Our Data, Ourselves: Privacy via Distributed Noise Generation

Cynthia Dwork[1], Krishnaram Kenthapadi[2,4,5], Frank McSherry[1], Ilya Mironov[1], and Moni Naor[3,4,6]

[1] Microsoft Research, Silicon Valley Campus,
{dwork,mcsherry,mironov}@microsoft.com
[2] Stanford University, kngk@cs.stanford.edu
[3] Weizmann Institute of Science, moni.naor@weizmann.ac.il

Abstract. In this work we provide efficient distributed protocols for generating shares of random noise, secure against malicious participants. The purpose of the noise generation is to create a distributed implementation of the privacy-preserving statistical databases described in recent papers [14, 4, 13]. In these databases, privacy is obtained by perturbing the true answer to a database query by the addition of a small amount of Gaussian or exponentially distributed random noise. The computational power of even a simple form of these databases, when the query is just of the form $\sum_i f(d_i)$, that is, the sum over all rows i in the database of a function f applied to the data in row i, has been demonstrated in [4]. A distributed implementation eliminates the need for a trusted database administrator.

The results for noise generation are of independent interest. The generation of Gaussian noise introduces a technique for distributing shares of many unbiased coins with fewer executions of verifiable secret sharing than would be needed using previous approaches (reduced by a factor of n). The generation of exponentially distributed noise uses two shallow circuits: one for generating many arbitrarily but identically biased coins at an amortized cost of two unbiased random bits apiece, independent of the bias, and the other to combine bits of appropriate biases to obtain an exponential distribution.

1 Introduction

A number of recent papers in the cryptography and database communities have addressed the problem of *statistical disclosure control* – revealing accurate

[4] Part of the work was done in Microsoft Research, Silicon Valley Campus.

[5] Supported in part by NSF Grant ITR-0331640. This work was also supported in part by TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: Cisco, ESCHER, HP, IBM, Intel, Microsoft, ORNL, Qualcomm, Pirelli, Sun and Symantec.

[6] Incumbent of the Judith Kleeman Professorial Chair. Research supported in part by a grant from the Israel Science Foundation.

statistics about a population while preserving the privacy of individuals [1, 2, 15, 11, 14, 5, 6, 4, 13]. Roughly speaking, there are two computational models; in a non-interactive solution the data are somehow sanitized and a "safe" version of the database is released (this may include histograms, summaries, and so on), while in an interactive solution the user queries the database through a privacy mechanism, which may alter the query or the response in order to ensure privacy. With this nomenclature in mind the positive results in the literature fall into three broad categories: non-interactive with trusted server, non-interactive with untrusted server – specifically, via *randomized response*, in which a data holder alters her data with some probability before sending it to the server – and interactive with trusted server. The current paper provides a *distributed* interactive solution, replacing the trusted server with the assumption that strictly fewer than one third of the participants are faulty (we handle Byzantine faults). Under many circumstances the results obtained are of provably better quality (accuracy and conciseness, i.e., number of samples needed for correct statistics to be computed) than is possible for randomized response or other non-interactive solutions [13]. Our principal technical contribution is in the cooperative generation of shares of noise sampled from in one case the Binomial distribution (as an approximation for the Gaussian) and in the second case the Poisson distribution (as an approximation for the exponential).

Consider a database that is a collection of rows; for example, a row might be a hospital record for an individual. A query is a function f mapping rows to the interval $[0, 1]$. The *true answer* to the query is the value obtained by applying f to each row and summing the results. By responding with an appropriately perturbed version of the true answer, privacy can be guaranteed. The computational power of this provably private "noisy sums" primitive is demonstrated in Blum et al. [4], where it was shown how to carry out accurate and privacy-preserving variants of many standard data mining algorithms, such as k-means clustering, principal component analysis, singular value decomposition, the perceptron algorithm, and anything learnable in the statistical queries (STAT) learning model[4].

Although the powerful techniques of secure function evaluation [25, 17] may be used to emulate any privacy mechanism, generic computations can be expensive. The current work is inspired by the combination of the simplicity of securely computing sums and the power of the noisy sums. We provide *efficient* methods allowing the parties holding their own data to act *autonomously* and without a central trusted center, while simultaneously preventing malicious parties from interfering with the utility of the data.

The approach to decentralization is really very simple. For ease of exposition we describe the protocol assuming that every data holder participates in every query and that the functions f are predicates. We discuss relaxations of these assumptions in Section 5.

[4] This was extended in [13] to handle functions f that operate on the database as a whole, rather than on individual rows of the database.

Structure of ODO (Our Data, Ourselves) Protocol

1. **Share Summands:** On query f, the holder of d_i, the data in row i of the database, computes $f(d_i)$ and shares out this value using a *non-malleable verifiable secret sharing scheme* (see Section 2), $i = 1, \ldots, n$. The bits are represented as 0/1 values in GF(q), for a large prime q. We denote this set $\{0,1\}_{\mathrm{GF}(q)}$ to make the choice of field clear.
2. **Verify Values:** Cooperatively verify that the shared values are *legitimate* (that is, in $\{0,1\}_{\mathrm{GF}(q)}$, when f is a predicate).
3. **Generate Noise Shares:** Cooperatively generate shares of appropriately distributed random noise.
4. **Sum All Shares:** Each participant adds together all the shares that it holds, obtaining a share of the noisy sum $\sum_i f(d_i) + \textbf{noise}$. All arithmetic is in GF(q).
5. **Reconstruct:** Cooperatively reconstruct the noisy sum using the reconstruction technique of the verifiable secret sharing scheme.

Our main technical work is in Step 3. We consider two types of noise, *Gaussian* and *scaled symmetric exponential*. In the latter distribution the probability of being at distance $|x|$ from the mean is proportional to $\exp(-|x|/R)$, the scale R determining how "flat" the distribution will be. In our case the mean will always be 0. Naturally, we must approximate these distributions using finite-precision arithmetic. The Gaussian and exponential distributions will be approximated, respectively, by the Binomial and Poisson distributions.

The remainder of this paper is organized as follows. In Section 2 we review those elements from the literature necessary for our work, including definitions of randomness extractors and of privacy. In Sections 3 and 4 we discuss implementations of Step 3 for Gaussian and Exponential noise, respectively. Finally, various generalizations of our results are mentioned in Section 5.

2 Cryptographic and Other Tools

Model of Computation. We assume the standard synchronous model of computation in which n processors communicate by sending messages via point-to-point channels and up to $t \leq \lfloor \frac{n-1}{3} \rfloor$ may fail in an arbitrary, Byzantine, adaptive fashion. If the channels are secure, then the adversary may be computationally unbounded. However, if the secure channels are obtained by encryption then we assume the adversary is restricted to probabilistic polynomial time computations.

We will refer to several well-known primitive building blocks for constructing distributed protocols: Byzantine Agreement [20], Distributed Coin Flipping [22], Verifiable Secret Sharing (VSS) [8], Non-Malleable VSS, and Secure Function Evaluation (SFE) [18].

A VSS scheme allows any processor distribute shares of a secret, which can be verified for consistency. If the shares verify, the honest processors can always reconstruct the secret regardless of the adversary's behavior. Moreover, the faulty processors by themselves cannot learn any information about the secret. A non-malleable VSS scheme ensures that the values shared by a non-faulty processor are completely independent of the values shared by the other processors; even exact copying is prevented.

Throughout the paper we will use the following terminology. Values that have been shared and verified, but not yet reconstructed, are said to be *in shares*. Values that are publicly known are said to be *public*.

A *randomness extractor* [21] is a method of converting a non-uniform input distribution into a near-uniform distribution on a smaller set. In general, an extractor is a randomized algorithm, which additionally requires a perfect source of randomness, called the seed. Provided that the input distribution has sufficiently high min-entropy, a good extractor takes a short seed and outputs a distribution that is statistically close to the uniform. Formally,

Definition 1. *Letting the min-entropy of a distribution $\mathcal{D}$ on X be denoted $H_\infty(\mathcal{D}) = -\log \max_{x\in X} \mathcal{D}(x)$, a function $F\colon X \times Y \mapsto \{0,1\}^n$ is a (δ, ϵ, n)-*extractor*, if for any distribution $\mathcal{D}$ on X such that $H_\infty(\mathcal{D}) > \delta$,*

$$|\{F(x,y)\colon x \in_{\mathcal{D}} X, y \in_U Y\} - U_n| < \epsilon,$$

where $|\cdot|$ is the statistical distance between two distributions, U_n is the uniform distribution on $\{0,1\}^n$, and $x \in_{\mathcal{D}} X$ stands for choosing $x \in X$ according to $\mathcal{D}$.

Optimal extractors can extract $n = \delta - 2\log(1/\epsilon) + O(1)$ nearly-random bits with the seed length $O(\log |X|)$ (see [23] for many constructions matching the bound).

While in general the presence of a truly random seed cannot be avoided, there exist *deterministic* extractors (i.e. without Y) for sources with a special structure [7, 9, 24, 19, 16] where the randomness is concentrated on k bits and the rest are fixed. Namely,

Definition 2. *A distribution $\mathcal{D}$ over $\{0,1\}^N$ is an (N,k) oblivious bit-fixing source if there exists $S = \{i_1, \ldots, i_k\} \subset [N]$, such that $X_{i_1}, \ldots, X_{i_k}$ are uniformly distributed in $\{0,1\}^k$, and the bits outside S are constant.*

For any (N,k) bit-fixing source and any constant $0 < \gamma < 1/2$ Gabizon et al. [16] give an explicit deterministic (k, ϵ)-extractor that extracts $m = k - N^{1/2+\gamma}$ bits of entropy with $\epsilon = 2^{-\Omega(n^\gamma)}$ provided that $k \gg \sqrt{N}$. In our case $N = 2n$ (n is the number of participants), and strictly more than 2/3 of the input bits will be good. Thus, $k > 2N/3$, and so we extract more than $N/2 = n$ high quality bits by taking $\gamma < 1/2$.

A *privacy mechanism* is an interface between a user and data. It can be interactive or non-interactive.

Assume the database consists of a number n of rows, $d_1, \ldots, d_n$. In its simplest form, a query is a predicate $f : \mathit{Rows} \to \{0,1\}$. In this case, the true answer is simply $\sum_i f(d_i)$. Slightly more generally, f may map $[n] \times \mathit{Rows} \to [0,1]$, and the true answer is $\sum_i f(i, d_i)$. Note that we are completely agnostic about the domain *Rows*; rows can be Boolean, integers, reals, tuples thereof, or even strings or pictures.

A mechanism gives ϵ-*indistinguishability* [13] if for any two data sets that differ on only one row, the respective output random variables (query responses) τ and τ' satisfy for all sets S of responses:

$$\Pr[\tau \in S] \leq \exp(\epsilon) \times \Pr[\tau' \in S] \ . \tag{1}$$

This definition ensures that seeing τ instead of τ' can only increase the probability of any event by at most a small factor. As a consequence, there is little incentive for any one participant to conceal or misrepresent her value, as so doing could not substantially change the probability of any event.

Similarly, we say a mechanism gives δ-*approximate* ϵ-*indistinguishability* if for outputs τ and τ' based, respectively, on data sets differing in at most one row,

$$\Pr[\tau \in S] \leq \exp(\epsilon) \times \Pr[\tau' \in S] + \delta \ .$$

The presence of a non-zero δ permits us to relax the strict relative shift in the case of events that are not especially likely. We note that it is inappropriate to add non-zero δ to the statement of ϵ-indistinguishability in [13], where the sets S are constrained to be singleton sets.

Historically, the first strong positive results for output perturbation added noise drawn from a Gaussian distribution, with density function $\Pr[x] \propto \exp(-x^2/2R)$. A slightly different definition of privacy was used in [14, 4]. In order to recast those results in terms of indistinguishability, we show in Section 2.1 that the addition of Gaussian noise gives δ-approximate ϵ-indistinguishability for the noisy sums primitive when $\epsilon > [\log(1/\delta)/R]^{1/2}$. In a similar vein, Binomial noise, where n tosses of an unbiased ± 1 coin are tallied and divided by 2, also gives δ-approximate ϵ-indistinguishability so long as the number of tosses n is at least $64 \log(2/\delta)/\epsilon^2$.

Adding, instead, exponential noise results in a mechanism that can ensure ϵ-indistinguishability (that is, $\delta = 0$) [4, 13]. If the noise is distributed as $\Pr[x] \propto \exp(-|x|/R)$, then the mechanism gives $1/R$-indistinguishability (cf. $\epsilon > [\log(1/\delta)/R]^{1/2}$ for Gaussian noise). Note that although the Gaussian noise is more tightly concentrated around zero, giving somewhat better accuracy for any given choice of ϵ, the exponential noise allows $\delta = 0$, giving a more robust solution.

2.1 Math for Gaussians and Binomials

We extend the results in [13] by determining the values of ϵ and δ for the Gaussian and Binomial distributions for which the noisy sums primitive yields δ-approximate ϵ-indistinguishability. Consider an output τ on a database D and query f. Let $\tau = \sum_i f(i, d_i) + \textbf{noise}$, so replacing D with D' differing only in one row changes the summation by at most 1. Bounding the ratio of probabilities that τ occurs with inputs D and D' amounts to bounding the ratio of probabilities that **noise** $= x$ and **noise** $= x+1$, for the different possible ranges of values for x. Thus, we first determine the largest value of x such that a relative bound of $\exp(\epsilon)$ holds, and then integrate the probability mass outside of this interval.

Recall the Gaussian density function: $p(x) \propto \exp(-x^2/2R)$. The ratio of densities at two adjacent integral points is

$$\frac{\exp(-x^2/2R)}{\exp(-(x+1)^2)/2R} = \exp(x/R + 1/2R).$$

This value remains at most $\exp(\epsilon)$ until $x = \epsilon R - 1/2$. Provided that $R \geq 2\log(2/\delta)/\epsilon^2$ and that $\epsilon \leq 1$, the integrated probability beyond this point will be at most

$$\Pr[x > \epsilon R - 1/2] \;\leq\; \frac{\exp(-(\epsilon R)^2/2R)}{(\epsilon R)\sqrt{\pi}} \;\leq\; \delta\ .$$

As a consequence, we get δ-approximate ϵ-indistinguishability when R is at least $2\log(2/\delta)/\epsilon^2$.

For the Binomial noise with bias 1/2, whose density at $n/2 + x$ is

$$\Pr[n/2 + x] = \binom{n}{n/2+x} 1/2^n\ ,$$

we see that the relative probabilities are

$$\frac{\Pr[n/2+x]}{\Pr[n/2+x+1]} = \frac{n/2+x+1}{n/2-x}\ .$$

So long as x is no more than $\epsilon n/8$, this should be no more than $(1+\epsilon) < \exp(\epsilon)$. Of course, a Chernoff bound tells us that for such x the probability that a sample exceeds it is

$$\begin{aligned}\Pr[y > n/2 + \epsilon n/8] &= \Pr[y > (1+\epsilon/4)n/2] \\ &\leq \exp(-(\epsilon^2 n/64)).\end{aligned}$$

We get δ-approximate ϵ-indistinguishability so long as n is chosen to be at least $64\log(2/\delta)/\epsilon^2$. This exceeds the estimate of the Gaussian due to approximation error, and general slop in the analysis, though it is clear that the form of the bound is the same.

2.2 Adaptive Query Sequences

One concern might be that after multiple queries, the values of ϵ and δ degrade in an inelegant manner. We now argue that this is not the case.

Theorem 1. *A mechanism that permits T adaptive interactions with a δ-approximate ϵ-indistinguishable mechanism ensures δT-approximate ϵT-indistinguishability.*

Proof. We start by examining the probability that the transcript, written as an ordered T-tuple, lands in a set S.

$$\Pr[x \in S] = \prod_{i \le T} \Pr[x_i \in S_i | x_1, \ldots, x_{i-1}].$$

As the noise is independent at each step, the conditioning on $x_1, \ldots, x_{i-1}$ only affects the predicate that is asked. As a consequence, we can substitute

$$\prod_{i \le T} \Pr[x_i \in S_i | x_1, \ldots, x_{i-1}] \le \prod_{i \le T} \left(\exp(\epsilon) \times \Pr[x'_i \in S_i | x_1, \ldots, x_{i-1}] + \delta\right).$$

If we look at the additive contribution of each of the δ terms, of which there are T, we notice that they are only ever multiplied by probabilities, which are at most one. Therefore, each contributes at most an additive δ.

$$\begin{aligned}\prod_{i \le T} \Pr[x_i \in S_i | x_1, \ldots, x_{i-1}] &\le \prod_{i \le T} \left(\exp(\epsilon) \times \Pr[x'_i \in S_i | x_1, \ldots, x_{i-1}]\right) + \delta T \\ &= \exp(\epsilon T) \times \prod_{i \le T} \left(\Pr[x'_i \in S_i | x_1, \ldots, x_{i-1}]\right) + \delta T \\ &= \exp(\epsilon T) \times \Pr[x' \in S] + \delta T\ .\end{aligned}$$

The proof is complete. □

3 Generating Gaussian Noise

Were we not concerned with malicious failures, a simple approach would be to have each participant i perturb $f(d_i)$ by sampling from a Gaussian with mean zero and variance $\frac{3}{2}\mathbf{var}/n$, where $\mathbf{var}$ is a lower bound on the variance needed for preserving privacy (see Section 2). The perturbed values would be shared out and the shares summed, yielding $\sum_i f(d_i) + \mathbf{noise}$ in shares. Since, as usual in the Byzantine literature, we assume that at least $2/3$ of the participants will survive, the total variance for the noise would be sufficient (but not excessive). However, a Byzantine processor might add an outrageous amount of noise to its share, completely destroying the integrity of the results. We now sketch the main ideas in our solution for the Byzantine case.

Recall that the goal is for the participants to obtain the noise in shares. As mentioned earlier, we will approximate the Gaussian with the Binomial distribution, so if the participants hold shares of sufficiently many unbiased coins they can sum these to obtain a share of (approximately) correctly generated noise. Coin flipping in shares (and otherwise) is well studied, and can be achieved by having each participant non-malleably verifiably share out a value in GF(2), and then locally summing (in GF(2)) the shares from all n secret sharings.

This suggests a conceptually straightforward solution: Generate many coins in shares, convert the shares from GF(2) to shares of values in a large field GF(q) (or to shares of integers), and then sum the shares. In addition to the conversion costs, the coins themselves are expensive to generate, since they require $\Omega(n)$ executions of verifiable secret sharing per coin, which translates into $\Omega(nc)$ secret sharings for c coins[5]. To our knowledge, the most efficient scheme for generating random bits is due to Damgård et al. [10], which requires n sharings and two multiplications per coin.

We next outline a related but less expensive solution which at no intermediate or final point uses the full power of coin-flipping. The solution is cost effective when c is sufficiently large, i.e., $c \in \Omega(n)$. As a result, we will require only $\Omega(c)$ sharings of values in GF(2) when $c \in \Omega(n)$. Let n denote both the number of players and the desired number of coins[6].

1. Each player i shares a random bit by sharing out a value $b_i \in \{0,1\}_{\mathrm{GF}(q)}$, using a non-malleable verifiable secret sharing scheme, where q is sufficiently large, and engages in a simple protocol to prove that the shared value is indeed in the specified set. (The verification is accomplished by distributively checking that $x^2 = x$ for each value x that was shared, in parallel. This is a single secure function evaluation of a product, addition of two shares, and a reconstruction, for each of the n bits b_i.) This gives a sequence of low-quality bits in shares, as some of the shared values may have been chosen adversarially. (Of course, the faulty processors know the values of the bits they themselves have produced.)
2. Now, suppose for a moment that we have a public source of unbiased bits, c_1, c_2,..., c_n. By XORing together the corresponding b's and c's, we can transform the low quality bits b_i (in shares) into high-quality bits $b_i \oplus c_i$, in shares. (Again, the faulty processors know the values of the (now randomized) bits they themselves have produced.) The XORing is simple: if $c_i = 0$ then the shares of b_i remain unchanged. If $c_i = 1$ then each share of b_i is replaced by one minus the original share.
3. Replace each share s by $2s - 1$, all arithmetic in GF(q). This maps shares of 0 to shares of -1, and shares of 1 to (different) shares of 1.
4. Finally, each participant sums her shares to get a share of the Binomial noise.

We now turn to the generation of the c_i. Each participant randomly chooses and non-malleably verifiably shares out two bits, for a total of $2n$ low-quality bits in shares. This is done in GF(2), so there is no need to check legitimacy. Let the low-quality source be $b'_1, b'_2, \ldots, b'_{2n}$. The b'_i are then reconstructed, so that they become public. The sequence $b'_1 b'_2 \ldots b'_{2n}$ is a bit-fixing source: some of the bits are biased, but they are independent of the other bits (generated by the good participants) due to the non-malleability of the secret sharing. The main advantage of such a source is that it is possible to apply a *deterministic*

[5] When a single player shares out many values (not the case for us), the techniques of Bellare, Garay, and Rabin [3] can be used to reduce the cost of verifying the shared out values. The techniques in [3] complement ours; see Section 5.

[6] If the desired number of coins is $o(n)$, we can generate $\Theta(n)$ coins and keep the unused ones in reserve for future executions of the protocol. If $m \gg n$ coins are needed, each processor can run the protocol m/n times.

extractor on those bits and have the output be very close to uniform. Since the bits $b'_1 \dots b'_{2n}$ are public, this extraction operation can be done by each party individually with no additional communication. In particular we may use, say, the currently best known deterministic extractor of [16], which produces a number $m > n$ of nearly unbiased bits. The outputs of the extractor are our public coins $c_1 \dots c_m$.

The principal costs are the multiplications for verifying membership in $\{0,1\}_{\mathrm{GF}(q)}$ and the executions of verifiable secret sharing. Note that all the verifications of membership are performed simultaneously, so the messages from the different executions can be bundled together. The same is true for the verifications in the VSS. The total cost of the scheme is $\Theta(n)$ multiplications and additions in shares, which can be all done in a constant number of rounds.

4 Generating Exponential Noise

Recall that in the exponential distribution the probability of obtaining a value at distance $|x|$ from the mean is proportional to $\exp(-|x|/R)$, where R is a scaling factor. For the present discussion we take $R = 1/(\ln 2)$, so that $\exp(-|x|/R) = 2^{-|x|}$. We approximate the exponential distribution with the Poisson distribution. An intuitively simple approach is to generate a large number of unbiased[7] random bits in shares, and then find (in shares) the position ℓ of the first 1. The value returned by this noise generation procedure is $\pm\ell$ (we flip one additional bit to get the sign). If there is no 1, then the algorithm fails, so the number of bits must be sufficiently large that this occurs with negligible probability. All the computation must be done in shares, and we can't "quit" once a 1 has been found (this would be disclosive). This "unary" approach works well when $R = 1/(\ln 2)$ and the coins are unbiased. For much larger values of R, the case in high-privacy settings, the coins need to be heavily biased toward 0, flattening the curve. This would mean more expected flips before seeing a 1, potentially requiring an excessive number of random bits.

Instead, we take advantage of the special structure of the exponential distribution, and see that we can generate the *binary* representation of an exponential variable using a number of coins that is independent of the bias. Let us return to the question of the location ℓ of the first 1 in a sequence of randomly generated bits. We can describe ℓ one bit at a time by answering the following series of questions:

1. What is the parity of ℓ? That is, $\ell = 2^i$ for some $i \geq 0$? (We begin counting the positions at 0, so that ℓ will be the number of 0's preceding the first 1.)
2. Is ℓ in the left half or the right half of a block of 4 positions, i.e., is it the case that $2^2 i \leq \ell < 2^2 i + 2$ for some $i \geq 0$?
3. Is ℓ in the left half or the right half of a block 8 positions, i.e., is it the case that $2^3 i \leq \ell < 2^3 i + 2^2$ for some $i \geq 0$?
4. And so on.

[7] For values of $R \neq 1/(\ln 2)$ we would need to use biased bits.

We generate the distribution of ℓ "in binary" by generating the answers to the above questions. (For some fixed d we simply assume that $\ell < 2^d$, so only a finite number of questions need be answered.)

To answer the questions, we need to be able to generate biased coins. The probability that ℓ is even (recall that we begin counting positions with 0) is $(1/2)\sum_{i=0}^{\infty}(2^{-2i})$. Similarly, the probability that ℓ is odd is $(1/2)\sum_{i=0}^{\infty}(2^{-(2i+1)})$. Thus,

$$\Pr[\ell \text{ odd}] = (1/2)\Pr[\ell \text{ even}].$$

Since the two probabilities sum to 1, the probability that ℓ is even is $2/3$. Similar analyses yield the necessary biases for the remaining questions.

The heart of the technical argument is thus to compute coins of arbitrary bias in shares in a manner that consumes on average a constant number of unbiased, completely unknown, random bits held in shares. We will construct and analyze a shallow circuit for this. In addition, we will present two incomparable probabilistic constructions. In any distributed implementation these schemes would need to be implemented by general secure function evaluation techniques. The circuits, which only use Boolean and finite field arithmetic, allow efficient SFE implementation.

4.1 Poisson Noise: The Details

In this section we describe several circuits for generating Poisson noise. The circuits will take as input random bits (the exact number depends on the circuit in question). In the distributed setting, the input would be the result of a protocol that generates (many) unbiased bits in shares. The circuit computation would be carried out in a distributed fashion using secure function evaluation, and would result in *many* samples, in shares, of **noise** generated according to the Poisson distribution. This fits into the high-level ODO protocol in the natural way: shares of the noise are added to the shares of $\sum_i f(i, d_i)$ and the resulting noisy sum is reconstructed.

For the remainder of this section, we let n denote the number of coins to be generated. It is unrelated to the number of participants in the protocol.

Recall the discussion in the Introduction of the exponential distribution, where $\Pr[x] \propto \exp(-|x|/R)$. Recall that one interpretation is to flip a (possibly biased) coin until the first 1 is seen, and then to output the number ℓ of 0's seen before the 1 occurs. Recall also that instead of generating ℓ in unary, we will generate it in binary.

We argue that the bits in the binary representation of the random variable ℓ are independent, and moreover we can determine their biases analytically. To see the independence, consider the distribution of the ith bit of ℓ:

$$\ell_i = \begin{cases} 0 \text{ w.p. } \Pr[0 \times 2^i \le \ell < 1 \times 2^i] + \Pr[2 \times 2^i \le \ell < 3 \times 2^i] + \ldots \\ 1 \text{ w.p. } \Pr[1 \times 2^i \le \ell < 2 \times 2^i] + \Pr[3 \times 2^i \le \ell < 4 \times 2^i] + \ldots \end{cases}$$

Notice that corresponding terms in the two summations, eg $\Pr[0 \times 2^i \le \ell < 1 \times 2^i]$ and $\Pr[1 \times 2^i \le \ell < 2 \times 2^i]$, are directly comparable; the first is exactly $\exp(2^i/R)$

times the second. This holds for every corresponding pair in the sums, and as such the two sums share the same ratio. As the two sum must total to one, we have additionally that

$$1 - \Pr[\ell_i] = \exp(2^i/R) \times \Pr[\ell_i] \ .$$

Solving, we find that

$$\Pr[\ell_i] = 1/(1 + \exp(2^i/R)) \ .$$

Recall as well that the observed ratio applied equally well to each pair of intervals, indicating that the bias is independent of the more significant bits. The problem of producing an exponentially distributed ℓ is therefore simply a matter of flipping a biased coin for each bit of ℓ. The circuit we will construct will generate many ℓ's according to the desired distribution, at an expected low amortized cost (number of input bits) per bit position in the binary expansion of ℓ. The circuit is a collection of circuits, each for one bit position, with the associated bias hard-wired in. It suffices therefore to describe the circuitry for one of these smaller circuits (Section 4.3). We let p denote the hard-wired bias.

A well-known technique for flipping a single coin of arbitrary bias p is to write p in binary, examine random bits until one differs from the corresponding bit in p, and then emit the complement of the random bit. To achieve a high fidelity to the original bias p, a large number d of random bits must be available. However, independent of p, the expected number of random bits consumed is at most 2. This fact will be central to our constructions.

In the sequel we distinguish between unbiased *bits*, which are inputs to the algorithm, and the generated, biased, *coins*, which are the outputs of the algorithm.

4.2 Implementation Details: Finite Resources

With finite randomness we will not be able to perfectly emulate the bias of the coins. Moreover, the expectation of higher order bits in the binary representation of ℓ diminishes at a doubly exponential rate (because the probability that $\ell \geq 2^i$ is exponentially small in 2^i), quickly giving probabilities that simply can not be achieved with any fixed amount of randomness.

To address these concerns, we will focus on the statistical difference between our produced distribution and the intended one. The method described above for obtaining coins with arbitrary bias, truncated after d bits have been consumed, can emulate any biased coin within statistical difference at most 2^{-d}. Accordingly, we set all bits of sufficiently high order to zero, which will simplify our circuit. The remaining output bits – let us imagine there are k of them – will result in a distribution whose statistical difference is at most $k2^{-d}$ from the target distribution. We note that by trimming the distribution to values at most 2^d in magnitude, we are introducing an additional error, but one whose statistical difference is quite small. There is an $\exp(-2^d/R)$ probability mass outside the $[-2^d, 2^d]$ interval that is removed and redistributed inside the interval. This results in an additional $2\exp(-2^d/R)$ statistical difference that should be incorporated into δ. For clarity, we absorb this term into the value k.

Using our set of coins with statistical difference at most $k2^{-d}$ from the target distribution, we arrive at a result akin to (1), though with an important difference. For response variables τ and τ' as before (based on databases differing it at most one row),

$$\forall S \subseteq U: \quad \Pr[\tau \in S] \leq \Pr[\tau' \in S] \times \exp(1/R) + k2^{-d} .$$

As before, the probability of any event increases by at most a factor of $\exp(1/R)$, but now with an additional additive $k2^{-d}$ term. This term is controlled by the parameter d, and can easily be made sufficiently small to allay most concerns.

We might like to remove the additive $k2^{-d}$ term, which changes the nature of the privacy guarantee. While this seems complicated at first, notice that it is *possible* to decrease the relative probability associated with each output coin arbitrarily, by adding more bits (that is, increasing d). What additional bits can not fix is our assignment of zero probability to noise values outside the permitted range (i.e., involving bits that we do not have circuitry for).

One pleasant resolution to this problem, due to Adam Smith, is to constrain the output range of the sum of noise plus signal. If the answer plus noise is constrained to be a k-bit number, and conditioned on it lying in that range the distribution looks exponential, the same privacy guarantees apply. Guaranteeing that the output will have only k bits can be done by computing the sum of noise and signal using $k+1$ bits, and then if there is overflow, outputting the noise-free answer. This increases the probability that $\mathbf{noise} = 0$ by a relatively trivial amount, and ensures that the output space is exactly that of k-bit numbers.

4.3 A Circuit for Flipping Many Biased Coins

We are now ready to construct a circuit for flipping a large number of independent coins with common bias. By producing many ($\Omega(n)$) coins at once, we could hope to leverage the law of large numbers and consume, with near certainty, a number of input bits that is little more than $2n$ and depends very weakly on d. For example, we could produce the coins sequentially, consuming what randomness we need and passing unused random bits on to the next coin. The circuit we now describe emulates this process, but does so in a substantially more parallel manner.

The circuit we construct takes 2^i unbiased input bits and produces 2^i output coins, as well as a number indicating how many of the coins are actually the result of the appropriate biased flips. That is, it is unlikely that we will be able to produce fully 2^i coins, and we should indicate how many of the coins are in fact valid. The construction is hierarchical, in that the circuit that takes 2^i inputs will be based on two level $i-1$ circuits, attached to the first and second halves of its inputs.

To facilitate the hierarchical construction, we augment the outputs of each circuit with the number of bits at the end of the 2^i that were consumed by the coin production process, but did not diverge from the binary representation of p. Any process that wishes to pick up where this circuit has left off should start under the assumption that the first coin is in fact this many bits into its production. For example, if this number is r then the process should begin by comparing the next random bit to the $(r+1)$st bit in the expansion of p. Bearing this in mind, we "bundle" d copies of this circuit together, each with a different assumption about the initial progress of the production of their first coin.

For each value $1 \leq j \leq d$ we need to produce a vector of 2^i coins c_j, a number of coins n_j, and d_j, a measure of progress towards the last coin. We imagine that we have access to two circuits of one level lower, responsible for the left and right half of our 2^i input bits, and whose corresponding outputs are superscripted by L and R. Intuitively, for each value of j we ask the left circuit for d_j^L, which we use to select from the right circuit. Using index j for the left circuit and d_j^L for the right circuit, we combine the output coins using a shift of n_j^L to align them, and add the output counts n_j^L and $n_{d_j^L}^R$. We simply pass $d_{d_j^L}^R$ out as the appropriate value for d_j.

$$\begin{aligned}
c_j &= c_j^L \mid (c_{d_j^L}^R >> n_j^L) \\
n_j &= n_j^L + n_{d_j^L}^R \\
d_j &= d_{d_j^L}^R
\end{aligned}$$

The operation of subscripting is carried out using a multiplexer, and shifts, bitwise ors, and addition are similarly easily carried out in logarithmic depth.

The depth of each block is bounded by $\Theta(\log(nd))$, with the size bounded by $\Theta(2^i d(\log(n) + d)$, as each of d outputs must multiplex d possible inputs (taking $\Theta(d)$ circuitry) and then operate on them (limited by $\Theta(\log(n)2^i)$ for the barrel shifter). All told, the entire circuit has depth $\Theta(\log(nd)^2)$, with size $\Theta(nd(\log(n) + d)\log(n))$.

4.4 Probabilistic Constructions with Better Bounds

We describe two probabilistic constructions of circuits that take as input unbiased bits and produce as output coins of arbitrary, *not necessarily identical*, bias. Our first solution is optimal in terms of depth ($\Theta(\log d)$) but expensive in the gate count. Our second solution dramatically decreases the number of gates, paying a modest price in depth ($O(\log(n+d))$) and a logarithmic increase in the number of input bits.

A module common to both constructions is the comparator – a circuit that takes two bit strings $b_1, \ldots, b_d$ and $p^{(1)} \ldots p^{(d)}$ and outputs 0 if and only if the first string precedes the second string in the lexicographic order. Equivalently, the comparator outputs $\bar{b}_i$, where i is the index of the earliest occurrence 1 in the sequence $b_1 \oplus p^{(1)}, \ldots, b_d \oplus p^{(d)}$, or 1 if the two strings are equal. Based on this observation, a circuit of depth $\Theta(\log d)$ and size $\Theta(d)$ can be designed easily. Notice that the result of comparison is independent of the values of the strings beyond the point of divergence.

Brute Force Approach. Assume that we have nd independent unbiased bits $b_i^{(j)}$, for $1 \le i \le n$ and $1 \le j \le d$. To flip n independent coins, each with its own bias p_i, whose binary representation is $0.p_i^{(1)} \dots p_i^{(d)}$, we run n comparators in parallel on inputs $(b_1^{(1)}, \dots, b_1^{(d)}, p_1^{(1)}, \dots, p_1^{(d)})$, ..., $(b_n^{(1)}, \dots, b_n^{(d)}, p_n^{(1)}, \dots, p_n^{(d)})$.

Our goal is to get by with many fewer than nd unbiased input bits of the brute force approach, since each of these requires an unbiased bit in shares. Intuitively, we may hope to get away with this because, as mentioned previously, the average number of bits consumed per output coin is 2, independent of the bias of the coin. Let c_i for $1 \le i \le n$ be the smallest index where $b_i^{(c_i)} \ne p_i^{(c_i)}$, and $d+1$ if the two strings are equal. The number c_i corresponds to the number of bits "consumed" during computation of the ith coin. Let $C = \sum_{i=1}^{n} c_i$. On expectation $E[C] = 2n$, and except with a negligible probability $C < 4n$.

Rather than having the set $\{b_i^{(j)}\}_{i,j}$ be given as input (too many bits), we will compute the set $\{b_i^{(j)}\}_{i,j}$ from a much smaller set of input bits. The construction will ensure that the *consumed* bits are independent except with negligible probability. Let the number of input bits be D, to be chosen later.

We will construct the circuit probabilistically. Specifically, we begin by choosing nd binary vectors $\{r_i^{(j)}\}_{i,j}$, $1 \le i \le n$ and $1 \le j \le d$, uniformly from $\{0,1\}^D$ to be hard-wired into the circuit. Let $b \in_R \{0,1\}^D$ be the uniformly chosen random input to the circuit.

The circuit computes the inner products of each of the hard-wired vectors $r_i^{(j)}$ with the input b. Let $b_i^{(j)} = \langle r_i^{(j)}, b \rangle$ denote the resulting bits. These are the $\{b_i^{(j)}\}_{i,j}$ we will plug into the brute force approach described above. Note that although much randomness was used in defining the circuit, the input to the circuit requires only D random bits.

Although the nd vectors are not linearly independent, very few of them – $O(n)$ – are actually used in the computation of our coins, since with overwhelming probability only this many of the $b_i^{(j)}$ are actually consumed. A straightforward counting argument therefore shows that the set of vectors actually used in generating consumed bits will be linearly independent, and so the coins will be mutually independent.

We claim that if $D > 4C$, then the consumed bits are going to be independent with high probability. Conditional on the sequence $c_1, \dots, c_n$, the vectors $r_i^{(j)}$ for $1 \le i \le n$ and $1 \le j \le c_i$ are independent with probability at least $1 - C2^{C-D} < 1 - 2^{-2C}$, where the probability space is the choice of the r's. For fixed C the number of possible $c_1, \dots, c_n$ is at most $\binom{C}{n} < 2^C$. Hence the probability that for some $C < 4n$ and some $c_1, \dots, c_n$, such that $c_1 + \dots + c_n = C$ the vectors $r_i^{(j)}$ are linearly independent is at least than $1 - 4n2^{-C}$. Finally, we observe that if the vectors are linearly independent, the bits $b_i^{(j)}$ are independent as random variables. The depth of this circuit is $\Theta(\log D)$, which is the time it takes to compute the inner product of two D-bit vectors. Its gate count is $\Theta(ndD)$, which is clearly suboptimal.

Using low weight independent vectors: Our second solution dramatically decreases the number of gates by reducing the weight (the number of non-zero elements) of the vectors r from the expected value $D/2$ to $s^2\lceil\log(n+1)\rceil$, where s is a small constant. To this end we adopt the construction from [12] that converts an expander-like graph into a set of linearly independent vectors.

The construction below requires a field with at least nd non-zero elements. Let $\nu = \lceil\log(nd+1)\rceil$. We use $\mathrm{GF}(2^\nu)$, representing its elements as ν-bit strings.

Consider a bipartite graph G of constant degree s connecting sets $L = \{u_1, \ldots, u_n\}$, where the u's are distinct field elements, and $R = \{1, \ldots, \Delta\}$. The degree s can be as small as 3. Define matrix M of size $n \times s\Delta$ as follows: if $(u_i, \tau) \in G$, the elements $M[i][s(\tau-1), s(\tau-1)+1, \ldots, s\tau-1] = u_i, u_i^2, \ldots, u_i^s$, and $(0, \ldots, 0)$ (s zeros) otherwise. Thus, each row of the matrix has exactly s^2 non-zero elements.

For any set $S \subseteq L$, let $\Gamma(S) \subseteq R$ be the set of neighbors of S in G. The following claim is easily obtained from the proof of Lemma 5.1 in [12]. It says that if for a set of vertices $T \in L$ all of T's subsets are sufficiently expanding, then the rows of M corresponding to vertices in T are linearly independent.

Theorem 2. *Let $T \subseteq L$ be any set for which $\forall S \subseteq T$, $|\Gamma(S)| > (1 - \frac{1}{s+1})|S|$. Then the set of vectors $\{M[u] \colon u \in T\}$ is linearly independent.*

Consider a random bipartite graph with nd/ν elements in one class and $2C$ elements in the other. Associate the elements from the first class with bits $b_i^{(j)}$'s, grouped in ν-tuples. Define the bits as the results of the inner product of the corresponding rows of the matrix M from above with the input vector of length $2s^2C$ that consists of random elements from $\mathrm{GF}(2^\nu)$. Observe that the random graph G satisfies the condition of Theorem 2 for all sets of size less than C with high probability if $C > (nd/\nu)^{1/(s-1)}$.

The depth of the resulting circuit is $\Theta(\log(n+d))$, the gate count is $\Theta(nds^2\log(n+d))$, and the size of the input is $2n\log(n+d)$.

5 Generalizations

In this section we briefly discuss several generalizations of the basic scheme.

5.1 Alternatives to Full Participation

The main idea is to use a set of facilitators, possibly a very small set, but one for which we are sufficiently confident that fewer than one third of the members are faulty. Let $\mathcal{F}$ denote the set of facilitators. To respond to a query f, participant i shares $f(i, d_i)$ among the facilitators, and takes no further part in the computation.

To generate the noise, each member of $\mathcal{F}$ essentially takes on the work of $n/|\mathcal{F}|$ participants. When $|\mathcal{F}|$ is small, the batch verification technique of [3] may be employed to verify the secrets shared out by each of the players (that is, one batch verification per member of $\mathcal{F}$), although this technique requires that the faulty players form a smaller fraction of the total than we have been assuming up to this point.

5.2 When f is Not a Predicate

Suppose we are evaluating f to k bits of precision, that is, k bits beyond the binary point. Let q be sufficiently large, say, at least $q > n2^k$. We will work in GF(q). Participant i will share out $2^k f(i, d_i)$, *one bit at a time*. Each of these is checked for membership in $\{0, 1\}_{\mathrm{GF}(q)}$. Then the shares of the most significant bit are multiplied by 2^{k-1}, shares of the next most significant are multiplied by 2^{k-2} and so on, and the shares of the binary representation of $f(i, d_i)$ are then summed. The noise generation procedure is amplified as well. Details omitted for lack of space.

5.3 Beyond Sums

We have avoided the case in which f is an arbitrary function mapping the entire database to a (tuple of) value(s), although the theory for this case has been developed in [13]. This is because without information about the structure of f we can only rely on general techniques for secure function evaluation of f, which may be prohibitively expensive.

One case in which we can do better is in the generation of *privacy-preserving histograms*. A histogram is specified by a partition of the domain Rows; the true response to the histogram query is the exact number of elements in the database residing in each of the cells of the histogram. Histograms are *low sensitivity* queries, in that changing a single row of the database changes the counts of at most two cells in the histogram, and each of these two counts changes by at most 1. Thus, as discussed in [13], ϵ-indistinguishable histograms may be obtained by adding exponential noise with $R = 1/2\epsilon$ to each cell of the histogram. A separate execution of ODO for each cell solves the problem. The executions can be run concurrently. All participants in the histogram query must participate in each of the concurrent executions.

5.4 Individualized Privacy Policies

Suppose Citizen C has decided she is comfortable with a *lifetime* privacy loss of, say $\epsilon = 1$. Privacy erosion is cumulative: any time C participates in the ODO protocol she incurs a privacy loss determined by R, the parameter used in noise generation. C has two options: if R is fixed, she can limit the number of queries in which she participates, *provided the decision whether or not to participate is independent of her data*. If R is not fixed in advance, but is chosen by consensus (in the social sense), she can propose large values of R, or to use large values of R for certain types of queries. Similarly, queries could be submitted with a stated value of R, and dataholders could choose to participate only if this value of R is acceptable to them for this type of query. However, the techniques will all fail if the set of participants is more than one-third faulty; so the assumption must be that this bound will always be satisfied. This implicitly restricts the adversary.

6 Summary of Results

This work ties together two areas of research: the study of privacy-preserving statistical databases and that of cryptographic protocols. It was inspired by the combination of the computational power of the noisy sums primitive in the first area and the simplicity of secure evaluation of sums in the second area. The effect is to remove the assumption of a trusted collector of data, allowing individuals control over the handling of their own information.

In the course of this work we have developed distributed algorithms for generation of Binomial and Poisson noise in shares. The former makes novel use of extractors for bit-fixing sources in order to reduce the number of secret sharings needed in generating massive numbers of coins. The latter examined for the first time distributed coin-flipping of coins with arbitrary bias.

References

1. D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.
2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 439–450, May 2000.
3. M. Bellare, J. A. Garay, and T. Rabin. Distributed pseudo-random bit generators—a new way to speed-up shared coin tossing. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 191–200, 1996.
4. A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: The SuLQ framework. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 128–138, June 2005.
5. S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Toward privacy in public databases. In *Proceedings of the 2nd Theory of Cryptography Conference*, pages 363–385, 2005.
6. S. Chawla, C. Dwork, F. McSherry, and K. Talwar. On the utility of privacy-preserving histograms. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 2005.
7. B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem of t-resilient functions. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 429–442, 1985.
8. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 383–395, 1985.
9. A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 14–19, 1989.

10. I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proceedings of the 3rd Theory of Cryptography Conference*, pages 285–304, 2006.
11. I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 202–210, 2003.
12. C. Dwork, J. Lotspiech, and M. Naor. Digital signets for protection of digital information. In *Proceedings of the 28th annual ACM symposium on Theory of computing*, pages 489–498, 1996.
13. C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Theory of Cryptography Conference*, pages 265–284, 2006.
14. C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *Advances in Cryptology: Proceedings of Crypto*, pages 528–544, 2004.
15. A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 211–222, June 2003.
16. Ariel Gabizon, Ran Raz, and Ronen Shaltiel. Deterministic extractors for bit-fixing sources by obtaining an independent seed. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 394–403, 2004.
17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.
18. Oded Goldreich. *Foundations of Cryptography - Basic Applications*, volume 2. Cambridge University Press, 2004.
19. J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 92–101, 2003.
20. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.
21. N. Nisan and D. Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.
22. Michael O. Rabin. Randomized Byzantine generals. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.
23. Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the EATCS*, 77:67–95, 2002.
24. L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 32–42, 2000.
25. A. Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.