On the Lindell-Pinkas Secure Computation of Logarithms: From Theory to Practice

Raphael S. Ryger*
Yale University
New Haven, CT USA
ryger@cs.yale.edu

Onur Kardes†
Stevens Institute of Technology
Hoboken, NJ USA
onur@cs.stevens.edu

Rebecca N. Wright†
Rutgers University
Piscataway, NJ USA
rebecca.wright@rutgers.edu

Abstract

Lindell and Pinkas demonstrated that it is feasible to preserve privacy in data mining by employing a combination of general-purpose and specialized secure-multiparty-computation (SMC) protocol components. Yet practical obstacles of several sorts have impeded a fully practical realization of this idea. In this paper, we address the correctness and practicality of one of their primary contributions, a secure natural logarithm computation, which is a building block crucial to an SMC approach to privacy-preserving data mining applications including construction of ID3 trees and Bayesian networks. We first demonstrate a minor error in the Lindell-Pinkas solution, then provide a correction along with several optimizations. We explore a modest trade-off of perfect secrecy for a performance advantage, a strategy that adds flexibility in the effective application of hybrid SMC to data mining.

1 Introduction

Privacy-preservation objectives in data mining can often be framed ideally as instances of secure multiparty computation (SMC), wherein multiple parties cooperate in a computation without thereby learning each other's inputs. The characterization of SMC is very encompassing, admitting a great variety of input and output configurations, so that a general recipe for adding the SMC input security to arbitrary well-specified multiparty computations would seem to solve many quite different problems in one fell swoop. Indeed, general approaches to SMC were proposed for a variety of settings already in the 1980s. Yet the framing of privacy preservation for particular data-mining tasks as SMC problems, making them amenable to the general approaches, is usually not useful. For all but the most trivial computations, the general SMC solutions have been too cumbersome to apply and would be impractical to run. They require the computation to be represented as an algebraic circuit, with all loops unrolled to as many iterations as would possibly be needed for the supported inputs, and with all contingent branches of the logic as conventionally expressed—such as iterations that happen not to be needed—executed in every run regardless of the inputs. One may reasonably conclude that SMC is just a theoretical curiosity, not relevant for real-world privacy-preserving data mining, where inputs are not just a few bits but rather entire databases.

Lindell and Pinkas [LP00, LP02] have shown the latter conclusion to be inappropriate. A privacy-preserving data-mining task, they point out, need not be cast as a monolithic SMC problem to which to apply an expensive general SMC solution. Instead, the task may be decomposed into modules requiring SMC, all within a computational superstructure that may itself admissibly be left public. The modules requiring SMC may, in part, be implemented with special-purpose protocols with good performance, leaving general SMC as a fallback (at the module-implementation level) only where special approaches have not been found. The key to such construction is that we are able to ensure secure chaining of the secure protocol components. We prevent information from leaking at the seams between the SMC components by having them produce not public intermediate outputs but rather individual-party shares of the logical outputs, shares that may then be fed as inputs to further SMC components. Lindell and Pinkas illustrate this creative, hybrid methodology by designing a two-party SMC version of the ID3 data-mining algorithm for building a classification tree, a query-sequencing strategy for predicting an unknown attribute—e.g., loan worthiness—of a new entity whose other attributes—e.g., those characterizing credit history, assets, and income—are obtainable by (cost-bearing) query. At each construction step, the

*Supported in part by ONR grant N00014-01-1-0795 and by US-Israel BSF grant 2002065.
†Supported in part by NSF grant 0331584.

P3DM'08, April 26, 2008, Atlanta, Georgia, USA.

ID3 algorithm enters an episode of information-theoretic analysis of the database of known-entity attributes. The privacy concern is introduced, in the Lindell-Pinkas setting, by horizontal partitioning of that database between two parties that must not share their records. The computation is to go through as if the parties have pooled their data, yet without them revealing to each other in their computational cooperation any more regarding their private data than is implied by the ultimate result that is to be made known to them both.

While demonstrating the potential in a modular SMC approach to prospective appliers of the theory, Lindell and Pinkas offer SMC researchers and implementors some design suggestions for particular SMC modules needed in their structuring of the two-party ID3 computation. Strikingly, they need only three such SMC modules, all relatively small and clearly useful for building other protocols, namely, shares-to-shares logarithm and product protocols and a shares-to-public-value minindex protocol. Their intriguing recommendation for the secure logarithm protocol, critical to the accuracy and performance of SMC data mining whenever information-theoretic analysis is involved, is our focus in this paper.

The present authors have been engaged in a privacy-preserving data-mining project [YW06, KRWF05] very much inspired by Lindell and Pinkas. Our setting is similar: a database is arbitrarily partitioned between two parties wishing to keep their portions of the data private to the extent that is consistent with achieving their shared objective of discovering a Bayes-net structure in their combined data. The information-theoretic considerations and the scoring formula they lead to are very similar to those in the ID3 algorithm for classification-strategy building, as is the external flow of control that invokes scoring on candidate next query attributes given a set of query attributes that has already been decided upon. (The details and their differences are not germane to the present discussion.) The adaptation we do for privacy preservation in our two-party setting is, not surprisingly, very similar to what Lindell and Pinkas do. Indeed, we need the same SMC components that they do and just one more, for computing scalar products of binary-valued vectors. The latter additional need has more to do with the difference in setting—we are admitting arbitrary, rather than just horizontal, partitioning of the data—than with the difference in analytical objective. In fact, our software would not require much adjustment to serve as a privacy-preserving two-party ID3 implementation—in fact, supporting arbitrarily partitioned data, given the incorporated scalar-product component.

Launching our investigation a few years after Lindell and Pinkas's paper, we have had the advantage of the availability of the Fairplay system [MNPS04] for actually implementing the Yao-protocol components. We have created tools to support the entire methodology, enabling us to take our protocol from a theoretical suggestion all the way to usable software. This exercise has been illuminating. On one hand, it has produced the most convincing vindication of which we are aware of Lindell and Pinkas' broad thesis regarding the practical achievability of SMC in data mining while teaching us much about the software engineering required for complex SMC protocols. On the other hand, as is typical in implementation work, it has revealed flaws in a number of areas of the underlying theoretical work, including our own. In this paper, we present our observations on the Lindell-Pinkas logarithm proposal. We correct a mathematical oversight and address a general modular-SMC issue that it highlights, the disposition of scaling factors that creep into intermediate results for technical reasons.

We begin in Section 2 with a careful account of the Lindell-Pinkas proposal for a precision-configurable secure two-party shares-to-shares computation of natural logarithms. In Section 3, we explain the mathematical oversight in the original proposal and show that the cost of a straightforward fix by scale-up is surprisingly low, although leaving us with a greatly inflated scale-up factor. In Sections 4 and 5, we propose efficient alternatives for doing arbitrary scaling securely. These enable a significant optimization in the first phase of the Lindell-Pinkas protocol, allowing removal of the table look-up from the Yao circuit evaluation. We briefly point out the effectiveness of a simple dodge of most of the problematics of the Lindell-Pinkas protocol in Section 6. We conclude with a discussion of our implementation of the revised Lindell-Pinkas protocol and its performance in Section 7.

2 The Lindell-Pinkas $\ln x$ protocol

The Lindell-Pinkas proposed protocol for securely computing $\ln x$ is intended as a component in a larger secure two-party protocol. The parties are presumed not to know, and must not hereby learn, either the argument or its logarithm. They contribute secret shares of the argument and obtain secret shares of its logarithm. The proposed design for this protocol module is itself modular, proceeding in two chained phases involving different technology. The first phase internally determines n and ε such that $x = 2^n(1+\varepsilon)$ with $-1/4 \leq \varepsilon < 1/2$. Note that, since n is an approximate base-2 logarithm of x, the first phase gets us most of the way to the desired logarithm of x. Furthermore, this phase dominates the performance time of the en-

tire logarithm protocol: in absence of a specialized SMC protocol for the first phase, Lindell and Pinkas fall back to dictating it be implemented using Yao's general approach to secure two-party computation, entailing gate-by-gate cryptography-laden evaluation of an obfuscated Boolean circuit. Yet the main thrust of the Lindell-Pinkas recommendation is in the second phase, which takes (the secret shares of) ε delivered by phase one and computes an additive correction to the logarithm approximation delivered (as secret shares) by phase one.

We will return to the performance-critical considerations in implementing phase one, not addressed by Lindell and Pinkas. We assume that its Boolean circuitry reconstitutes x from its shares; consults the top 1-bit in its binary representation and the value of the bit following it to determine n and ε as defined; represents n and ε in a manner to be discussed; and returns shares of these representations to the respective parties. These values allow an additive decomposition of the sought natural logarithm of x,

$$\ln x \;=\; \ln\, 2^n(1+\varepsilon) \;=\; n\ln 2 + \ln(1+\varepsilon) \tag{2.1}$$

The purpose is to take advantage of the Taylor expansion of the latter term,

$$\ln(1+\varepsilon) \;=\; \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\varepsilon^i}{i} \;=\; \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3} - \frac{\varepsilon^4}{4} + \cdots \tag{2.2}$$

to enable, in phase two, correction of the phase-one approximation of the logarithm with configurable precision by choice of the number of series terms to be used—a parameter k to be agreed upon by the parties. The computation in the second, refining phase is to proceed by oblivious polynomial evaluation, a specialized SMC technology which is inexpensive compared to the Yao protocol of the first phase.

In this rough mathematical plan, the value ε to be passed from phase one to phase two is a (generally non-integer) rational and the terms in the decomposition of the final result in equation (2.1) are (generally non-integer) reals, whereas the values we will accept and produce in the two SMC phases are most naturally viewed as integers. We are, then, representing the rational and the reals as integers through scale-up and finite-precision approximation. We have considerable latitude in choice of the scale-up factors, particularly considering that the scale-up of a logarithm is just the logarithm to a different base—just as good for information-theoretic purposes as long as the base is used consistently. Still, several considerations inform our choice of scale-up factors. We want the scale-ups to preserve enough precision. On the other hand, there is a performance penalty, here and elsewhere in the larger computation to which this component is contributing, especially in Yao-protocol episodes, for processing additional bits. The chosen scale-up must work mathematically within the larger computation. If an adjustment of the scaling were to be needed for compatibility with the rest of the computation—other than further scale-up by an integer factor—it would entail another secure computation. (We return to this issue in §4.) For the Lindell-Pinkas ID3 computation or for our Bayes-net structure-discovery computation, both information-theoretic, no adjustment would be needed. All the terms added and subtracted to get scores within the larger computation would be scaled similarly, and those scaled scores serve only in comparison with each other.

We assume that the parties have common knowledge of some upper bound N on n, the approximate base-2 logarithm of the input x, and we have phase one deliver the rational ε scaled up by 2^N. This loses no information, deferring control of the precision of the correction term, $\ln\, 2^n(1+\varepsilon)$ in some scale-up, to phase two. Bearing in mind that the slope of the natural-logarithm function is around 1 in the interval around 1 to which we are constraining $1+\varepsilon$, we aim for a scale-up of the correction term by at least 2^N, and plan to scale up the main term of the decomposition, $n\ln 2$, to match. Lindell and Pinkas suggest that the mapping from n to $n\ln 2\cdot 2^N$ be done by table look-up within the Yao protocol of phase one. Any further integer scale-up of the main term to match the scaling of the correction term can be done autonomously by the parties, without SMC, by modular multiplication of their respective shares.

Lindell and Pinkas stipulate that the sharing be with respect to a finite field $\mathcal{F}$ that is large enough in a sense we discuss in more detail in Section 3. A non-field ring will do provided that any particular needed inverses exist. This allows us, e.g., to use Paillier homomorphic encryption in a $\mathbb{Z}_{pq}$ both for the oblivious polynomial evaluation needed in phase two of this logarithm component and, subsequently in the larger computation, for the shares-to-shares secure multiplication to compute $x\ln x$—without additional secure Yao computations to convert the sharing from one modulus to another. The only inverses Lindell and Pinkas need here are of powers of 2, and these would be available in $\mathbb{Z}_{pq}$.

The set-up for phase two, then, preserving the Lindell-Pinkas notation, is that phase one has delivered to the parties, respectively, shares β_1 and β_2 such that $\beta_1 +_{\mathcal{F}} \beta_2 = n\ln 2\cdot 2^N$, toward (whatever ultimate scale-up of) the main term of the decomposition (2.1); and shares α_1 and α_2 such that $\alpha_1 +_{\mathcal{F}} \alpha_2 = \varepsilon\cdot 2^N$, toward the phase-two computation of (the scale-up of) the correction term of the decomposition. We continue to

phase two.

Replacing ε in formula (2.2) with $(\alpha_1 +_{\mathcal{F}} \alpha_2)/2^N$, we get

$$\ln(1+\varepsilon) \;=\; \sum_{i=1}^{\infty} \frac{(-1)^{i-1}(\alpha_1 +_{\mathcal{F}} \alpha_2)^i}{i\ 2^{Ni}} \tag{2.3}$$

In this infinite-series expression, the only operation to be carried out in the finite ring $\mathcal{F}$ is the recombination of the shares, α_1 and α_2, as noted. The objective in phase two is to compute the series in sufficiently good approximation through oblivious polynomial evaluation by the two parties, returning shares of the value to the parties. So we need to get from the infinite series—a specification of a limit in $\mathbb{R}$ for what appear to be operations in $\mathbb{Q}$—to a polynomial over the finite ring $\mathcal{F}$ that may be evaluated so as to contribute to the sought shares. This will entail several steps of transformation.

Step 1. The computation must be finite. We take only k terms of the series.

Step 2. We deal somehow with the division that appears in the summand. We need to be sure we end up, when the transformation is complete, with a polynomial over $\mathcal{F}$. We can scale up the whole formula to cancel some or all of the division. The disposition of any remaining division, as we work toward determining the coefficients of the polynomial to be evaluated, turns out to be problematic, largely occasioning this paper. (The existence of modular inverses in $\mathcal{F}$ for the remaining divisors is not sufficient.) For the moment, let σ be whatever scale-up factor we decide to use here.

Step 3. We *reinterpret* the outer summation and the multiplication, including the binomial exponentiation and the multiplication by σ, as modular addition and multiplications in $\mathcal{F}$. Note that we cannot even open the parentheses by formal exponentiation, applying a distributive law, without first reinterpreting the multiplication as in $\mathcal{F}$. We have no law regarding the distribution of multiplication in $\mathbb{Z}$ over addition in $\mathcal{F}$. This requires that we assure ourselves that the reinterpretation does not alter the value of the expression. Lindell and Pinkas ensure this by requiring $\mathcal{F}$ to be sufficiently large, and we will review the consideration.

Step 4. We replace the occurrence of 'α_2' in (2.3)—as truncated, division-resolved, and modularly reinterpreted—with the variable 'y'. Knowing α_1, party 1 does the formal exponentiations and collects terms, all modulo $|\mathcal{F}|$, yielding a polynomial in 'y' over $\mathcal{F}$. Party 1 randomly chooses $z_1 \in \mathcal{F}$ and subtracts it from the constant term of the polynomial. Where $Q(y)$ is the resulting polynomial and z_2 is its value at $y = \alpha_2$, to be obtained by party 2 through the oblivious polynomial evaluation to follow, we have

$$z_2 \;=\; Q(y)\,|_{y=\alpha_2} \;=\; \sum_{i=1}^{k} \frac{\sigma(-1)^{i-1}(\alpha_1 + y)^i}{i\ 2^{Ni}} - z_1 \Bigg|_{y=\alpha_2} \tag{2.4}$$

where all operations—once the approach to the division in the summand is sorted out—are in $\mathcal{F}$, so that

$$z_1 +_{\mathcal{F}} z_2 \;\approx\; \sum_{i=1}^{\infty} \frac{\sigma(-1)^{i-1}(\alpha_1 +_{\mathcal{F}} \alpha_2)^i}{i\ 2^{Ni}} \;=\; \ln(1+\varepsilon)\cdot\sigma$$

—all operations here, except as indicated, back in $\mathbb{R}$. Thus, the computation of z_2 according to (2.4) by oblivious polynomial evaluation accomplishes the sharing of $\ln(1+\varepsilon)\cdot\sigma$ as z_1 and z_2. The parties may autonomously modularly multiply β_1 and β_2 by $\mathrm{lcm}(2^N, \sigma)/2^N$, giving β_1' and β_2', respectively; and modularly multiply z_1 and z_2 by by $\mathrm{lcm}(2^N, \sigma)/\sigma$, giving z_1' and z_2', respectively; and modularly add their respective results from these scale-ups. Then, per the decomposition in (2.1),

$$\begin{aligned}(\beta_1' +_{\mathcal{F}} z_1') +_{\mathcal{F}} (\beta_2' +_{\mathcal{F}} z_2') \;&=\; (\beta_1' +_{\mathcal{F}} \beta_2') +_{\mathcal{F}} (z_1' +_{\mathcal{F}} z_2') \\ \approx\; (n\ln 2 + \ln(1+\varepsilon))\cdot \mathrm{lcm}(2^N, \sigma) \;&=\; \ln x \cdot \mathrm{lcm}(2^N, \sigma)\end{aligned}$$

accomplishing the original goal of securely computing shares of $\ln x$ from shares of x—if with a scale-up that we hope is innocuous. But this sketch of the protocol still needs to be fleshed out. We back up now, first briefly to step 3, and then to step 2, our main focus.

By the time we get to step 3, we should be left with an expression prescribing finitely many operations in $\mathbb{Z}$, viewing $+_{\mathcal{F}}$ as an operation in $\mathbb{Z}$ and viewing division as a partially-defined operation in $\mathbb{Z}$. Looking ahead to step 4, we will be replacing the occurrences of 'α_2' in this expression with the variable 'y' and algebraically reorganizing it into the polynomial $Q(y)$ (with a change to the constant term). In this step 3, we change only the semantics of the expression arrived at, not its syntactic composition. The claim to be made is that the hybrid expression at hand, involving some modular additions but otherwise non-modular operations, can be reinterpreted to involve only modular operations without change to the induced expression value—allowing the expression then to be transformed syntactically with guarantee of preservation of value, but now with respect to the new semantics. This tricky claim, made implicitly, bears explicit examination. We can frame the issue abstractly. Suppose φ is an arbitrarily complex numerical expression built recursively of variables and function symbols (admitting constants as 0-ary function symbols). We have a conventional interpretation of φ in the domain $\mathbb{Z}$. We also have an alternate interpretation

of φ in the domain $\mathbb{Z}_m$. Furthermore, we have an alternate expression, φ', obtained from φ by transformations guaranteed to preserve the value of the whole under the interpretation in $\mathbb{Z}_m$ for any assignment of values from $\mathbb{Z}_m$ to the variables. We intend to compute φ' as interpreted in $\mathbb{Z}_m$. Under what circumstances can we be assured that this computation will yield the same value as does evaluation of the original expression φ according to the original interpretation in $\mathbb{Z}$? In the case at hand, φ is

$$\sum_{i=1}^{k} \frac{\sigma(-1)^{i-1}(\alpha_1 +_{\mathcal{F}} y)^i}{i\ 2^{Ni}} \tag{2.5}$$

(with some decision as to how to interpret the division), whereas φ' is

$$Q(y) + z_1$$

to be interpreted in $\mathbb{Z}_m$ (where $m = |\mathcal{F}|$) and be so computed, with the value to be assigned to 'y' in both cases being α_2.

There are obvious strong sufficient conditions under which modular reinterpretation preserves value. We do have to be careful to take into account, in generalizing, that in our instance ε may be negative, and that our summation expression has sign alternation, so we need to proceed via a "signed-modular" interpretation, wherein the mod-m integers $\lceil \frac{m}{2} \rceil$ to $m-1$ are viewed as "negative", i.e., they are isomorphically replaced by the integers $-\lfloor \frac{m}{2} \rfloor$ to -1. (Choosing the midpoint for the cutover here is arbitrary, in principle, but appropriate for our instance.) If (a) for the values we are contemplating assigning to the variables, the recursive evaluation of φ under the original interpretation assigns values to the subexpressions of φ that are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; and if (b) the functions assigned to the function symbols in the signed-modular reinterpretation agree with the functions assigned by the original interpretation whenever the arguments and their image under the original function are all in that signed-mod-m interval; then the signed-modular reinterpretation will agree with the original interpretation on the whole expression φ for the contemplated value assignments to the variables. Note that we need not assume that the reinterpretation associates with the function symbols the signed-modular analogues of the original functions, although this would ensure (b). Nor would a stipulation of modular agreement be sufficient, in general, without condition (a), even if the original evaluation produces only (overall) values in the signed-mod-m domain for value assignments of interest. The danger is that modular reduction of intermediate values, if needed, may lose information present in the original evaluation. In our case, the single variable, 'y', is assigned the value α_2, which may be as large as, but no larger than, $m-1$. The constant α_1 is similarly less than m. We can view these mod-m values, returned by the Yao protocol in phase one, as being the corresponding signed-mod-m values instead, with $+_{\mathcal{F}}$ operating on them isomorphically. Moreover, $\alpha_1 +_{\mathcal{F}} y$ then evaluates into the interval $[-\frac{1}{4}2^N, \frac{1}{2}2^N)$, where we can arrange for the endpoints to be *much* smaller in absolute value than $\lfloor \frac{m}{2} \rfloor$. This allows Lindell and Pinkas to reason about setting m high enough so that indeed all subexpressions of our φ will evaluate, in the original interpretation, into the signed-mod-m domain. Note that if formal powers of 'α_1' and of 'y' appeared as subexpressions in our original expression φ, as they do in our φ', the polynomial $Q(y)+z_1$ which we actually compute, we would have concern over potential loss of information in modular reduction impeding the modular reinterpretation; but the power subexpressions appear only *after* we have reinterpreted and transformed φ, and are by then of no concern.

We now return to step 2, attending to the division in the Taylor-series terms.

3 The division problem

We have already seen that choices of scaling factor are governed by several considerations including preservation of precision, avoidance of division where it cannot be carried out exactly, and compatibility among intermediate results. For preservation of precision, we have been aiming to compute the main and correction terms of (2.1) scaled up by at least 2^N. Lindell and Pinkas incorporate this factor into their σ in preparing the polynomial. To dispose of the i factors in the denominator in (2.4), they increase the scale-up by a factor of $\mathrm{lcm}(2,\ldots,k)$. With σ now at $2^N\mathrm{lcm}(2,\ldots,k)$, the truncated Taylor series we are looking at in step 2 becomes

$$\ln(1+\varepsilon)\cdot 2^N\mathrm{lcm}(2,\ldots,k)\ \approx \sum_{i=1}^{k} \frac{(-1)^{i-1}\ (\mathrm{lcm}(2,\ldots,k)/i)\ (\alpha_1 +_{\mathcal{F}} \alpha_2)^i}{2^{N(i-1)}} \tag{3.6}$$

We know that in step 3 we will be reinterpreting the operations in this expression—more precisely, in the expression we intend this expression to suggest—as operations in $\mathcal{F}$. Clearly, since k is agreed upon before the computation, the subexpression '$\mathrm{lcm}(2,\ldots,k)/i$' may be replaced immediately by (a token for) its integer value. We are still left with a divisor of $2^{N(i-1)}$, but Lindell and Pinkas reason that $(\alpha_1 +_{\mathcal{F}} \alpha_2)^i$, although not determined until run time, will be divisible by $2^{N(i-1)}$. After all, $(\alpha_1 +_{\mathcal{F}} \alpha_2)^i$ will be $(\varepsilon \cdot 2^N)^i$, and the denominator was designed expressly to divide this

to leave $\varepsilon^i \cdot 2^N$. Apparently, all we need to do is allow the division bar to be reinterpreted in step 3 as the (partially defined) division operation in $\mathbb{Z}_m$, i.e., multiplication by the modular inverse of the divisor. We can assume that m is not even, so that powers of 2 have inverses modulo m. Furthermore, whenever a divides b (in $\mathbb{Z}$) and $b < m$, if a has an inverse ($a \in \mathbb{Z}_m^*$) then $a^{-1}b$ in $\mathbb{Z}_m$ is just the integer b/a. It would appear that the strong sufficient conditions for reinterpretation are met.

The trouble is that, although $(\alpha_1 +_{\mathcal{F}} \alpha_2)^i = (\varepsilon \cdot 2^N)^i$ is an integer smaller than m (given that we will ensure that m is large enough) and although the *expression* '$(\varepsilon \cdot 2^N)^i$' appears to be formally divisible by the expression '$2^{N(i-1)}$', the integer $(\varepsilon \cdot 2^N)^i$ is not, in general, divisible by the integer $2^{N(i-1)}$. In $\mathbb{Q}$, the division indeed yields $\varepsilon^i 2^N$, which is just the scale-up by 2^N we engineered it to achieve. That rational scale-up is an integer for $i = 1$, but will generally not be an integer for $i > 1$. (Roughly, $\varepsilon^i 2^N$ is an integer if the lowest-order 1 bit in the binary representation of x is within N/i digits of its highest-order 1 bit—a condition that excludes most values of x already for $i = 2$.) This undermines the sufficient condition Lindell and Pinkas hoped to rely on to justify the modular reinterpretation, our step 3. Without the divisibility in the integers, there is no reason to believe that reinterpretation of the division by $2^{N(i-1)}$ as modular multiplication by its mod-m inverse $(2^{N(i-1)})^{-1}$ would have anything to do with the approximation we thought we were computing. The ensuing formal manipulation in step 4 to get to a polynomial to be evaluated obliviously would be irrelevant.

The immediate brute-force recourse is to increase the scale-up factor, σ, currently at $2^N \mathrm{lcm}(2, \ldots, k)$, to $2^{Nk}\mathrm{lcm}(2, \ldots, k)$. This leaves our truncated Taylor series as

$$\ln(1+\varepsilon) \cdot 2^{Nk}\mathrm{lcm}(2, \ldots, k) \approx$$
$$\sum_{i=1}^{k} (-1)^{i-1}\, 2^{N(k-i)} (\mathrm{lcm}(2, \ldots, k)/i)\, (\alpha_1 +_{\mathcal{F}} \alpha_2)^i \quad (3.7)$$

Phase one still feeds phase two shares of ε scaled up by 2^N. For compatibility with the larger scale-up of the correction term of the decomposition as now delivered (in shares) by phase two, the parties will autonomously scale up their shares of the main term of the decomposition by a further factor of $2^{N(k-1)}$.

The natural concern that a scaling factor so much larger will require $\mathcal{F}$ to be much larger, with adverse performance implications, turns out to be unfounded. Surprisingly, the guideline given by Lindell and Pinkas for the size of $\mathcal{F}$—namely, 2^{Nk+2k} or more—need not be increased by much. The original guideline actually remains sufficient for the step-3 reinterpretation of the operations to be sound. But now, with the (unshared) scaled-up correction term alone so much wider, requiring some 2^{Nk} bits of representation, we are in danger of running out of room in the space for the scaled-up main term if $\log_2 N > 2k$. Raising the size requirement for $\mathcal{F}$ to $2^{Nk+2k+\log_2 N}$ should be sufficient. If we want to provide, in the larger protocol, for computation of $x \ln x$, scaled up to $x(\sigma \ln x)$, in the same space $\mathcal{F}$, we need to raise the size requirement for $\mathcal{F}$ to $2^{Nk+2k+\log_2 N+N}$

Our larger scale-up here does not carry any additional information, of course. The creeping growth in the computational space does affect performance, but only minimally. Even in Yao SMC episodes, the larger space affects only the modular addition to reconstitute shared inputs at the outset and the modular addition to share the computed results at the end. The computation proper is affected by the size of the space of the actual unshared inputs, but not by the size of the space for modular sharing.

The more significant issue is that we continue to be saddled with scaling factors that are best not incurred in building blocks intended for general use. We explore efficient ways to reverse unwanted scaling. The problem is tantamount to that of efficiently *introducing* wanted arbitrary—i.e., not necessarily integral—scaling. Lindell and Pinkas need such scaling to get from base-2 logarithms to natural logarithms in phase one of the protocol. A good solution to this problem of secure arbitrary scaling will enable us to do better than (even a smart implementation of) the table look-up inside the phase-one Yao protocol that they call for, in addition to allowing reversal of whatever scale-up is delivered by the entire logarithm protocol.

4 Secure non-integer scaling of shared values

Suppose parties 1 and 2 hold secret shares modulo m, respectively γ_1 and γ_2, of a value γ; and suppose $\sigma = \kappa + \rho$ is a scaling factor to be applied to γ, where κ is a non-negative integer and $0 \leq \rho < 1$. $\sigma\gamma$ is not, in general, an integer, but a solution that can provide the parties shares of an integer approximation of $\sigma\gamma$ suffices. $\kappa\gamma$ may be shared exactly simply by having the parties autonomously modularly scale up their shares by κ. That leaves the sharing of (an approximation of) $\rho\gamma$, the shares to be added modularly to the shares of $\kappa\gamma$ to obtain shares of (an approximation of) $\sigma\gamma$. The problem is that approximate multiplication by a non-integer does not distribute over modular addition, even approximately!

A bifurcated distributive property does hold, however. If the ordinary sum $\gamma_1 + \gamma_2$ is $< m$, the usual distributive law for multiplication of the sum by ρ holds

approximately for approximate multiplication. If, on the other hand, the ordinary sum $\gamma_1 + \gamma_2$ is $\geq m$, then the modular sum is, in ordinary terms, $\gamma_1 + \gamma_2 - m$, so that the distribution of the multiplication by ρ over the modular addition of γ_1 and γ_2 will need an adjustment of approximately $-\rho m$. This suggests the following protocol to accomplish the scaling by ρ mostly by autonomous computation by the parties on their own shares, but with a very minimal recourse to a Yao protocol to select between the two cases just enumerated. The Yao computation takes $\rho\gamma_1$ and $\rho\gamma_2$, each rounded to the nearest integer, as computed by the respective parties; and the original shares γ_1 and γ_2 as well. Party 1 also supplies a secret random input $z_1 < m$. The circuit returns to party 2 either $(\rho\gamma_1 + \rho\gamma_2) +_{\mathrm{mod}\, m} z$ or $(\rho\gamma_1 + \rho\gamma_2 - \rho m) +_{\mathrm{mod}\, m} z$ accordingly as $\gamma_1 + \gamma_2 < m$ or not. Party 1's share is $m - z_1$. The integer approximation of ρm is built into the circuit. The cumulative approximation error is less than 1.5, and usually less than 1.

But an unconventional approach can allow us to do better still.

5 The practical power of imperfect secrecy

In implementing secure protocols, we tend to be induced by different considerations to choose moduli for sharing that are vastly larger than the largest value that will be shared. In the Lindell-Pinkas logarithm proposal, for instance, if N is 13, as to accommodate ID3 database record counts of around 8,000, and k is 4, our share space is of a size greater than 10^{20}. Prior to our correction, logarithms are to be returned scaled up by around 10^5, making for a maximum output of around 10^6. Thus, the size of the sharing space is larger than the largest shared value by a factor of 10^{14}. In such a configuration, it is a bit misleading to state that the distributive law is bifurcated. The case of the shares *not* jointly exceeding the modulus is very improbable. If we could *assume* the nearly certain case of the shares being excessive—i.e., needing modular reduction—to hold, we would not need a Yao episode to select between two versions of the scaling computation. Each party would scale autonomously and party 1 would subtract ρm to correct for the excess.

We could abide the very small chance of error in this assumption. But better would be to guarantee (approximate) correctness of the autonomous scaling by contriving to *ensure* that the shares be excessive. This turns out to be quite tricky in theory while straightforward in practice. It entails a small sacrifice of the information-theoretic perfection of the secrecy in the sharing, but the sacrifice should be of no practical significance.

Let t be the largest value to be shared, much smaller than the modulus m. We can ensure that shares are excessive by restricting the independently set share to be greater than t. But we can show that if it is agreed that the independent share will be chosen uniformly randomly from the interval $[t+1, m-1]$ then, if it is actually chosen within t of either end of this interval, information will leak to the other party through the complementary share given him for certain of the values from $[0, t]$ that might be shared—to the point of completely revealing the value to the other party in the extreme case. If the choice is at least t away from the ends of the choice interval, perfect secrecy is maintained. But if we take this to heart and agree that the independent share must be from the smaller interval $[2t + 1, m - 1 - t]$ then the same argument can be made regarding the possibility that the choice is actually within t of the ends of this smaller interval. Recursively, to preserve secrecy, we would lop off the ends of the choice interval until nothing was left.

But as in the "surprise quiz" (or "unexpected hanging") paradox, wherein we establish that it is impossible to give a surprise quiz "some day next week," the conclusion here, too, is absurd from a practical point of view. If the independent share is chosen from some huge, but *undeclared*, interval around $m/2$, huge by comparison with t but tiny by comparison with m, there simply is no problem with loss of secrecy. We can assume that the sharing is excessive, and arbitrary scaling can be accomplished by the parties completely autonomously.

We may be able to look at the random choice of the independent share from an undeclared interval instead as a non-uniform random choice, the distribution being almost flat, with the peak probability around $m/2$ dropping off extremely gradually to 0 as the ends of $[t + 1, m - 1]$ are approached. As long as the probabilities are essentially the same in a cell of radius t around whatever independent share is actually chosen—and it is exceedingly unlikely that there not exist a complete such cell around the choice—secrecy is preserved. But theorizing about the epistemology here is beyond our scope. The point is that, in practice, it seems worth considering that we can gain performance by not requiring Yao episodes when non-integer scaling is needed.

In the Lindell-Pinkas protocol, for scaling the approximate base-2 logarithms determined in phase one to corresponding approximate natural logarithms, this approach is fine. For getting rid of the scale-up delivered in the final result, beyond whatever scale-up is sufficient for the precision we wish to preserve, we would need to extend the size of $\mathcal{F}$ somewhat before using this

approach, now that our correction has greatly increased the maximum value that may be delivered as shares by the oblivious polynomial evaluation. On balance, considering the added expense that would be incurred in other components of the larger protocol, it is best not to enlarge $\mathcal{F}$ (further) and to reverse the scaling of the result, if necessary, by the method of the preceding section.

6 Alternative: Pretty good precision, high performance

For many purposes, a much simpler secure computation for logarithms may offer adequate precision. The base is often not important, as noted, so base 2 may do—as indeed it would in the ID3 computation. Noting that in the interval $[1, 2]$ the functions $y = \log_2 x$ and $y = x - 1$ agree at the ends of the interval and deviate by only 0.085 in the middle, we have the Yao circuit determine the floor of the base-2 logarithm and then append to its binary representation the four bits of the argument following its top 1-bit. This gives a result within 1/16 of the desired base-2 logarithm. We used this approach in our Bayes-net structure computation [YW06, KRWF05] while sorting out the issues with the much more complex Lindell-Pinkas proposal. As in the Lindell-Pinkas secure ID3 computation, the logarithms inform scores that, in turn, are significant only in how they compare with other scores, not in their absolute values. As long as the sense of these score comparisons is not affected, inaccuracies in the logarithms are tolerable. We bear in mind also that, in the particular data-mining contexts we are addressing, the algorithms are based on taking the database as a predictive sample of a larger space. In so depending on the database, they are subject to what may be regarded as sampling error in any case. From that perspective, even the reversal of sense in some comparisons of close scores cannot be regarded as rendering the approach inappropriate.

However, as much simpler as this approach is, the performance consideration in its favor is considerably weakened once we remove the conversion from base-2 to scaled-up natural logarithms from the Yao portion of the Lindell-Pinkas protocol, as we now see we can do.

7 Implementation and performance

We have evolved an array of tools to aid in developing hybrid-SMC protocols of the style demonstrated by Lindell and Pinkas. These will be documented in a Yale Computer Science Department technical report and will be made available. Among the resources are a library of Perl functions offering a level of abstraction and control we have found useful for specifying the generation of Boolean circuits; scripts for testing circuits without the overhead of secure computation; particular circuit generators, as for the phase-one Yao episode in the Lindell-Pinkas logarithm protocol and for the minindex Yao episode needed for the best-score selection in their larger secure ID3 computation; additional SMC components not involving circuits; and a library of Perl functions facilitating the coordination of an entire hybrid-SMC computation involving two parties across a network.

We have been developing and experimenting on NetBSD and Linux operating systems running on Intel Pentium 4 CPUs at 1.5 to 3.2 GHz. We use the Fairplay run-time system, written in Java and running over Sun JRE 1.5, to execute Yao-protocol episodes. The Yao episode in phase one of the Lindell-Pinkas logarithm protocol completely dominates the running time of the entire logarithm computation, making the peformance of Fairplay itself critical.

We cannot address the performance of multiparty computations without giving special attention to the cost of communication. This element is a wildcard, dependent on link quality and sheer propagation delay across the network distance between the parties. We have done most of our experimentation with the communication component trivialized by running both parties on the same machine or on two machines on the same LAN. For a reality check, we did some experimenting with one party at Yale University in New Haven, CT and the other party at Stevens Institute of Technology in Hoboken, NJ, with a 15 ms round-trip messaging time between them. There was no significant difference in performance in Yao computations. Admittedly, this is at a relatively small network distance. But there is another way to look at this. If network distance were really making the communication cost prohibitive, the two parties anxious to accomplish the joint data-mining computation securely could arrange to run the protocol from outposts of theirs housing prepositioned copies of their respective private data, the outposts securely segregated from each other but at a small network distance. From this perspective, and recognizing that the protocols we are considering involve CPU-intensive cryptographic operations, it is meaningful to assess their performance with the communication component minimized.

With the parties running on 3.2 GHz CPUs, and working with a 60-bit modulus, it takes around 5 seconds to run the complete Lindell-Pinkas logarithm computation. In more detail, to accommodate input x of up to 17 bits (≤ 131071), with $k = 3$ terms of the series to be computed in phase 2 (for an absolute error within 0.0112), we generate a circuit of 1497 gates and the computation runs in around 5.0 seconds. With

the same modulus, to accommodate input x of only up to 13 bits ($\leq$ 8191), allowing $k = 4$ terms of the series to be computed in phase 2 (for an absolute error within 0.0044), we generate a circuit of 1386 gates and the computation runs in around 4.9 seconds. Accommodating inputs of only up to 10 bits ($\leq$ 1023), allowing as many $k = 5$ series terms (for an absolute error within 0.0018), the gate count comes down to 1314 and the running time comes down to around 4.8 seconds.

Clearly, a 5-second wait for a single result of a Lindell-Pinkas secure-logarithm computation seems quite tolerable, but it serves little purpose in itself, of course. This is a shares-to-shares protocol intended for incorporation in a larger data-mining protocol that will ultimately leave the parties with meaningful results. It is reasonable to ask, in such a larger hybrid-SMC protocol, how badly would a 5-second delay for each logarithm computation—and, presumably, comparable delays for other needed SMC building blocks—bog down the entire data-mining algorithm?

We can give a rough idea, based on experiment, of the performance that appears to be possible now in an entire privacy-preserving data-mining computation based on a hybrid-SMC approach. Without fully qualifying the tasks, software versions, and hardware involved, our secure Bayes-net structure-discovery implementation has run against an arbitrarily privately partitioned database of 100,000 records of six fields in about 2.5 hours. This involved almost 500 invocations of the secure logarithm protocol, each involving a Yao-protocol episode run using the Fairplay system, as well as other component protocols. The overall time, computing against this many records, was dominated not by the Yao protocol episodes of the logarithm and minindex components but rather by the scalar-product computations needed to determine securely the numbers of records matching patterns across the private portions of the logical database. The scalar-product computations require a number of homomorphic-encryption operations linear in the number of records in the database.

In developing and using these tools over some time, we note that the room for improvement in performance as implementations are optimized is large. Improvements that do not affect complexity classes, hence of lesser interest to theoreticians, are very significant to practitioners. Improvements in complexity class are there as well; we gained a log factor in our gate counts in the logarithm circuits over our initial naive implementation. Meanwhile, it is clear that significant hybrid-SMC computations are already implementable in a maintainable, modular manner with a development effort that is not exorbitant. Performance of such computations is becoming quite reasonable for realistic application in privacy-preserving data-mining contexts.

Acknowledgments

We thank Benny Pinkas for helpful discussion of the design of the original Lindell-Pinkas logarithm protocol.

References

[KRWF05] Onur Kardes, Raphael S. Ryger, Rebecca N. Wright, and Joan Feigenbaum. Implementing privacy-preserving Bayesian-net discovery for vertically partitioned data. In *Proceedings of the ICDM Workshop on Privacy and Security Aspects of Data Mining*, pages 26–34, 2005.

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–54. Springer-Verlag, 2000.

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[MNPS04] Dahlia Malkhi, Noam Nissan, Benny Pinkas, and Yaron Sella. Fairplay – a secure two-party computation system. In *Proc. of the 13th Symposium on Security*, pages 287–302. Usenix, 2004.

[YW06] Zhiqiang Yang and Rebecca N. Wright. Privacy-preserving computation of Bayesian networks on vertically partitioned data. *IEEE Transactions on Data Knowledge Engineering*, 18(9), 2006. An earlier version appeared in KDD 2004.