefficients in Appendix A and find that in general the data covariance matrix is not leaked.

Previous methods for privacy preserving data mining have focused on cases where the data are split between the parties in a certain regular fashion. The two most common patterns are "horizontal partitioning" where each party has a subset of the cases, and "vertical partitioning" where each party has a subset of the features. We propose a protocol that will work irrespective of the partitioning scheme, and may be used in the general case wherever parties hold interlocking parts of the database. This is the situation which may occur when the parties are data warehouses. Note that although our protocol is general and works for any partitioning scheme, certain regular structures will allow for further optimizations, but we will not discuss them.

In Section 2, we outline the privacy guarantees that our protocol gives, and then, in Sections 3 and 4, we describe the protocol for two parties before showing, in Section 5, how to extend our approach to multiple parties. We then perform a simulation of the protocol (and make the code available) to show that it is fast enough to be useful in practice. We conclude by considering the possible invasion of privacy which occurs from simply having the regression coefficients, from the perspective of statistical disclosure limitation.

## 2. Setting

We consider a setting where $K \geq 2$ parties each hold a part of the design matrix $X \in \mathbb{R}^{n \times p}$ and the response vector $y \in \mathbb{R}^n$, and they are interested in analyzing the statistical regression model:

$$y = X\beta + \epsilon. \tag{1}$$

Their goal is to compute the estimated regression coefficient vector or estimated ridge regression vector:

$$\hat{\beta} = (X^T X)^{-1} X^T y, \tag{2}$$

$$\hat{\beta}_\lambda = (X^T X + \lambda I)^{-1} X^T y. \tag{3}$$

We first assume that there is no missing data, so that each entry in $X$ and $y$ is known to exactly one party. If there is overlapping data then the parties must either agree to use one party's data, or securely compute a weighted average (e.g., as a measurement error). We proceed by assuming the former. We also assume that each case has a unique identifier which is known to every party. Obtaining such identifiers when they do not naturally exist (e.g., social security numbers or ID numbers) is the problem of record linkage, and is given a secure treatment in, e.g., Scannapieco et al. (2007). Under these assumptions, each party may take whatever data is known to him, and assemble it into a design matrix and response vector with missing values (where the missing values are the data belonging to the other parties). Denote the input held by party $i$ as $X_i \in \mathbb{R}^{n \times p}$, $y_i \in \mathbb{R}^n$. We suppose that each party replaces those elements unknown to him by zeroes. Then if the matrices were "superimposed" (i.e., by summing them), we would obtain the full design matrix and response vector. We make no further assumptions about the structure of the data. Note that this is a relaxation of the requirements for the protocol of Karr et al. (2009), which works when the parties each hold complete columns of $X$.

We build up a *protocol* (see, e.g., Goldreich 2004) for computing (2) and (3). The protocol is sequence of steps which consist of parties doing local computations, and transmitting messages to other parties. We aim for the cryptographic definition of security under a "semi-honest" model. This model of security assumes that each party will follow the protocol and use their true input values, but will also be curious about the other parties' secret inputs. A protocol is secure so long as the messages received by the parties during execution of the protocol do not leak information about the secret inputs which belong to each party.

The requirement for security in this setting is that the transcript of messages received by a party can be "simulated" based on nothing more than the input known to that party, and the output of the protocol. Formally, this requires a probabilistic polynomial time algorithm (the simulator) which takes the input of a party, the output, and the random seed that the party uses, and outputs a transcript of messages which is *computationally indistinguishable* from a transcript generated during a run of the protocol. Computational indistinguishability is defined formally in Goldreich (2004; 1998) but for our purposes we note the following ways to achieve the requirement:

- Messages which are encryptions due to a *semantically secure* (see e.g., Goldreich 2004) public key encryption scheme may be simulated by encrypting any arbitrary value with the same public key.
- Random samples from a distribution which depends only on the input and output may be simulated by drawing from that distribution (so long as the random number generator produces draws computationally indistinguishable from the distribution).
- Random samples which depend on a secret value may be simulated so long as the distribution is sufficiently close (e.g., in variation distance) to one with no dependence on the secret value. Here sufficiently close means that the variation

distance between the distributions is a negligible function of some *security parameter* (Goldreich 2004).

Intuitively, if the messages can be simulated in such a manner, then they can reveal no information about the inputs belonging to other parties *beyond that revealed by the output.* An example of a protocol which does not achieve this definition of security is one where all parties send their data to Party 1, who computes the estimate locally on the combined data and then sends it back to all other parties. In this case the messages received by Party 1 consist of the data of other parties, and in general it is impossible to simulate these messages given only the input and output belonging to Party 1. Likewise the protocol of Karr et al. (2009) does not achieve this definition of security, but since the full data covariance matrix is shared between all parties, this reveals more information than just the estimate $\hat{\beta}$.

This privacy definition makes no guarantees when faced with a party who is malicious and is willing to deliberately deviate from the protocol in an attempt to learn the other parties' secret inputs. We note however that a protocol which is secure in this semi-honest model may be "compiled" into a protocol secure against malicious parties (Goldreich 2004) but may be too inefficient for practical use.

A final point is that this security model simply ensures that the computation of the estimate does not reveal more information than the parameter itself. As Lindel and Pinkas (2002) point out, it makes no statement about whether the estimate should be computed in the first place. For example if there are two parties, one of whom holds a single case and another who holds many cases, the latter may be able to compare the parameter output by the protocol to a parameter he computes locally on his data only. This may reveal information about the case held by the first party. In this article, we presume that the parties have decided that the benefit of knowing $\hat{\beta}$ outweighs whatever private information it leaks. We show in Appendix A that under the two most common data partitioning (vertical and horizontal partitioning), it is rather difficult for either party to learn anything about the other party's data.

We can also transform the estimated parameter vector using a differential-privacy technique such as that presented by Chaudhuri and Monteleoni (2008) during the protocol, if the parties deem such protection necessary. This is a relatively straightforward extension of the methodology we describe in this article.

## 3. A Two-Party Protocol for Computing Sums and Products

One of the earliest results in the field of cryptographic protocols, due to Yao (1982), is that any computation which can be encoded as a Boolean circuit (i.e., a function on the field $GF(2)$) may be computed securely by a reduction to "oblivious transfer" (see Goldreich (2004) for more details). Therefore one way to construct a protocol for (2) would be to construct a huge circuit to compute all the sums and products and the matrix inverse, and have each party feed the individual bits of their inputs into the circuit. While this would work, it may be too slow to use in practice, since even the simplest operations, e.g., summing two numbers, would require a number of oblivious transfers proportional to the size of the operands in bits. In this section, we will first describe a trivial extension to Yao's original idea, which computes on the integers where the operations are multiplication

and addition. We then describe how we may extend this technique to work for finite precision real numbers. We focus on a two-party protocol, then in a later section we describe how to extend the protocol to $K > 2$ parties.

Our protocol makes heavy use of a particular homomorphic encryption scheme due to Paillier (1999). This is a public key cryptography scheme (Goldreich 2004), that allows two important operations. First, two values encrypted with the same public key may be multiplied together to give an encryption of the sum of the values. Second, an encrypted value may be taken to some power, yielding an encryption of the product of the values. If we use $E_n(a)$ to denote the encryption of $a$ using the public key $n$, then Paillier's cryptosystem has the properties:

$$E_n(a) \cdot E_n(b) \bmod n = E_n(a+b), \quad E_n(a)^c \bmod n = E_n(a \cdot c)$$

Since the operations of modular multiplication and modular exponentiation are widely used in modern cryptography, e.g., in RSA encryption, they are implemented in a number of mathematical packages such as GMP (www.gmplib.org).

A semantically secure system (Goldreich 2004) implies that encryptions of different values are computationally indistinguishable. Paillier's cryptosystem requires a choice of the public key length, $k$, which determines the hardness of breaking the encryption, as well as the length of the "ciphertexts" (the encryptions themselves), and the range of values which may be encrypted. We recommend using $k = 1{,}024$ bits since this gives a good balance of security and efficiency in practice. The key length $k$ becomes the security parameter to our protocol.

We first note that in Yao's original protocol (Goldreich 2004; Yao 1982), the main idea is to keep intermediate values (i.e., the outputs of the intermediate layers of logic gates in the circuit) as "additive shares." That is, rather than a particular party holding an output bit $\sigma$, it is "shared" between the two parties in the sense that party $i$ holds a bit $\sigma_i$ so that $\sigma_1 \oplus \sigma_2 = \sigma$ (where $\oplus$ means "exclusive or"). This way, since each party knows only his value, he learns nothing about the true value of $\sigma$. The final output of the protocol is obtained by the parties combining their shares of the output to reveale the value.

We can easily extend the idea to $\mathbb{Z}/n\mathbb{Z}$ where $n$ is the public key to an instantiation of Paillier's encryption scheme. Shares are now numbers in $\mathbb{Z}_n = \{0 \ldots n-1\}$ instead of bits, "exclusive or" becomes "addition modulo $n$," "and" becomes "multiplication modulo $n$." We will refer to the numbers $a_i \in \mathbb{Z}_n$ so that $\sum_i a_i \equiv a \bmod n$ as an "$n$-sharing" of $a$. First note that an $n$-sharing of the sum of two $n$-sharings may be computed locally by each party, by performing a modular addition. Computing shares of products requires interaction between the parties. We can modify the protocol of Goethals et al. (2004) to obtain shares of the product (rather than the product itself).

In this article, our first contribution is to extend the protocols for computation on $\mathbb{Z}/n\mathbb{Z}$ to approximate the same computations on real numbers by using a fixed precision arithmetic scheme. We use a "2s complement" approach to represent negative numbers, and then a division by a constant to represent real numbers to some fixed precision.

The mapping from $\mathbb{Z}_n$ to the fixed precision real numbers is:

$$f : \mathbb{Z}_n \rightarrow \mathbb{R}, f(a) = M^{-1} \begin{cases} a & a \leq \frac{n}{2} \\ a - n & a > \frac{n}{2} \end{cases} \tag{4}$$

In this way, we associate each element of $\mathbb{Z}_n$ with an element in $\mathbb{R}$. The constant $M$ determines the balance between the range of values which may be represented, and the precision of the fractional quantities which may be represented. A higher value for $M$ yields numbers with greater precision but with a smaller range. Note that we may not just naively apply the protocol for integers, since multiplication of two numbers in this representation results in a stray factor of $M$ (e.g., $f(ab) = Mf(a)f(b)$). Unfortunately, we cannot simply alter the protocol to include a multiplication by the multiplicative inverse of $M$ (if it exists), since we would first need to round the encrypted product to a multiple of $M$ (which would require a subprotocol in itself). Furthermore, we may not simply construct an $n$-sharing of the product and then divide each share locally, since then we may end up with results which are off by multiples of $nM^{-1}$. The reason for this is that the shares may add up to a multiple of $n$ more than the hidden value (due to their definition which used modular addition).

Our proposed solution is to first assume that an upper bound $P$ exists for the magnitude of the product. We then obtain $n$-shares of the product in such a way so that they may be locally converted to P-shares. Concretely, supposing we have the encryption $E(Mab)$ where $a, b$ are not integers but rather these fixed point numbers, we first obtain shares mod $P$, choosing $P$ to be a multiple of $M$, which we write $P = kM$. Hence we obtain $c_1 + c_2 \bmod kM = Mab$, we see that $c_1 + c_2 = Mab + k'M$. Then since we have an encryption of the product as well as a $P$-sharing, we may obtain an encryption of the difference between the sum of shares and the true value of the product (i.e., the multiple of $P$ which is removed by the modular addition). We therefore may obtain the encryption of $c_1 + c_2 - Mab = k'M$. By choosing $M$ to be a number with a multiplicative inverse in the ring mod $n$, we may then securely divide this residual by $M$. Thus, we may securely obtain $M^{-1}Mk' = k'$. With this in hand we first locally divide the shares $c_1, c_2$ by $M$, yielding a new sharing with an error of $k'$. Then we may correct this error by sharing the value $k'$ which was already securely computed, and subtracting these shares locally. The protocol is presented in two parts, the first part turns an encryption into a $P$-sharing (Figure 1), and the second part turns an encryption and, its corresponding $P$-sharing into an $n$-sharing of the floor of the product divided by $M$ (Figure 2).

Below we state the two required protocols. With them we can construct a simple protocol for computing a sharing of a product of shares. The first step is to obtain the encryption of the product, and then run the protocol of Figure 2. To obtain the encryption of the product, the idea is for the party who knows the private keys to send encryption of his shares along with the encryption of his product of shares to the other party. This way the other party may construct the encryption of the product, using the homomorphic properties of Paillier's encryption. Furthermore this complete protocol is secure, since the messages being passed are encrypted values under a semantically secure encryption scheme.

- **Input**
  Party 1 has the private key to an instance of Paillier's encryption scheme, Party 2 knows the corresponding public key $n$, and has the encryption under $n$ of a value $p$, denoted $E(p)$. Furthermore $p$ is bounded so that $2|p| < P$, or in other words $p \in \{0 \ldots P/2\} \cup \{n - P/2 \ldots n - 1\}$.
- **Step 1** Party 2 draws $r$ uniformly at random from the set $\{P \ldots n - 1\}$ then computes $E(p - r)$ by means of the homomorphic properties of the crypto system. This encrypted value is sent to Party 1.
- **Step 2** Party 1 decrypts the value to obtain $s = p - r \bmod n$.
- **Output** Party 1 outputs $s - n \bmod P$ and Party 2 outputs $r \bmod P$, where "mod" in this case means the operator which returns the remainder from integer division by $P$.

*Fig. 1. A protocol for generating a P-sharing of an encrypted value*

Then security follows from the composition theorem (essentially, using a secure protocol as a subroutine will still yield a secure protocol); see for instance, Goldreich (2004).

We first demonstrate that the protocol of Figure 1 is correct. At Step 2, we have that $s + r \equiv p \bmod n$ and so $s + r + (P/2) \equiv p + (P/2) \bmod n$. Since $s, r \in \mathbb{Z}_n$ we have that $0 \leq s + r \leq 2n$, and so either $s + r + (P/2) = p + (P/2)$ or $s + r + (P/2) = p + (P/2) + n$. Since in Step 1 we constructed $r$ to be greater than $P$, which is greater than $p + (P/2)$, we see that the latter condition holds. Therefore the sum of outputs mod $P$ is:

$$s + r - n \bmod P = s + r - n + \frac{P}{2} - \frac{P}{2} \bmod P = p + n - n + \frac{P}{2} - \frac{P}{2} \bmod P = p \bmod P \qquad (5)$$

and so the protocol computes a $P$ sharing of $p$.

To show that the protocol is secure in the face of semi-honest parties, we construct a simulator for the view of each party during execution. The only message sent in this protocol is $E(p - r)$ in Step 1. The decryption of this value is not uniform on $\mathbb{Z}_n$ since the noise added was in the interval $\{P, \ldots, n\}$, so the value $s$ is uniform on a subset of $\mathbb{Z}_n$ which depends on the variable. Nevertheless, since that subset consists of the

- **Input**
  Party 1 has the private key to an instance of Paillier's encryption scheme, Party 2 knows the corresponding public key $n$, and has the encryption under $n$ of a value $p$, denoted $E(p)$. Furthermore $p$ is bounded so that $2|p| < P$, or in other words $p \in \{0 \ldots P/2\} \cup \{n - P/2 \ldots n - 1\}$. $M = 2^m$ is a power of 2, and $P$ is a multiple of $M$.
- **Step 1** The parties run the protocol of Figure 1, and obtain the $P$-sharing $p_1$, $p_2$ of $p$.
- **Step 2** Party 1 encrypts $p_1$ and sends $E(p_1)$ to Party 2, who uses the homomorphic properties to compute $E(p_1 + p_2)$ and then $E(\hat{p}) = E(p_1 + p_2 - p)$.
- **Step 3** Party 2 uses the multiplicative inverse of $M$ along with the homomorphic properties, to compute $E(M^{-1}\hat{p})$, then draws $r$ uniformly from $\{0 \ldots n - 1\}$ and sends $E(M^{-1}\hat{p} - r)$ to Party 1.
- **Step 4** Party 1 decrypts the message to obtain $s = M^{-1}\hat{p} - r \bmod n$.
- **Output** Party 1 outputs $\left\lfloor \frac{p_1}{M} \right\rfloor - s \bmod n$. Party 2 outputs $\left\lfloor \frac{p_2}{M} \right\rfloor - r \bmod n$.

*Fig. 2. A protocol for securely computing the floor of the division of an encrypted value by a public constant M*

overwhelming majority of $\mathbb{Z}_n$, the distribution is computationally indistinguishable from a uniform distribution. Denoting the distribution of $s$ as $U_n^P$, and the uniform distribution on $\mathbb{Z}_n$ by $U_n$, consider the variation distance:

$$\frac{1}{2}\sum_{x=0}^{n} |U_n^P(x) - U_n(x)| = \frac{m^2}{n} \leq \frac{P^2}{2^k} \tag{6}$$

Since $P$ is a constant, we have that the variation distance between the two distributions is bounded above by a negligible function of the security parameter $k$, which implies that the two distributions are computationally indistinguishable (see Goldreich 1998, page 81). Therefore we may draw values from $U_n$ to simulate the message received by Party 1.

We now turn to the protocol of Figure 2. We have that, in Step 2, $\hat{p}$ is a multiple of $P$, which is itself a multiple of $M$. The multiplicative inverse of $M$ is $((n+1)/2)^m \bmod n$. Multiplying $\hat{p}$ by this value then gives

$$M^{-1}\hat{p} = \left\lfloor \frac{\hat{p}}{M} \right\rfloor.$$

The sum of the output is:

$$\begin{aligned} \left\lfloor \frac{p_1}{M} \right\rfloor - s + \left\lfloor \frac{p_2}{M} \right\rfloor - r \bmod n &= \left\lfloor \frac{p_1}{M} \right\rfloor + \left\lfloor \frac{p_2}{M} \right\rfloor - \left\lfloor \frac{\hat{p}}{M} \right\rfloor \bmod n \\ &= \left\lfloor \frac{p+\hat{p}}{M} \right\rfloor - \left\lfloor \frac{\hat{p}}{M} \right\rfloor + \epsilon_1 = \left\lfloor \frac{p}{M} \right\rfloor + \epsilon_2 \bmod n \end{aligned}$$

where $|\epsilon_1| \leq M^{-1}$ and $|\epsilon_2| \leq 2M^{-1}$ are error terms resulting from taking the sum of the floors as opposed to the floor of the sum.

The protocol is secure since all messages are either encrypted under a semantically secure encryption scheme, or are distributed uniformly at random in the set $\{0 \ldots n-1\}$. In the first case, such messages may be simulated by encrypting a random value with the public key $n$. In the latter case, messages may be simulated by drawing a number uniformly at random from the set $\{0 \ldots n-1\}$.

We conclude this section by noting that with these constructions we have a means to compute a function consisting of sums and products on the real numbers. There is approximation involved, since only those real numbers which correspond to multiples of $M^{-1}$ may be represented exactly. Using a large value for $M$ such as $2^{64}$ yields sufficient precision for our purposes.

## 4. A Two-Party Protocol for Secure Linear Regression

Using the tools of Section 3 we may construct a secure protocol for evaluating (2) and (3). First, using the constructions for sums and products we may compute additive shares of the data covariance matrix $X^TX$ and the vector $X^Ty$. All that remains is to securely invert the covariance matrix. We use a technique explored by Guo and Higham (2006) which reduces the problem of inverting a matrix to the problem of computing sums and products of matrices, which we may do securely with the above constructions.

### 4.1. Matrix Inversion

First, we note that we can obtain the reciprocal of a number $a$ without any actual division by an application of Newton's method to the function $f(x) = x^{-1} - a$. Iterations follow $x_{s+1} = x_s(2 - ax_s)$, which requires multiplication and subtraction only.

It turns out that we can apply the same scheme to matrix inversion; see for instance, Guo and Higham (2006) and references therein. A numerically stable, coupled iteration for computing $A^{-1}$, takes the form:

$$\begin{aligned} X_{s+1} &= 2X_s - X_s M_s, \quad X_0 = c^{-1} I, \\ M_{s+1} &= 2M_s - M_s^2, \quad M_0 = c^{-1} A, \end{aligned} \tag{7}$$

where $M_s = X_s A$, and $c$ is to be chosen by the user. A possible choice, leading to a quadratic convergence of $X_s \to A^{-1} (M_s \to I)$, is $c = \max_i \lambda_i(A)$. In our actual implementation we consider instead the trace (which dominates the largest eigenvalue, as the matrix in question is positive definite), since we can compute shares of the trace from shares of the matrix locally by each party. To compute $c^{-1}$ we use the same iteration, with scalars instead of matrices. For this iteration we initialize with an arbitrarily small $\epsilon > 0$ (as convergence depends on the magnitude of the initial value being lower than that of the inverse we compute).

This technique is iterative, so we have a choice regarding how many iterations to run. We may either employ a convergence check after each iteration, or instead simply upper bound the number of iterations required for convergence and just perform that many iterations. The former choice may terminate after fewer iterations, however the number of iterations performed will unnecessarily reveal information about the input. For example, when computing the reciprocal, a larger value of $a$ will take fewer iterations to converge. For computing a reciprocal, an extremely conservative lower bound on the number of iterations required is $2 \log_2 M$. After this many iterations even the smallest value representable ($M^{-1}$) will be inverted. For our suggested $M = 2^{64}$ we may then iterate for 128 iterations to be guaranteed to compute the reciprocal.

Computing the inverse of a matrix is a more expensive operation, since $2p^{2.8}$ products are required at each iteration (using Strassen's approach for matrix multiplication), therefore we seek a tighter lower bound on the number of iterations required for convergence. Unfortunately, the exact number of iterations required for convergence depends on the eigenvalues of the matrix, which we assume should be secret. Note that we have convergence as soon as $M_s = I^{p \times p}$, as then the $X$ iterate of (7) ceases changing. Denote by $\lambda_i(s)$ the *ith* eigenvalue of $M_s$, then we have that the eigenvalues of $M_{s+1}$ follow

$$\lambda_i(s+1) = 2\lambda_i(s) - \lambda_i^2(s). \tag{8}$$

We can restate the condition for convergence as $\lambda_i(s) = 1$ for all $i$. Unfortunately, unless we have a lower bound for the lowest eigenvalue of the matrix $M_0$, we must use $2 \log_2 M$ iterations as before. In our case, $A = X^T X$, and so the smallest eigenvalue of $M_0$ depends on the condition number of the sample covariance matrix, that is, the ratio of the largest and smallest eigenvalues. If the parties have prior information that their covariance matrix will be well conditioned, then this could be the basis for choosing the number of iterations

to run, for instance, if as a preprocess they engaged in a protocol for uncorrelating the variables, or standardizing them.

If there is no usable bound on the condition number, then the parties may either run for $2 \log_2 M$ iterations as before, or may choose to test for convergence every few iterations, and stop when convergence is achieved. This way would leak information about the condition number of the covariance matrix, which may be acceptable depending on how the data is shared between the parties. By increasing the number of iterations between convergence checks, the amount of information leaked decreases, since the mapping from matrices to iteration numbers becomes coarser. To test convergence we suggest testing whether $Tr(M_s) \geq p - \epsilon$ for some small epsilon. As the method converges, $M_s$ tends towards the identity matrix, and so the trace equals $p$ when the algorithm has converged. Moreover, it stays below $p$ until that point, and so only a one-sided check is required. Performing this test on additive shares of the matrix trace is equivalent to one of the first protocols in cryptography, the so-called "Millionaires Problem." A protocol for evaluating the "greater-than" predicate is given in for example, Blake and Kolesnikov (2004). An alternative to the above is to first standardize the data, which gives some improvement to the condition of the matrix, and then to run for some fixed number of iterations decided a-priori. If the number is sufficiently large, the result will be accurate.

### *4.2. A Complete Protocol for Multiple Linear Regression*

By putting together all the techniques we have constructed so far, we are in a position to propose a complete protocol for the linear and ridge regression. The protocol begins by using the constructions of Section 3 to compute additive shares of $X^T X$ and $X^T y$. Using the above method for matrix inversion, we may then compute additive shares of $(X^T X)^{-1}$. Finally we compute shares of the product $(X^T X)^{-1} X^T y$. After this, each party should send his shares of the parameter to the other party. After receiving all the shares, each party may then do a modular addition to reveal the final estimated vector $\hat{\beta}$. To extend this to ridge regression, all that is required is for one party to add $\lambda I$ to his share of $X^T X$ (or alternatively, for each party to add $2^{-1}\lambda I$ to their shares). Security of the full protocol follows from the composition theorem (see, e.g., Goldreich 2004) where secure subprotocols compose into a larger secure protocol.

For the protocol to be correct, we must be careful in choosing $P$ so that it is a correct upper bound. First we note that $P$ should bound every element of $X^T X$. A loose upper bound on the magnitude of these values is to take the largest element of $x_m = \max_{i,j} |X_{i,j}|$ and take $P \geq p x_m^2$. It is reasonable that parties may be unwilling to divulge the maximum magnitude of elements of $X$. In this case we may base $P$ off a very loose upper bound for the maximum, for instance, the maximum size of an integer. In our experiments we choose $P = 2^{128}$ and $M = 2^{64}$ as this gives the ability to store numbers with magnitude up to $2^{63}$. For this choice, using 1,024 bit long keys is also fine, as the bound obtained in (6) is vanishingly small.

## 5. Extension to $K > 2$ Parties

We note that our protocol has two main phases. The first phase is the secure computation of the data covariance matrix from the inputs, and the second phase is the inversion of

this matrix. We see that in extending to three or more parties, the first phase decomposes into a set of pairwise interactions between parties, and so the existing protocol may be used by each pair, to generate a share of the data covariance matrix. This is due to the decomposition:

$$X^T X = \sum_i X_i^T \sum_j X_j = \sum_i X_i^T X_i + \sum_{i \neq j} X_i^T X_j + X_j^T + X_i$$

Each term on the right-hand side may be computed locally by the respective party, then each term in the second sum may be computed by invoking our basic construction for products with two parties. Lastly, each pair of parties will need exactly one party who knows the key to the instance of Paillier's encryption scheme. Therefore there will be more instances of Paillier's encryption scheme used during execution of the protocol. Since the sharing of outputs generated by our product protocols depend on the public keys used, we need some way to ensure that all the outputs of the pairwise protocol invocations can be transformed into sharing with a common $n$.

To convert the shares, the main idea is to create a $P$-sharing for some upper bound $P$, then compare the encrypted sum of shares to the encrypted sum of $n$-shares (i.e., the encryption of the hidden value). Subtracting one from the other gives a term which is a multiple of $P$. The idea is to divide by $P$ (if $P$ has a multiplicative inverse in the ring mod $n$), then share the first two bits of the result. This way, it is possible to construct the encryption of the same multiple of $P$ under a new public key $m$. This value may then be shared with techniques similar to those described in Figure 1.

After running this protocol the $K$ parties will have a $n$ sharing of the data covariance and $X^T y$, where $n$ is a public key to which only one party knows the private keys. At this point one possibility is for the parties to break into two groups, and pool their shares within each group. Then each group could act as a single "party" and invoke the two-party version of the matrix inversion protocol. This technically would achieve the definition of privacy, however it would give an opportunity for two parties to try to collude to learn these intermediate values. Therefore we instead extend the protocols for computing products of shares, so that all $K$ parties are involved in the computation. This way it takes all the parties to "collude" in this way in order to be guaranteed to learn the intermediate values.

The first step is to generalize the protocol for computing the $P$-sharing of the product. The idea is that, like Figure 1 one party will know the private key, and a different party will know the encryption of the product. The parties will form a chain and pass the encrypted value along, so that each party will add a random draw to the encrypted value, until it finally arrives at the party who holds the key, who decrypts it. This way every party is given an $n$-share of the product. In order to convert this value to a $P$-share we must be careful about the domain of the random variables, so that we may ensure that the multiple of $n$ which appears in their sum is known to us (as in Figure 1 where we constructed $r$ so that the sum was equal to $n$ plus the product). Here we suggest to ensure the sum is $(K-1)n$ plus the hidden value.

Correctness of this protocol follows a similar argument as the two party protocol. Showing security of this protocol is more complicated, since the distribution of $s$ is no longer uniform. The reason is that the sum of uniform draws is distributed in a way which is peaked around the expectation, and nonuniform. Nevertheless it is possible to show that

- **Input**
  Party 1 has the private key to an instance of Paillier's encryption scheme, party K has the encryption under $n$ of a value $p$, denoted E($p$). Furthermore $p$ is bounded so that $2|p| < P$, or in other words $p \in \{0 \ldots P/2\} \cup \{n - P/2 \ldots n - 1\}$. Every party knows the public key $n$.
- **Step 1** Whichever party holds the encryption (Party $i$) draws $r_i$ uniformly at random from the set $\{\frac{K-1}{K}n + \frac{P}{K} \ldots n - 1\}$. He then computes this value for the encrypted value by means of the homomorphic properties of the crypto system. This encrypted value is sent to Party $i - 1$.
- **Step 2** Step 1 is iterated for each party until the encryption arrives at Party 1.
- **Step 3** Party 1 decrypts the value to obtain $s = \rho - \Sigma_{i=2}^{K} r_i \bmod n$.
- **Output** Party 1 outputs $s - (K - 1)n \bmod P$ and Party $i > 1$ outputs $r_i \bmod P$, where "mod" in this case means the operator which returns the remainder from integer division by $P$.

*Fig. 3. A K-party protocol for generating a P-sharing of an encrypted value*

the variation distance between two such distributions whose expectations differ by less than $P$ is negligible as log $n$ increases.

Having generated a $P$-sharing of the products, we can easily extend the protocol of Figure 2 to a $K$-party protocol. The main idea of the protocol of Figure 2 is to compare the sum of shares to the encryption of the true value, and then create a term which gets added to the scaled shares in order to cancel whatever stray factor comes from the sharing. Now, in Step 2, the encrypted sum involves terms from all parties. In Step 3, after multiplying the encryption by the inverse of $M$, the encryption needs to be passed to each party (as in the above protocol) so that each one has a chance to get a piece of the sharing of the value, before it arrives back at Party 1 who decrypts it.

We note that this protocol is more complicated than the two-party protocol. Nevertheless as we show above, a subset of the parties may not subvert the protocol so long as they remain semi-honest. Perhaps more of a concern is to determine which party will act as the holder of the private key. We note that the computational demands placed upon the parties are roughly the same so this decision likely has little impact on the overall running time. The order of the parties in the round-robin type protocol could be decided by for instance, attempting to minimize the total round trip time (e.g., to choose the ordering which results in the smallest average latency between neighbors in the ordering). Although this is technically a computationally hard problem (the "traveling salesman" problem), for a small number of parties as we envision here it may be performed reasonably quickly.

## 6. Scalability

We now adress the scalability of this approach by examining the number of invocations of the subprotocols as a function of the number of covariates $p$ and samples in the data $n$ (note that in this section $n$ is the size of the data and not a public key). We break the protocol down into three parts, the construction of $X^T X$, $X^T y$, the inversion of $X^T X$, and finally the matrix-vector multiply to yield the estimate. We only count the number of secure products required by the protocol, since sums are carried out locally and therefore have a very small impact on the runtime.

We first consider the construction of $X^T X$, $X^T y$. For horizontally partitioned data this step is trivial as the parties locally compute these quantities on their sample, which yields a sharing of the full quantities. In the case of vertically partitioned data, our protocol requires inner products of vectors of size $n$ (namely $y$ and the columns of the design matrix). Evidently each such inner product requires $n$ secure multiplications. Therefore this stage of the computation requires time linear in $n$. Suppose the parties each have $p_i$ of these vectors so that $\sum_{i=1}^{P} p_i = p + 1$, then the number of inner products required is $\sum_{i \neq j} p_i p_j$. This quantity is bounded above by $4^{-1} P p^2 + 2^{-1} p$ (for example with two parties, the worst case is when each has half of the columns). So we see that the time taken at this stage grows at most quadratically in the number of covariates $p$.

The matrix inversion (irrespective of the partitioning scheme) requires $O(p^2 \log\ p)$ multiplications. The reason is that each step of our iterative scheme requires a matrix multiplication of two $p \times p$ matrices, and the number of steps for convergence is $O(\log p)$. Since we envision a scenario in which $p \ll n$, the time for this step is overshadowed by the above step. Likewise the final step consists of a multiplication of a $p \times p$ matrix with a $p \times 1$ vector. This takes $p^2$ multiplications. Therefore the computational burden of our protocol is contained within the construction of the first two terms.

As we noted above for $K > 2$ parties, there is opportunity to parallelize the computation of shares of $X^T X$ since the product breaks down into pairwise products between parties. If the number of cases is large, then each of these computations may take a long time to complete. We see that we may break down each product into a sum of products over horizontal slices of the matrices. If we have $X_i^T = \left(X_{i,1}^T, X_{i,2}^T\right)$ and $X_j^T = \left(X_{j,1}^T, X_{j,2}^T\right)$, and the corresponding blocks are the same size, then:

$$X_i^T X_j = X_{i,1}^T, X_{j,1} + X_{i,2}^T X_{j,2}.$$

Thus each pair of parties may agree on a partitioning of the data matrix, so that they may compute products of the blocks in parallel and then at the end sum their shares locally. This data splitting technique yields a procedure which is "embarrassingly parallel" and so we anticipate that if the blocks are approximately equal size, then the procedure should be sped up by a factor equal to the number of blocks (modulo differences in hardware used for different blocks of the partition). We also note that the matrix inversion procedure is built out of matrix multiplies of matrices of shares. It is possible to parallelize each of these products in the same way, although that matrix is only $p \times p$ and so we anticipate the products not taking long on even one machine (for, e.g., $p$ up to 20 or so).

## 7. Model Checking and Inference

Obtaining the regression coefficient estimates is just the tip of the inference iceberg, as far as modeling and inference are concerned. Checking the aptness of the fitted model is usually carried out by exploring functions of the regression residuals. The coefficient of determination $R^2$ is a standard summary of the predictive ability of the regression equation:

$$R^2 = 1 - \frac{e^T e}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

where $e = (I - H)y$ is the residual vector, given in terms of the so-called *hat* matrix $H = X(X^TX)^{-1}X^T$. By using the constructions of this article, we can construct a protocol for estimating the coefficient of determination. We can compute the "hat" matrix $H$ by means of the protocols for sums and products, making use of the matrix inversion outlined in Section 4.1. We would then apply the same protocols to the numerator and to the denominator, leading to a secure computation of $R^2$, once we "invert" the denominator. Similarly we can compute variants on $R^2$, and other summary functions used in model search.

Testing the fit of the linear regression is yet another important statistical concern. Model inferences about the estimated coefficient vector, which include constructing confidence intervals and performing hypothesis testing, require an estimate of the noise variance and an estimate of the covariance matrix of the coefficients estimate. The former is given by $\hat{\sigma}^2 = e^Te/(n - p - 1)$ when the design is in $\mathbb{R}^{n\times p}$, and the latter is $\hat{\sigma}^2(X^TX)^{-1}$. These two estimates are linear functions of the "hat" matrix. Therefore, we can construct a protocol to compute them. The square roots of diagonal elements of $\hat{\sigma}^2(X^TX)^{-1}$ are the standard errors of the coefficient estimates. These are necessary, as mentioned above, for the construction of confidence intervals for the coefficient estimates as well as for hypothesis tests about the latter. Instead of explicitly calculating and releasing these standard errors, we may construct fully secure protocols which only reveal $p$-values related to hypothesis tests. Taking one step further along that path, we can report an interval containing the $p$-value instead of the exact figure. For example, if the $p$-value is .0005, one may report "smaller than .001." By doing so, we guarantee that no adversary can unwind a formula to determine a coefficient's standard error.

As we mentioned towards the end of Section 2, we are primarily concerned here with leakage from the computation itself, and not from whatever can be learned from the output. Appendix A explores possible leakage when the output consists solely of the estimated parameter vector. When the goal is to produce other summaries beyond the parameter vector (such as the coefficient of determination), then the output of the protocol must be modified to include these statistics, in order to enjoy the security guarantees described in the article. In such a case, an assessment of the security implications of revealing such statistics is necessary.

### *7.1. The Release of the Full Covariance Matrix*

When the goal is a detailed statistical analysis then the parties might choose to share the full data covariance matrix $X^TX$. This is the setting explored in for instance, Karr et al. (2009). Although the protocol we proposed above computes the regression coefficients directly, note that it is trivially modified to reveal the statistics $\{X^TX, X^Ty\}$ instead, so that the parties themselves may carry out the remaining analysis "in situ." We could achieve this by running the protocol to compute the shares of $\{X^TX, X^Ty\}$, but before the matrix inversion takes place. Thereupon the parties would pool these shares to recover the requisite statistics, at which point the protocol would end. In making this change we have redefined the protocol so that these values would take the place of the regression coefficients as the output of the protocol. As we stated previously, our protocol only seeks to prevent the leakage of private information due to the computation itself, and it is

ambivalent about privacy implications due to the output itself. Therefore before considering such a modification, the parties would have to agree that the data covariance matrix is essential to the analysis and that therefore they can tolerate whatever risk to privacy it introduces. To see that (in general) the regression coefficients reveal less information than the data covariance matrix please see Appendix A below.

In the event that the parties choose to share the data covariance matrix, it is plausible that the protocol of Karr et al. (2009) is more appropriate from the perspective of computational efficiency—since it relies on techniques which are less burdensome than homomorphic encryption. Nevertheless our approach is clearly advantageous when security is a top priority, since it maintains the cryptographic definition of security, whereas their protocol achieves a weaker notion of security. How important this distinction will be depends heavily on the data in question and whatever properties of the information are known beforehand.

The comparison with the work of Karr et al. (2009) (and alike) brings us naturally to a discussion about a risk-utility paradigm. Associated with every statistical disclosure limitation method, or any statistical protocol aimed at providing statistical analysis without compromising privacy, there is a risk-utility tradeoff that gets affected by changes in the method's parameters (see Duncan et al. 2001a, b). In contrast to the approach of Karr et al. (2009), we note that the utility here may be measured by computational efficiency rather than by statistical efficiency. When the decision is to share data covariances matrices, both our approach and the approach of (Karr et al. 2009) provide the same (statistically efficient) answer. It is the computational aspects in which the two approaches differ. Our approach provides stronger security guarantees, and therefore may require longer computations. It is up to the managers of information organizations to decide what is more important for them in a specific task.

## 8. Simulation Experiments

As Karr et al. (2009) mention, the repeated application of public key cryptography (which we advocate) is slower than alternative techniques that supposedly preserve privacy, although without achieving the strict definition we adhere to. We demonstrate that our protocol is useful in practice by implementing a simulator of a three party setup, using 1,024 bit long keys.

We used the GMP library to handle the arithmetic operations on the large numbers required during the protocol. We simulated the computation required for the three party version of the protocol. Our simulations took place on individual machines, rather than two computers communicating over a network, and so our timings do not take into account the time taken to transfer data between the parties. Instead our experiments show that the amount of *computation* required to run our protocol is acceptably small even with large datasets.

For the experiment we took the Current Population Survey data,[4] which consists of 51,016 cases with 23 covariates each (after converting categorical covariates into sets of binary flags). Each case represents one household and the covariates consist of measurements such as income, education level etc. We constructed a regression problem

[4] http://www.bls.gov/CPS/

where the response is the log of the household income, and the other 22 covariates are used as predictors. We split the data among the three parties in a column-wise fashion, so that each party held a data set which was the union of a set of columns, and the parties' datasets did not overlap with each other. The simulated parties held 10, 8 and 4 covariates respectively, with each holding the same set of attributes for all the cases. We note that although we described our approach in the context of continuous covariates, it handles binary flags equally well, by using values of 0.0, 1.0 for these covariates.

We split the data matrix into three blocks for computing the data covariance matrix, as indicated in (6), so there were nine machines used in computing this stage of the protocol (three for each pair of parties). For inverting the matrix, we used only one machine. We standardized the data ahead of time so that the problem would be better conditioned. We also halted the matrix inversion procedure after 40 iterations, as opposed to the 128 which would be necessary to guarantee "convergence" if the problem was extremely ill-conditioned. Comparing the estimated parameters to the regression coefficients estimated using R locally on one machine, we found that the estimates agreed up to at least 3 decimal places. The disagreement in the remaining digits can be attributed to the matrix inversion procedure, and the slight loss of precision faced by our fixed point arithmetic scheme. Greater precision would be obtained by taking larger values for the constants $P$ and $M$, but this would also require larger public keys (and hence computation and communication overhead) to ensure the same degree of security. Overall, our experiment took two days to complete, where the first day roughly corresponded to the computation of the shares of the covariance matrix, and the second day corresponded to inverting the matrix on one computer.

We believe that we could speed up these calculations substantially by making further use of parallelization.

## 9. Discussion

In this article, we have presented a protocol which achieves the cryptographic definition of security, when the only output is the regression coefficient estimates, and perhaps multiple statistics related to the goodness of fit. We have demonstrated that a fully secure approach to linear regression based on the homomorphic encryption is practical for use on moderately large datasets shared between several parties. We emphasize that our protocol (like any cryptographic protocol) prevents leakage of information which may arise from the computation itself. It does not address any leakage which results from the output. Below in Appendix A we give some comments on the amount of leakage under different input partitioning schemes.

Our approach offers more rigorous guarantees with respect to the privacy of the input data than previous such protocols. But since we use computationally demanding cryptographic primitives to achieve this security, our protocol is rather slow when compared with that of Karr et al. (2009). It is important to understand the strengths and weaknesses of both methods in order to make an informed choice about which is more appropriate to use in a particular situation.

First, we recall that the subprotocol used for products and sums in Karr et al. (2009) fails to meet the standards of the cryptographic definition of security. To eschew this issue we may consider an alternative protocol, which uses, for instance, our above methods for

products and sums to construct $X^TX$ and $X^Ty$, at which points these statistics are reassembled from the shares and then disseminated amongst the parties. This would give a secure analog to their protocol that is clearly less computationally demanding than our full protocol since it does not require the secure matrix inversion or the final round of secure inner products for computing the final estimate. Rather, these operations are performed locally by each party. Whether or not it provides a sufficient privacy guarantee depends on whether the data covariance matrix and the projection of $y$ onto the covariates are appropriate to release. From the discussion in Appendix A, we see that these statistics cannot be reconstructed completely from the regression coefficients, and therefore our full secure protocol does not release them. If the parties somehow come to the consensus that these statistics do not impinge on privacy, then clearly the above modification of (Karr et al. 2009) is appropriate to use instead.

We can, in principle, combine both approaches and at the same time attempt to make the output satisfy a formal definition of privacy such as "$\epsilon$-differential privacy," an approach due to Dwork (2008) and Nissim (2008). Dwork et al. (2006) discuss efficient ways to do this for several problems involving the secure evaluation of sums, whereas our protocols involve calculation of secure sums and products. This combined secure-private approach would involve computing some form of perturbed regression coefficients and statistics for assessing goodness of fit.

A very different approach involves carrying out data sanitization directly on the data held by the parties. This would entail the parties each adding random noise to their data in an effort to preserve individuals' privacy, while maintaining some form of utility in the data. Next, the parties would share these sanitized databases among themselves, at which point they could perform whatever statistical analysis they wanted. This approach requires a formal definition of privacy to be achieved via the sanitization process, for instance, using "$\epsilon$-differential privacy." Were we to insist on this cryptographic definition of privacy, the use of a sanitization scheme would thwart the data merger, except in the case of horizontal partitioning, and even then it would affect the regression coefficients and related goodness of fit statistics. There is no developed theory that would allow us to carry out accurate statistical inference under such a scheme. Therefore, although the approach is a conceptually appealing alternative, we would need to do further work before it would be practical for multi-party statistical calculations, especially in moderate to high-dimensional problems.

The problem of secure regression is far from solved, however, since we have yet to deal with missing data (e.g., by multiple imputation), measurement error, and possibly overlapping entries, in a secure way. Furthermore, record linkage due to a statistical model may be incorrect and result in biased estimates of model parameters. Extensions of the present work would include approaches that are robust to these kinds of errors, and also methods for generalized linear models such as logistic and poisson regression.

## Appendix

### A. Privacy Implications Associated with the Release of OLS Estimates

Our protocol computes the ordinary least squares regression estimates for a design and response vector which is somehow shared between two parties. The OLS is a function of

the inputs from the parties consisting of "private" data. The definitions used in constructing the protocol itself ensure that nothing is revealed besides the estimated parameter vector *and whatever is implied by it*. Here we investigate the implications of knowing an OLS estimator which is based partially on the private data of another party.

The main idea is to look at the set of data matrices which could be input by the other party and would produce the same coefficients vector as the one observed during an actual run of the protocol. First let $\hat{\beta} = \hat{\beta}(X_1, X_2, y_1, y_2)$ be the coefficients computed by our protocol, on the inputs $(X_1, y_1)$ belonging to Party 1, and $(X_2, y_2)$ belonging to Party 2. We will examine the structure of the set:

$$M_2 = \{(M, v) | \hat{\beta}(X_1, X_2, y_1, y_2) = \hat{\beta}(X_1, M, y_1, v)\}$$

All that Party 1 may conclude after running the protocol with Party 2, is that the data belonging to Party 2 is an element of set *M2*. The construction is the same for the privacy of Party 1. If this set contains a single element (i.e., if $\hat{\beta}$ had an inverse) then Party 1 would know the data of Party 2 and completely violate his privacy. On the other hand if the set $M_2$ is large, then Party 1 may only conclude whatever is implied by the structure of the set (i.e., whatever properties are shared by every element). We will examine the structure of the set under two common data partitioning schemes. Note that although we concentrate on a setup which mimics the two-party protocol, the same ideas carry through for analysis of the case where there are multiple parties. We may either say that Party 1 wants to investigate the other parties' data by himself in which case take $(X_2, y_2)$ to mean the union of the data belonging to other parties, or perhaps several parties will collude to try to reveal the data of another party, in which case take $(X_1, y_1)$ to mean the union of the data of the colluding parties.

### *A.1. Horizontally Partitioned Data*

Consider the case where there are two parties who each have a horizontal slice of the design matrix and the response vector. We have $X = \left(X_1^T, X_2^T\right)^T$ where $X \in \mathbb{R}^{(n+m)\times p}$, $X_1 \in \mathbb{R}^{n\times p}$ and $X_2 \in \mathbb{R}^{m\times p}$. Also $y = \left(y_1^T, y_2^T\right)^T$ for $y \in \mathbb{R}^{n+m}$, $y_1 \in \mathbb{R}^n$ and $y_2 \in \mathbb{R}^m$.

We can write he OLS estimator as:

$$\hat{\beta}(X_1, M, y_1, v) = \left(X_1^T X_1 + M^T M\right)^{-1}\left(X_1^T y_1 + M^T v\right)$$

and so we have:

$$\left(X_1^T X_1 + M^T M\right)\hat{\beta} = X_1^{T y_1} + M^T v$$
$$\left(X_1^T X_1\right)\hat{\beta} - X_1^T y_1 = M^T v - M^T M\hat{\beta}$$
$$\Delta = M^T(v - M\hat{\beta}),$$

where we have defined $\Delta$ accordingly. If we assume that $X_1$ is of full rank (i.e., rank $p$) then we have that $\Delta$ is in one-to-one correspondence with $\hat{\beta}$. We may then rewrite the set $M_2$:

$$M_2 = \{(M, v) \in \mathbb{R}^{m\times p} \times \mathbb{R}^m | M^T(v - M\hat{\beta}) = \Delta\}$$

where $\hat{\beta} = \hat{\beta}(X_1, X_2, y_1, y_2)$. Therefore all Party 1 may conclude is the equality implied in the definition of the set. Note that if he drew a full rank matrix $M \in \mathbb{R}^{m\times p}$ at random and

set $v = M(M^TM)^{-1}\Delta + M\hat{\beta}$, then the pair $(M, v)$ could be used in place of the data of Party 2, and would produce the same coefficient vector. Note that $v - M\hat{\beta}$ is the coordinates of $\Delta$ in the row space of $M$. For $m > p$ this is not unique since the rows form an overcomplete basis for the space. Therefore, we have that the design held by Party 2 could be any $m \times p$ matrix where $\Delta$ is in the row space, and associated with each choice is a set of at least one vector which could be the response vector. Since $M_2$ contains so many different elements which are all over the space of matrices $M$, it will be difficult for Party 1 to conclude anything about the data matrix.

### A.2. *Vertically Partitioned Data*

Consider the case where there are two parties, each with a matrix of covariates $X_1 \in \mathbb{R}^{n\times p}$, $X_2 \in \mathbb{R}^{n\times q}$, and Party 1 holds the response vector $y \in \mathbb{R}^n$. We take $X = (X_1, X_2) \in \mathbb{R}^{n\times(p+q)}$ as the complete matrix of predictors. Our protocol computes the coefficient vector:

$$\hat{\beta}(X_1, X_2, y) = [(X_1, X_2)^T(X_1, X_2)]^{-1}(X_1, X_2)^T y = (X^TX)^{-1}X^Ty$$

We will start by investigating what Party 1 (the holder of the response) may learn about the predictors belonging to Party 2. First note that we may write the formula for $\hat{\beta}$ in terms of the block matrix, and use the technique for inverting a block matrix:

$$\begin{aligned}\hat{\beta}(X_1,M,y) &= \begin{pmatrix} X_1^TX_1 & X_1^TM \\ M^TX_1 & M^TM \end{pmatrix}^{-1} \begin{pmatrix} X_1^T \\ M^T \end{pmatrix} y \\ &= \begin{pmatrix} \left(X_1^TX_1 - X_1^TM(M^TM)^{-1}M^TX_1\right)^{-1}\left(X_1^Ty - X_1^TM(M^TM)^{-1}M^Ty\right) \\ \left(M^TM - M^TX_1\left(X_1^TX_1\right)^{-1}X_1^TM\right)^{-1}\left(-M^TX_1\left(X_1^TX_1\right)^{-1}X_1^Ty + M^Ty\right) \end{pmatrix}\end{aligned}$$

It will be useful to express each $M$ in terms of its projection onto the column space of $X_1$ and note a few properties:

$$\begin{aligned} &M = X_1A + \tilde{M} \\ &A = \left(X_1^TX_1\right)^{-1}X_1^TM \\ &\tilde{M}^TX_1 = (M - X_1A)^TX_1 = M^TX_1 - A^TX_1^TX_1 = M^TX_1 - M^TX_1 = 0 \\ &\tilde{M}^TM = \tilde{M}^TX_1A + \tilde{M}^T\tilde{M} = 0A + \tilde{M}^T\tilde{M} = \tilde{M}^T\tilde{M}\end{aligned}$$

Thus $\tilde{M}$ is a matrix where every column is orthogonal to every column of $X_1$. $A \in \mathbb{R}^{p\times q}$ is the matrix which gives the projection of each column of $M$ into the column space of $X_1$. Note that we tacitly assumed that $X$ is not rank deficient, and thus we consider only $M$ (and hence $A$) which are also not rank deficient.

Applying these equalities, and simplifying the matrix form by applying the matrix inversion lemma to the matrix inverse in the top row yields:

$$\hat{\beta} = \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} \left(X_1^T X_1\right)^{-1} X_1^T y - A(\tilde{M}^T \tilde{M})^{-1} \tilde{M}^T y \\ (\tilde{M}^T \tilde{M})^{-1} \tilde{M}^T y \end{pmatrix}$$

$$= \begin{pmatrix} \left(X_1^T X_1\right)^{-1} X_1^T y - A\hat{\beta}_2 \\ \hat{\beta}_2 \end{pmatrix}$$

In this way, we can index the set $M_2$ by $A$ and $\tilde{M}$:

$$M_2 = \{M \in \mathbb{R}^{n \times q} | M = X_1 A + \tilde{M} \text{ for } (A, \tilde{M}) \in \tilde{M}_2\}$$

$$\begin{aligned} \hat{M}_2 &= \{(A, \tilde{M}) \in \mathbb{R}^{p \times q} \times \mathbb{R}^{n \times q} | \tilde{M}^T X_1 = 0,\ (\tilde{M}^T \tilde{M})^{-1} \tilde{M}^T y = \hat{\beta}_2,\ A\hat{\beta}_2 = (X_1^T X_1)^{-1} X_1^T y - \hat{\beta}_1\} \\ &= \{A \in \mathbb{R}^{p \times q} | A\hat{\beta}_2 = (X_1^T X_1)^{-1} X_1^T y - \hat{\beta}_1\} \times \{\tilde{M} \in \mathbb{R}^{n \times q} | \tilde{M}^T X_1 = 0, (\tilde{M}^T \tilde{M})^{-1} \tilde{M}^T y = \hat{\beta}_2\} \\ &= A_2 \times \tilde{M}_2 \end{aligned}$$

We see that the elements of $A_2$ are the matrices where $(X_1^T X_1)^{-1} X_1^T y - \hat{\beta}_1$ is in the column space, and the coordinates are given by $\hat{\beta}_2$. When $q = 1$ and $\hat{\beta}_2 \neq 0$ the set contains exactly one element, and so Party 1 learns:

$$M^T X_1 = A^T X_1^T X_1$$

Hence the off diagonal blocks of the data covariance matrix would be revealed to Party 1. For $q > 1$ this information is not revealed.

For $\tilde{M}_2$, we note that the column space of every element is in the left nullspace of $X_1$ which is an $(n - p)$-dimensional space. Further, if any element of $\hat{\beta}_2$ is zero, then the corresponding column of $\tilde{M}$ is orthogonal to $y$. Columns corresponding to nonzero elements of $\hat{\beta}_2$ are not orthogonal to $y$. If we choose a set of $p$ vectors spanning a $p$-dimensional subspace of the left nullspace of $X_1$, and if we respect the previous condition regarding $y$, we can choose an appropriate scaling to ensure that $\hat{\beta}_2$ gives the coordinates of the projection of $y$ into that subspace.

## 10. References

Blake, I. and Kolesnikov, V. (2004). Strong Conditional Oblivious Transfer and Computing on Intervals. Advances in Cryptology—ASIACRYPT, 2004, 515–529.

Chaudhuri, K. and Monteleoni, C. (2008). Privacy-preserving Logistic Egression. Advances in Neural Information Processing Systems 21 (NIPS 2008), 289–296.

Chen, B.-C., Kifer, K., LeFevre, K., and Machanavajjhala, A. (2009). Privacy-preserving Data Publishing. Foundations and Trends in Databases, 2, 1–167.

Du, W., Han, Y.-S., and Chen, S. (2004). Privacy-preserving Multivariate Statistical Analysis: Linear Regression and Classification. 2004 SIAM International Conference on Data Mining, M.W. Berry, U. Dayal, C. Kamath, and D.B. Skillicorn (eds). Florida: Lak Buena Vista.

Duncan, G.T., Elliot, M., and Salazar-González, J.J. (2001a). Statistical Confidentiality: Principles and Practice. New York: Springer.

Duncan, G.T., Keller-McNulty, S., and Stokes, L. (2001b) Disclosure Risk vs. Data Utility: The R-U Confidentiality Map. Technical Report, National Institute of Statistical Sciences, December, No. 121.

Duncan, G.T. and Pearson, R.W. (1991). Enhancing Access to Microdata while Protecting Confidentiality: Prospects for the Future. Statistical Science, 6, 219–232.

Duncan, G.T. and Stokes, L. (2009). Data Masking for Disclosure Limitation. Wiley Interdisciplinary Reviews: Computational Statistics, 1, 83–92.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. (2006). Our Data Ourselves: Privacy via Distributed Noise Generation. Advances in Cryptology (EUROCRYPT 2006), LNCS Vol. 4004. Berlin: Springer, 486–503.

Dwork, C. (2008). Differential Privacy: A Survey of Results. Theory and Applications of Models of Computation, 1–19.

Fuller, W.A. (1993). Masking Procedures for Microdata. Journal of Official Statistics, 9, 383–406.

Goethals, B., Laur, S., Lipmaa, H., and Mielikainen, T. (2004). On Secure Scalar Product Computation for Privacy-preserving Data Mining. ISISC, 2004.

Goldreich, O. (2004). Foundations of Cryptography: Volume 2 Basic Applications. New York: Cambridge University Press.

Goldreich, O. (1998). Modern Cryptography, Probabilistic Proofs, and Pseudorandomness. New York: Springer.

Guo, C. and Higham, N.J. (2006). A Schur-Newton Method for the Matrix p'th Root and its Inverse. SIAM Journal on Matrix Analysis and Applications, 28, 788–804.

Karr, A.F., Lin, X., Reiter, J.P., and Sanil, A.P. (2005). Secure Regression on Distributed Databases. Journal of Computational and Graphical Statistics, 14, 263–279.

Karr, A.F., Lin, X., Reiter, J.P., and Sanil, A.P. (2006). Secure Analysis of Distributed Databases. In Statistical Methods in Counterterrorism: Game Theory, Modeling, Syndromic Surveillance, and Biometric Authentication, D. Olwell, A.G. Wilson, and G. Wilson (eds). New York: Springer-Verlag, 237–261.

Karr, A.F., Lin, X., Sanil, A.P., and Reiter, J.P. (2009). Privacy-preserving Analysis of Vertically Partitioned Data using Secure Matrix Products. Journal of Official Statistics, 25, 125–138.

Lindell, Y. and Pinkas, B. (2002). Privacy Preserving Data Mining. Journal of Cryptology, 15, 177–206.

Nissim, K. (2008). Private Data Analysis via Output Perturbation. In Privacy-Preserving Data Mining: Models and Algorithms, Charu C. Aggarwal and P.S. Yu (eds). New York: Springer, 385–416.

O'Keefe, C.M. and Good, N.M. (2007). Risk and Utility of Alternative Regression Diagnostics in Remote Analysis Servers. Proceedings of the 55th Session of the International Statistical Institute, August 22–29, Lisbon.

O'Keefe, C.M. and Good, N.M. (2008). A Remote Analysis Server—What Does Regression Output Look Like? Privacy in Statistical Databases, PSD 2008, J. Domingo-Ferrer and Y. Saygin (eds). LNCS 5262. Berlin: Springer, 270–283.

O'Keefe, C.M. and Good, N.M. (2009). Regression Output from a Remote Analysis Server. Data & Knowledge Engineering, 68, 1175–1186.

Paillier, P. (1999). Public-key Cryptosystems based on Composite Degree Residuosity Classes. Advances in Cryptology (EUROCRYPT 1999), J. Stern (ed.). LNCS Vol. 1592. Berlin: Springer-Verlag, 223–238.

Reiter, J.P. (2003). Model Diagnostics for Remote-access Regression Servers. Statistics and Computing, 13, 371–380.

Scannapieco, M., Figotin, I., Bertino, E., and Elmagarmid, A.K. (2007). Privacy Preserving Schema and Data Matching. SIGMOD 2007, 653–664.

Ting, D., Fienberg, S.E., and Trottini, M. (2008). Random Orthogonal Matrix Masking Methodology for Microdata Release. International Journal of Information and Computer Security, 2, 86–105.

Trottini, M., Fienberg, S.E., Makov, U.E., and Meyer, M.M. (2004). Additive Noise and Multiplicative Bias as Disclosure Limitation Techniques for Continuous Microdata: A Simulation Study. Journal of Computational Methods in Sciences and Engineering, 4, 5–16.

Vaidya, J., Zhu, Y., and Clifton, C. (2005). Privacy Preserving Data Mining (Advances in Information Security). New York: Springer-Verlag.

Yao, A.C. (1982). Protocols for Secure Computations. Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 160–164.

Received December 2009
Revised June 2011