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

April 26, 2008

Overview

Introduction

The Lindell-Pinkas $\ln x$ protocol

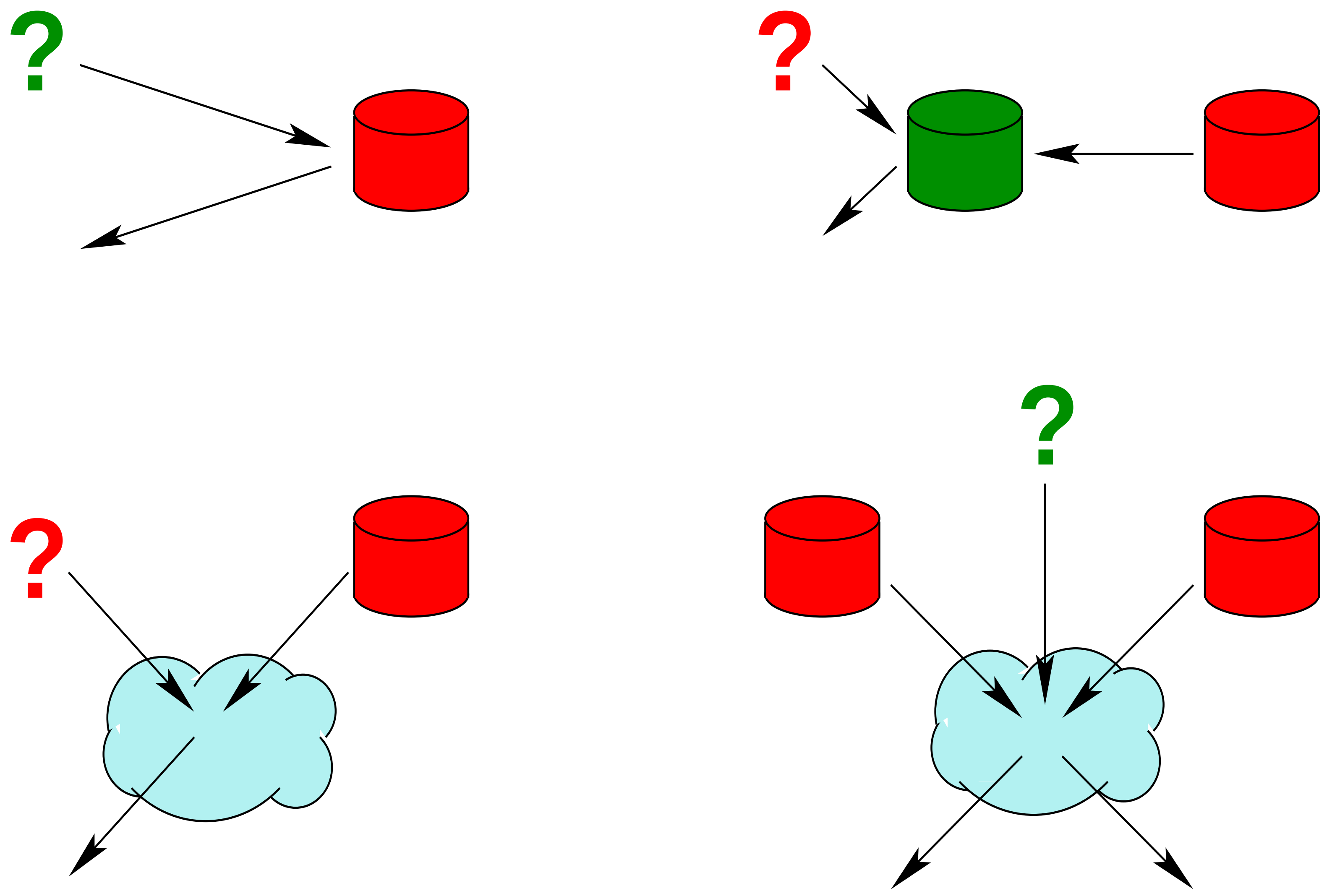
The division problem

Secure non-integer scaling of shared values

Implementation and performance

Conclusion

A variety of PPDM settings



SMC and PPDM

- PPDM dilemmas:
  - **what data to expose** for analysis;
  - **what analyses to allow**.
- Secure multiparty computation – SMC – theoretically eliminates the former, reducing PPDM to the latter.
- **Generic approaches** to achieving SMC are computationally expensive for non-trivial algorithms and large amounts of input data, making them **impractical for PPDM**.
- Lindell, Pinkas, 2000: A **modular, hybrid** SMC approach, combining building blocks implemented through generic or specialized technologies, can be **practical for PPDM**!
- Lindell, Pinkas, 2000: **Logarithm** computation, an important building block, is itself amenable to this approach.

- PPDM dilemmas:
  - **what data to expose** for analysis;
  - **what analyses to allow**.
- Secure multiparty computation – SMC – theoretically eliminates the former, reducing PPDM to the latter.
- **Generic approaches** to achieving SMC are computationally expensive for non-trivial algorithms and large amounts of input data, making them **impractical for PPDM**.
- Lindell, Pinkas, 2000: A **modular, hybrid** SMC approach, combining building blocks implemented through generic or specialized technologies, can be **practical for PPDM**!
- Lindell, Pinkas, 2000: **Logarithm** computation, an important building block, is itself amenable to this approach.

SMC and PPDM

- PPDM dilemmas:
  - **what data to expose** for analysis;
  - **what analyses to allow**.
- Secure multiparty computation – SMC – theoretically eliminates the former, reducing PPDM to the latter.
- **Generic approaches** to achieving SMC are computationally expensive for non-trivial algorithms and large amounts of input data, making them **impractical for PPDM**.
- Lindell, Pinkas, 2000: A **modular, hybrid** SMC approach, combining building blocks implemented through generic or specialized technologies, can be **practical for PPDM**!
- Lindell, Pinkas, 2000: **Logarithm** computation, an important building block, is itself amenable to this approach.

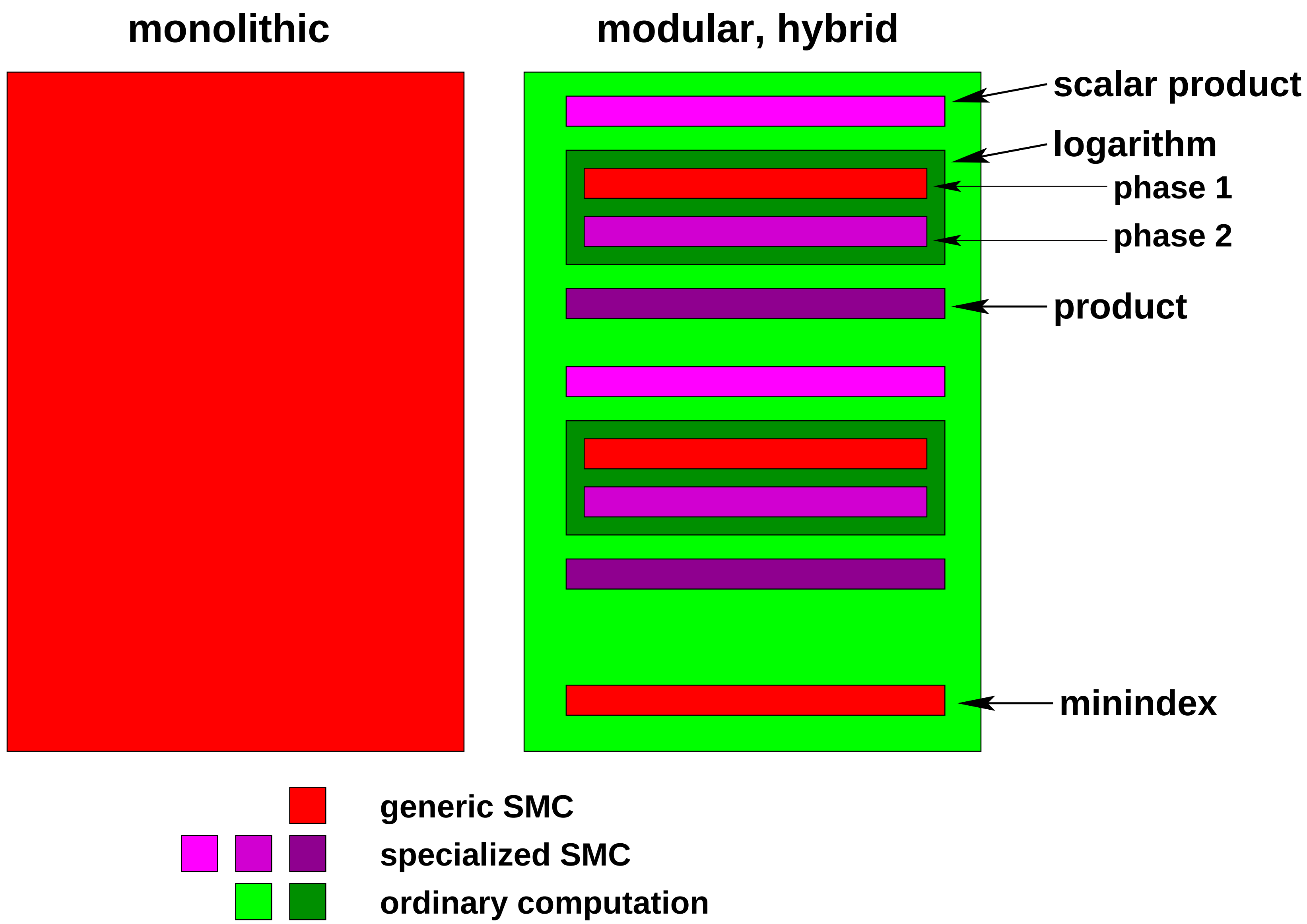
SMC and PPDM

- PPDM dilemmas:
  - **what data to expose** for analysis;
  - **what analyses to allow**.
- Secure multiparty computation – SMC – theoretically eliminates the former, reducing PPDM to the latter.
- **Generic approaches** to achieving SMC are computationally expensive for non-trivial algorithms and large amounts of input data, making them **impractical for PPDM**.
- Lindell, Pinkas, 2000: A **modular, hybrid** SMC approach, combining building blocks implemented through generic or specialized technologies, can be **practical for PPDM**!
- Lindell, Pinkas, 2000: **Logarithm** computation, an important building block, is itself amenable to this approach.

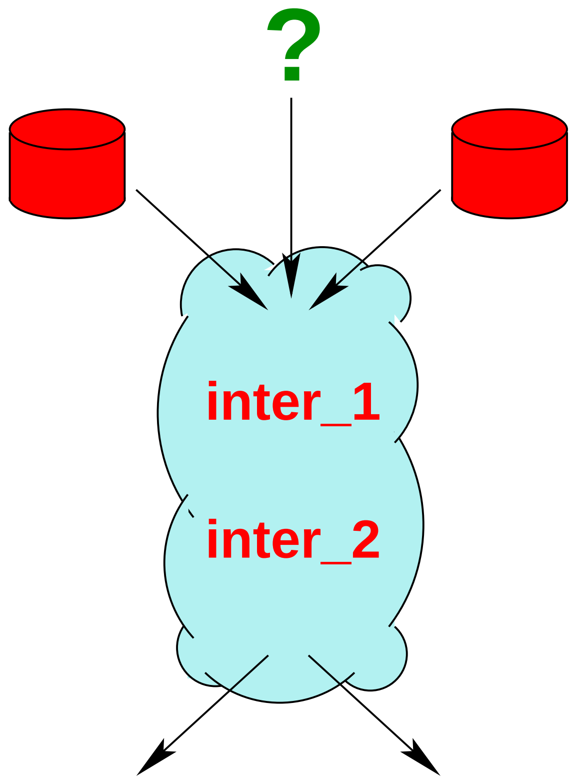
SMC and PPDM

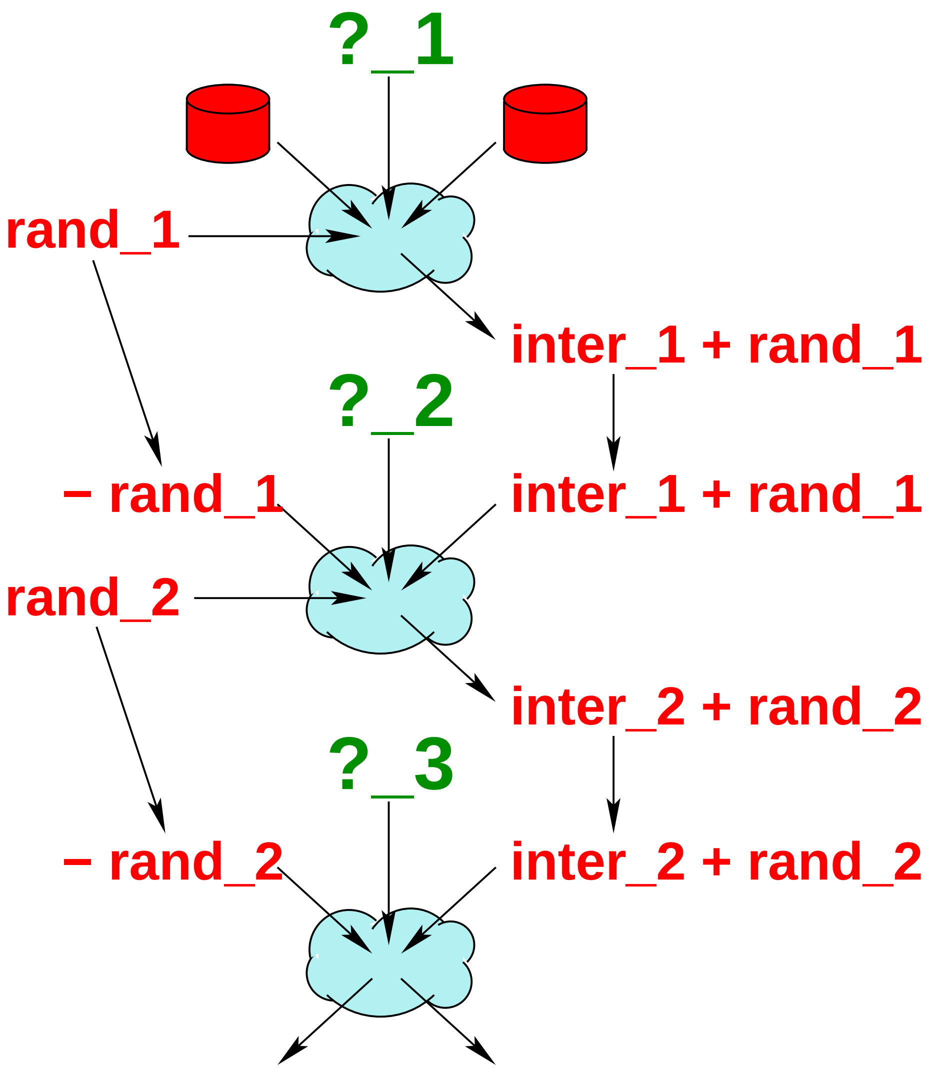
- PPDM dilemmas:
  - **what data to expose** for analysis;
  - **what analyses to allow**.
- Secure multiparty computation – SMC – theoretically eliminates the former, reducing PPDM to the latter.
- **Generic approaches** to achieving SMC are computationally expensive for non-trivial algorithms and large amounts of input data, making them **impractical for PPDM**.
- Lindell, Pinkas, 2000: A **modular, hybrid** SMC approach, combining building blocks implemented through generic or specialized technologies, can be **practical for PPDM**!
- Lindell, Pinkas, 2000: **Logarithm** computation, an important building block, is itself amenable to this approach.

Monolithic vs. modular SMC



Shares to shares: the key to modularity with security





Toward the Lindell-Pinkas theses in practice

- Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- (Increasing available computing power.)
- Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- A circuit-generation library suitable for use with Fairplay.
- A development methodology and a coordination framework for modular multiparty protocols.
- Implementations of building-block modules ...

Toward the Lindell-Pinkas theses in practice

- Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- (Increasing available computing power.)
- Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- A circuit-generation library suitable for use with Fairplay.
- A development methodology and a coordination framework for modular multiparty protocols.
- Implementations of building-block modules ...

Toward the Lindell-Pinkas theses in practice

- Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- (Increasing available computing power.)
- Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- A circuit-generation library suitable for use with Fairplay.
- A development methodology and a coordination framework for modular multiparty protocols.
- Implementations of building-block modules ...

- ☐ Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- ☐ (Increasing available computing power.)
- ☐ Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- ☐ A circuit-generation library suitable for use with Fairplay.
- ☐ A development methodology and a coordination framework for modular multiparty protocols.
- ☐ Implementations of building-block modules ...

Toward the Lindell-Pinkas theses in practice

- Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- (Increasing available computing power.)
- Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- A circuit-generation library suitable for use with Fairplay.
- A development methodology and a coordination framework for modular multiparty protocols.
- Implementations of building-block modules ...

Toward the Lindell-Pinkas theses in practice

- Yang, Wright, Kardes, Ryger, Feigenbaum, 2004, 2005, 2006: Design and implementation of secure two-party **Bayes-net structure discovery** in arbitrarily partitioned data. Using ...
- (Increasing available computing power.)
- Malkhi, Nissan, Pinkas, Sella, 2004: the **Fairplay** system implementing the Yao 1986 generic scheme for secure two-pary computation.
- A circuit-generation library suitable for use with Fairplay.
- A development methodology and a coordination framework for modular multiparty protocols.
- Implementations of building-block modules ...

Building-block SMC modules

Using homomorphic encryption:

- ☐ Private bit vectors to private shares of their **scalar product**.
- ☐ Private shares of arguments to private shares of their **product**.

Building-block SMC modules

Using homomorphic encryption:

- Private bit vectors to private shares of their **scalar product**.
- Private shares of arguments to private shares of their **product**.

Using the Yao generic two-party SMC scheme:

- Sequences of private shares of a sequence of values to their (public) **minindex**, the (smallest) index of the minimum.

Building-block SMC modules

Using homomorphic encryption:

- ☐ Private bit vectors to private shares of their **scalar product**.
- ☐ Private shares of arguments to private shares of their **product**.

Using the Yao generic two-party SMC scheme:

- ☐ Sequences of private shares of a sequence of values to their (public) **minindex**, the (smallest) index of the minimum.

... And using both the Yao generic scheme and homomorphic encryption:

- ☐ Private shares of an argument to private shares of its **logarithm**, following the Lindell-Pinkas proposal—corrected, optimized, and implemented in the work presented here.

The Lindell-Pinkas $\ln x$ protocol: overall plan

- Multiplicatively decompose x as $2^n(1+\varepsilon)$, where $-1/4 \leq \varepsilon < 1/2$. Additively decompose the logarithm,

$$\ln x \;=\; \ln 2^n(1+\varepsilon) \;=\; n\ln 2 + \ln(1+\varepsilon) \qquad (1)$$

  The Taylor expansion of the latter term,

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

  will allow **configurable accuracy**.
- Protocol phase 1: From shares of x, compute shares of n and ε using **generic Yao** two-party secure computation.
- Protocol phase 2: From the shares of ε yielded by phase 1, compute shares of $\ln(1+\varepsilon)$—to "enough" terms of its expansion—using **oblivious polynomial evaluation**.

The Lindell-Pinkas $\ln x$ protocol: overall plan

- Multiplicatively decompose x as $2^n(1+\varepsilon)$, where $-1/4 \le \varepsilon < 1/2$. Additively decompose the logarithm,

$$\ln x = \ln 2^n(1+\varepsilon) = n \ln 2 + \ln(1+\varepsilon) \quad (1)$$

The Taylor expansion of the latter term,

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

will allow **configurable accuracy**.

- Protocol phase 1: From shares of x, compute shares of n and ε using **generic Yao** two-party secure computation.
- Protocol phase 2: From the shares of ε yielded by phase 1, compute shares of $\ln(1+\varepsilon)$—to "enough" terms of its expansion—using **oblivious polynomial evaluation**.

The Lindell-Pinkas $\ln x$ protocol: overall plan

- Multiplicatively decompose x as $2^n(1+\varepsilon)$, where $-1/4 \leq \varepsilon < 1/2$. Additively decompose the logarithm,

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

The Taylor expansion of the latter term,

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

will allow **configurable accuracy**.

- Protocol phase 1: From shares of x, compute shares of n and ε using **generic Yao** two-party secure computation.
- Protocol phase 2: From the shares of ε yielded by phase 1, compute shares of $\ln(1+\varepsilon)$—to “enough” terms of its expansion—using **oblivious polynomial evaluation**.

How many bits of precision?

- **Must** be decided in advance!
- Let N be the lowest agreed upper bound on n. ε may have as many as N bits of precision, which we want to preserve.
- We want similar precision in the output.
- Therefore, since we will be computing in integers, the polynomial we compute in phase 2 must be adjusted to accept ε scaled up by 2^N; and to deliver $\ln(1+\varepsilon)$ scaled up by some factor σ that should be at least 2^N.
- ... But **scaling** of **inputs/outputs** of SMC modules if they are to be accepted/delivered **as private shares** is not as trivial as we are accustomed to thinking.

How many bits of precision?

- **Must** be decided in advance!
- Let N be the lowest agreed upper bound on n. ε may have as many as N bits of precision, which we want to preserve.
- We want similar precision in the output.
- Therefore, since we will be computing in integers, the polynomial we compute in phase 2 must be adjusted to accept ε scaled up by 2^N; and to deliver $\ln(1+\varepsilon)$ scaled up by some factor σ that should be at least 2^N.
- ... But **scaling** of **inputs/outputs** of SMC modules if they are to be accepted/delivered **as private shares** is not as trivial as we are accustomed to thinking.

How many bits of precision?

- **Must** be decided in advance!
- Let N be the lowest agreed upper bound on n. ε may have as many as N bits of precision, which we want to preserve.
- We want similar precision in the output.
- Therefore, since we will be computing in integers, the polynomial we compute in phase 2 must be adjusted to accept ε scaled up by 2^N; and to deliver $\ln(1+\varepsilon)$ scaled up by some factor σ that should be at least 2^N.
- ... But **scaling** of **inputs/outputs** of SMC modules if they are to be accepted/delivered **as private shares** is not as trivial as we are accustomed to thinking.

How many bits of precision?

- **Must** be decided in advance!
- Let N be the lowest agreed upper bound on n. ε may have as many as N bits of precision, which we want to preserve.
- We want similar precision in the output.
- Therefore, since we will be computing in integers, the polynomial we compute in phase 2 must be adjusted to accept ε scaled up by 2^N; and to deliver $\ln(1+\varepsilon)$ scaled up by some factor σ that should be at least 2^N.
- ... But **scaling** of **inputs/outputs** of SMC modules if they are to be accepted/delivered **as private shares** is not as trivial as we are accustomed to thinking.

How many bits of precision?

- **Must** be decided in advance!
- Let N be the lowest agreed upper bound on n. ε may have as many as N bits of precision, which we want to preserve.
- We want similar precision in the output.
- Therefore, since we will be computing in integers, the polynomial we compute in phase 2 must be adjusted to accept ε scaled up by 2^N; and to deliver $\ln(1+\varepsilon)$ scaled up by some factor σ that should be at least 2^N.
- ... But **scaling** of **inputs/outputs** of SMC modules if they are to be accepted/delivered **as private shares** is not as trivial as we are accustomed to thinking.

Accommodating the scaling in phase 2

- Where α_1 and α_2 are the parties' respective additive shares, in some finite field (or ring) $\mathcal{F}$, of $\varepsilon \cdot 2^N$ to be delivered by phase 1,

$$\varepsilon = (\alpha_1 +_{\mathcal{F}} \alpha_2)/2^N$$

- Scaling the phase 2 output up by factor σ, the Taylor series of (2) becomes

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

- ... But we will need a finite polynomial over $\mathcal{F}$ for the oblivious polynomial evaluation.

Accommodating the scaling in phase 2

- Where α_1 and α_2 are the parties' respective additive shares, in some finite field (or ring) $\mathcal{F}$, of $\varepsilon \cdot 2^N$ to be delivered by phase 1,

$$\varepsilon \;=\; (\alpha_1 +_{\mathcal{F}} \alpha_2)/2^N$$

- Scaling the phase 2 output up by factor σ, the Taylor series of (2) becomes

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

- ... But we will need a finite polynomial over $\mathcal{F}$ for the oblivious polynomial evaluation.

Accommodating the scaling in phase 2

- Where α_1 and α_2 are the parties' respective additive shares, in some finite field (or ring) $\mathcal{F}$, of $\varepsilon \cdot 2^N$ to be delivered by phase 1,

$$\varepsilon = (\alpha_1 +_{\mathcal{F}} \alpha_2)/2^N$$

- Scaling the phase 2 output up by factor σ, the Taylor series of (2) becomes

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

- ... But we will need a finite polynomial over $\mathcal{F}$ for the oblivious polynomial evaluation.

From Taylor series over $\mathbb{R}$ to polynomial over $\mathcal{F}$

- Truncate the series at k terms for the desired accuracy.
- **If** the numerator will always be divisible by the denominator (in $\mathbb{Z}$); and ...
- **if** we use an $\mathcal{F}$ large enough so that, where $m = |\mathcal{F}|$, all values in the recursive evaluation are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; ...
- **then** we can reinterpret the additions and multiplications, and even the divisions, as the corresponding operations in $\mathcal{F}$, ...
- allowing us to replace 'α_2' with variable 'y', then open parentheses and collect terms to arrive at a polynomial over $\mathcal{F}$ for oblivious polynomial evaluation.

From Taylor series over $\mathbb{R}$ to polynomial over $\mathcal{F}$

- ☐ Truncate the series at k terms for the desired accuracy.
- ☐ **If** the numerator will always be divisible by the denominator (in $\mathbb{Z}$); and ...
- ☐ **if** we use an $\mathcal{F}$ large enough so that, where $m = |\mathcal{F}|$, all values in the recursive evaluation are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; ...
- ☐ **then** we can reinterpret the additions and multiplications, and even the divisions, as the corresponding operations in $\mathcal{F}$, ...
- ☐ allowing us to replace 'α_2' with variable 'y', then open parentheses and collect terms to arrive at a polynomial over $\mathcal{F}$ for oblivious polynomial evaluation.

From Taylor series over $\mathbb{R}$ to polynomial over $\mathcal{F}$

- Truncate the series at k terms for the desired accuracy.
- **If** the numerator will always be divisible by the denominator (in $\mathbb{Z}$); and ...
- **if** we use an $\mathcal{F}$ large enough so that, where $m = |\mathcal{F}|$, all values in the recursive evaluation are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; ...
- **then** we can reinterpret the additions and multiplications, and even the divisions, as the corresponding operations in $\mathcal{F}$, ...
- allowing us to replace 'α_2' with variable 'y', then open parentheses and collect terms to arrive at a polynomial over $\mathcal{F}$ for oblivious polynomial evaluation.

From Taylor series over $\mathbb{R}$ to polynomial over $\mathcal{F}$

- Truncate the series at k terms for the desired accuracy.
- **If** the numerator will always be divisible by the denominator (in $\mathbb{Z}$); and ...
- **if** we use an $\mathcal{F}$ large enough so that, where $m = |\mathcal{F}|$, all values in the recursive evaluation are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; ...
- **then** we can reinterpret the additions and multiplications, and even the divisions, as the corresponding operations in $\mathcal{F}$, ...
- allowing us to replace 'α_2' with variable 'y', then open parentheses and collect terms to arrive at a polynomial over $\mathcal{F}$ for oblivious polynomial evaluation.

From Taylor series over $\mathbb{R}$ to polynomial over $\mathcal{F}$

- Truncate the series at k terms for the desired accuracy.
- **If** the numerator will always be divisible by the denominator (in $\mathbb{Z}$); and ...
- **if** we use an $\mathcal{F}$ large enough so that, where $m = |\mathcal{F}|$, all values in the recursive evaluation are always integers in the interval $[-\lfloor \frac{m}{2} \rfloor, \lfloor \frac{m}{2} \rfloor]$; ...
- **then** we can reinterpret the additions and multiplications, and even the divisions, as the corresponding operations in $\mathcal{F}$, ...
- allowing us to replace 'α_2' with variable 'y', then open parentheses and collect terms to arrive at a polynomial over $\mathcal{F}$ for oblivious polynomial evaluation.

Setting the scale-up: the original Lindell-Pinkas version

- Lindell and Pinkas set the scale-up factor σ at $2^N \operatorname{lcm}(2,\ldots,k)$, giving the truncated Taylor series

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

- In the numerator,

$$(\alpha_1 +_{\mathcal{F}} \alpha_2)^i = (\varepsilon \cdot 2^N)^i = \varepsilon^i \cdot 2^{Ni}$$

- Yet this is **not** generally divisible by $2^{N(i-1)}$.

Setting the scale-up: the original Lindell-Pinkas version

- Lindell and Pinkas set the scale-up factor σ at $2^N \operatorname{lcm}(2, \ldots, k)$, giving the truncated Taylor series

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

- In the numerator,

$$(\alpha_1 +_{\mathcal{F}} \alpha_2)^i = (\varepsilon \cdot 2^N)^i = \varepsilon^i \cdot 2^{Ni}$$

- Yet this is **not** generally divisible by $2^{N(i-1)}$.

- Lindell and Pinkas set the scale-up factor σ at $2^N \operatorname{lcm}(2,\ldots,k)$, giving the truncated Taylor series

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

- In the numerator,

$$(\alpha_1 +_{\mathcal{F}} \alpha_2)^i = (\varepsilon\cdot 2^N)^i = \varepsilon^i\cdot 2^{Ni}$$

- Yet this is **not** generally divisible by $2^{N(i-1)}$.

Brute-force scale-up is not too expensive!

- Brute-force solution: We set σ at $2^{Nk}\,\mathrm{lcm}(2,\ldots,k)$, giving the truncated Taylor series

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

- Surprisingly, this does not require that $\mathcal{F}$ be significantly larger!

- But are other modules in the invoking modular protocol now saddled with the expense of the larger scaling factor?

Brute-force scale-up is not too expensive!

- Brute-force solution: We set σ at $2^{Nk}\,\mathrm{lcm}(2,\ldots,k)$, giving the truncated Taylor series

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

- Surprisingly, this does not require that $\mathcal{F}$ be significantly larger!
- But are other modules in the invoking modular protocol now saddled with the expense of the larger scaling factor?

Brute-force scale-up is not too expensive!

- Brute-force solution: We set σ at $2^{Nk}\operatorname{lcm}(2,\ldots,k)$, giving the truncated Taylor series

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

- Surprisingly, this does not require that $\mathcal{F}$ be significantly larger!
- But are other modules in the invoking modular protocol now saddled with the expense of the larger scaling factor?

Arbitrary scaling: naive Yao recourse

- Scaling **up** by an **integer** factor:
  autonomously by the parties, no problem.
- Scaling **down** by an integer factor, or, more generally, scaling by a **non-integer** factor:
  requires an SMC episode.
- Autonomous scaling by a non-integer factor is not possible—even to integer approximation! **Approximate division does not distribute over modular addition.**
- A Yao SMC episode can accomplish arbitrary scaling, but division and table look-ups are expensive.

Arbitrary scaling: naive Yao recourse

- Scaling **up** by an **integer** factor:
  autonomously by the parties, no problem.
- Scaling **down** by an integer factor, or, more generally, scaling by a **non-integer** factor:
  requires an SMC episode.
- Autonomous scaling by a non-integer factor is not possible—even to integer approximation! **Approximate division does not distribute over modular addition.**
- A Yao SMC episode can accomplish arbitrary scaling, but division and table look-ups are expensive.

Arbitrary scaling: naive Yao recourse

- Scaling **up** by an **integer** factor:
  autonomously by the parties, no problem.
- Scaling **down** by an integer factor, or, more generally, scaling by a **non-integer** factor:
  requires an SMC episode.
- Autonomous scaling by a non-integer factor is not possible—even to integer approximation! **Approximate division does not distribute over modular addition.**
- A Yao SMC episode can accomplish arbitrary scaling, but division and table look-ups are expensive.

Arbitrary scaling: naive Yao recourse

- Scaling **up** by an **integer** factor:
  autonomously by the parties, no problem.

- Scaling **down** by an integer factor, or, more generally, scaling by a **non-integer** factor:
  requires an SMC episode.

- Autonomous scaling by a non-integer factor is not possible—even to integer approximation! **Approximate division does not distribute over modular addition.**

- A Yao SMC episode can accomplish arbitrary scaling, but division and table look-ups are expensive.

Arbitrary scaling: optimized Yao recourse

- Integer part of scale-up factor σ handled separately, leaving a scale-**down** to compute and add modularly.
- For p parties, only p variants of excess in the simple distribution of the scale-down over p original shares.
- A Yao circuit can
  - accept the parties' original shares;
  - accept the parties' simple-minded autonomous scale-downs;
  - accept a random value from parties 1 through $p-1$;
  - determine from the non-modular sum of the original shares which correction to apply to the autonomous scale-downs, and share the corrected scale-down using the random values.

Arbitrary scaling: optimized Yao recourse

- Integer part of scale-up factor σ handled separately, leaving a scale-**down** to compute and add modularly.
- For p parties, only p variants of excess in the simple distribution of the scale-down over p original shares.
- A Yao circuit can
  - accept the parties' original shares;
  - accept the parties' simple-minded autonomous scale-downs;
  - accept a random value from parties 1 through $p-1$;
  - determine from the non-modular sum of the original shares which correction to apply to the autonomous scale-downs, and share the corrected scale-down using the random values.

Arbitrary scaling: optimized Yao recourse

- Integer part of scale-up factor σ handled separately, leaving a scale-**down** to compute and add modularly.

- For p parties, only p variants of excess in the simple distribution of the scale-down over p original shares.

- A Yao circuit can
  - accept the parties' original shares;
  - accept the parties' simple-minded autonomous scale-downs;
  - accept a random value from parties 1 through $p - 1$;
  - determine from the non-modular sum of the original shares which correction to apply to the autonomous scale-downs, and share the corrected scale-down using the random values.

Arbitrary scaling: imperfect secrecy

- It is possible to trade off the perfection of the perfect secrecy in the sharing for the possibility of autonomous scaling after all—no additional SMC needed!
- Theoretically challenging.
- Eminently practical.

Arbitrary scaling: imperfect secrecy

- ☐ It is possible to trade off the perfection of the perfect secrecy in the sharing for the possibility of autonomous scaling after all—no additional SMC needed!
- ☐ Theoretically challenging.
- ☐ Eminently practical.

Arbitrary scaling: imperfect secrecy

- It is possible to trade off the perfection of the perfect secrecy in the sharing for the possibility of autonomous scaling after all—no additional SMC needed!
- Theoretically challenging.
- Eminently practical.

Benefits for the Lindell-Pinkas logarithm protocol

- Compatibility:
  We can efficiently **reverse unwanted scale-ups** that have entered as technical artifacts.

- Performance:
  We can efficiently **achieve wanted scale-ups**, and so avoid the **table look-up** recommended by Lindell and Pinkas to convert n to $2^N \cdot n \ln 2$ **within the Yao computation** of phase 1.

Benefits for the Lindell-Pinkas logarithm protocol

- Compatibility:
  We can efficiently **reverse unwanted scale-ups** that have entered as technical artifacts.

- Performance:
  We can efficiently **achieve wanted scale-ups**, and so avoid the **table look-up** recommended by Lindell and Pinkas to convert n to $2^N \cdot n \ln 2$ **within the Yao computation** of phase 1.

Implementation

- Yao-circuit generator in Perl.

Implementation

- ☐ Yao-circuit generator in Perl.
- ☐ Fairplay Yao-circuit runner in Java.

Implementation

- Yao-circuit generator in Perl.
- Fairplay Yao-circuit runner in Java.
- Controlling program, invoking Fairplay for phase 1 and implementing the oblivious polynomial evaluation of phase 2, in C.

Implementation

- Yao-circuit generator in Perl.
- Fairplay Yao-circuit runner in Java.
- Controlling program, invoking Fairplay for phase 1 and implementing the oblivious polynomial evaluation of phase 2, in C.
- Bignums and basic cryptographic math from libssl and libcrypto.

Performance

- Both parties running as processes on this laptop.

Performance

- Both parties running as processes on this laptop.
- Intel Pentium M at 1.86 GHz.

N	k	modulus bits	gates	absolute error	time (seconds)
13	4	60	1386	< 0.00458	3.57
22	5	120	2797	< 0.00183	6.16
28	7	210	4732	< 0.00034	10.04

Conclusion

- The Lindell-Pinkas two-party secure logarithm protocol, as it has evolved in the course of our implementation, seems to work well and be quite usable as a module in a complex two-party SMC data-mining protocol.
- SMC usability and performance enhancements will continue.
- ... But SMC can already do much now. The main impediment to real-world application is a **gap in awareness and understanding** of what can already be done with SMC today, a gap that is just beginning to be addressed.