Two embarrassingly parallel methods for Secure Multiparty Computation: Point Counting

Toomas Krips, Jan Willemson

October 4, 2014

University of Tartu, Department of Mathematics and Computer Science, STACC
Cybernetica AS

Point-Counting:Basic Idea

- Have a set of points.

Point-Counting:Basic Idea

- Have a set of points.
- Perform some test on every point.

Point-Counting:Basic Idea

- Have a set of points.
- Perform some test on every point.
- (Weighted) sum of points that pass the test is (proportional to) our answer.

Point-Counting:Basic Idea

- Have a set of points.
- Perform some test on every point.
- (Weighted) sum of points that pass the test is (proportional to) our answer.
- Good because round complexity very low.

Secure Multiparty Computation

- How to compute with private/secret/encrypted data?
- $[\![x]\!]$ will mean that the value of x is secret.

More precisely about our case

- We already have some existing protocols, but:
- Program flow MAY NOT depend on private data.
- Parallel execution very-very desirable.

Ingredients

- You will need the following ingredients:

Ingredients

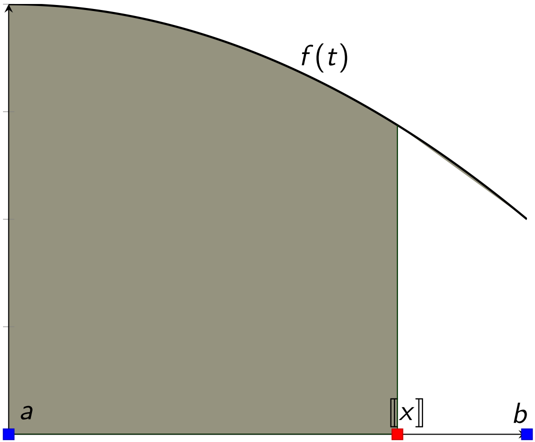
- You will need the following ingredients:
- Ability to represent non-integer numbers (such as floats or fixes) and on them do:

Ingredients

- You will need the following ingredients:
- Ability to represent non-integer numbers (such as floats or fixes) and on them do:
- Cheap addition of private values.
- Comparison $[\![c]\!] = \begin{cases} 0 & \text{if } [\![x]\!] \leq [\![y]\!] \\ 1 & \text{if } [\![x]\!] > [\![y]\!] \end{cases}$
- Multiplication of private and public values.
- The "test" that is applied to all points.

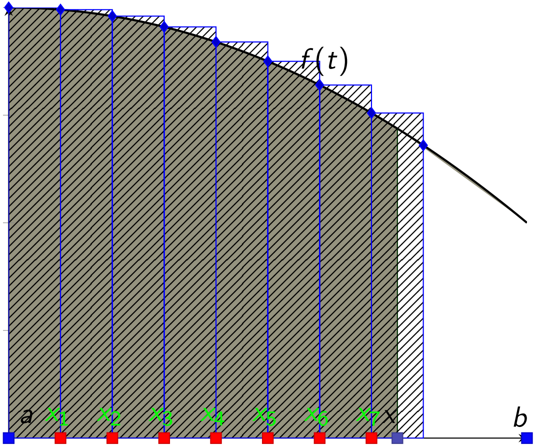
Riemann sums

Want to compute $\int_a^{[\![x]\!]} f(t)dt$, $[\![x]\!]$ is secret, we know $x \in [a, b)$.



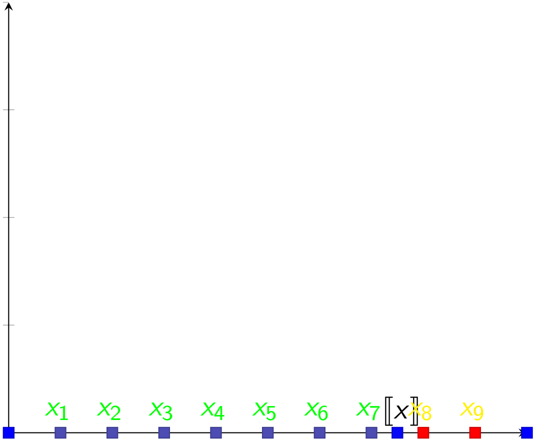
Riemann sums

In non-secret world use rectangle formula. For a small h, compute $x_i = a + i \cdot h$ for $x_i \in [a, x)$ and let answer be $\sum_i f(x_i)h$.



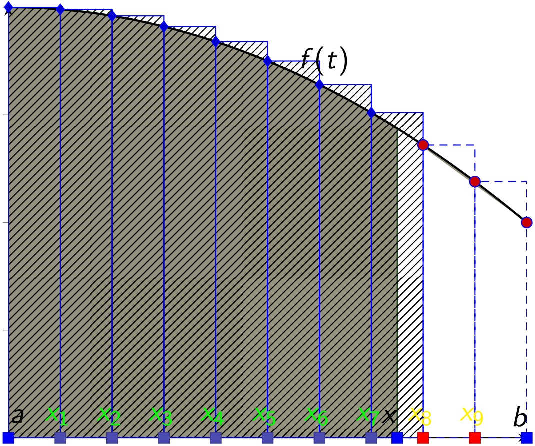
Riemann sums

Similar solution for secret world. Let $x_i = a + i \cdot h$ for $x_i \in [a, b)$ and securely compute $[\![c_i]\!] = x_i \stackrel{?}{\leq} [\![x]\!]$.



Riemann sums

Now compute $\sum_{x_i \in [a,b)} c_i f(x_i) h$.



Functions with easily computable inverses

- Consider functions that are a bit tricky to compute but for which there is a "reverse function" that is much easier to compute.
- For example: computing $\sqrt{x}$ requires computing a series and works only locally, but computing x^2 requires only one computation and works globally.

Functions with easily computable inverses: Theorem

Theorem

Let f be a function. Let g and h be such functions that $g(f(x)) = h(x)$, g is strictly monotonous. Let x be such that $f(x) \in [a, a + 2^k)$. Let $y_0, y_1, \ldots, y_{2^s}$ be such that $y_i := a + i \cdot 2^{k-s}$. Let $j := |\{y_i | g(y_i) < h(x)\}|$. Then $f(x) \in [y_j, y_{j+1})$ if g is monotonously increasing and $f(x) \in [y_{2^s-j-1}, y_{2^s-j})$ if it is monotonously decreasing.

Brief argument

- $y_i := a + i \cdot 2^{k-s}$.
- We know that $f(x) \in [y_r, y_{r+1})$ for some r.

Brief argument

- $y_i := a + i \cdot 2^{k-s}$.
- We know that $f(x) \in [y_r, y_{r+1})$ for some r.
- All y_i where $i \leq r$, pass the test (i.e. $g(y_i) \leq h(x)$), all y_i where $i > r$, don't.

Brief argument

- $y_i := a + i \cdot 2^{k-s}$.
- We know that $f(x) \in [y_r, y_{r+1})$ for some r.
- All y_i where $i \leq r$, pass the test (i.e. $g(y_i) \leq h(x)$), all y_i where $i > r$, don't.
- Thus $j = |\text{number of } i \text{ that pass the test}| = r$.

So

- If we know that $f(\llbracket x \rrbracket) \in [\llbracket a \rrbracket, \llbracket a + 2^k \rrbracket)$ and f is as described in theorem.

So

- If we know that $f(\llbracket x \rrbracket) \in [\llbracket a \rrbracket, \llbracket a + 2^k \rrbracket)$ and f is as described in theorem.
- Compute in parallel $g(\llbracket y_i \rrbracket)$.

So

- If we know that $f(\llbracket x \rrbracket) \in [\llbracket a \rrbracket, \llbracket a + 2^k \rrbracket)$ and f is as described in theorem.
- Compute in parallel $g(\llbracket y_i \rrbracket)$.
- Compute $h(\llbracket x \rrbracket)$.

So

- If we know that $f(\llbracket x \rrbracket) \in [\llbracket a \rrbracket, \llbracket a + 2^k \rrbracket)$ and f is as described in theorem.
- Compute in parallel $g(\llbracket y_i \rrbracket)$.
- Compute $h(\llbracket x \rrbracket)$.
- Comprare in parallel $\llbracket c_i \rrbracket = \llbracket g(y_i) \rrbracket \overset{?}{\leq} \llbracket h(x) \rrbracket$

So

- If we know that $f(\llbracket x \rrbracket) \in [\llbracket a \rrbracket, \llbracket a + 2^k \rrbracket)$ and f is as described in theorem.
- Compute in parallel $g(\llbracket y_i \rrbracket)$.
- Compute $h(\llbracket x \rrbracket)$.
- Comprare in parallel $\llbracket c_i \rrbracket = \llbracket g(y_i) \rrbracket \overset{?}{\leq} \llbracket h(x) \rrbracket$
- Set $\llbracket a \rrbracket + \sum \llbracket c_i \rrbracket \cdot 2^k$ to be the answer.

2^sary search

- Note that we began with knowledge that $[\![f(x)]\!] \in [a_1, a_1 + 2^{k_1})$ and ended with much finer knowledge that $[\![f(x)]\!] \in [a_2, a_2 + 2^{k_2})$.

2^sary search

- Note that we began with knowledge that $[\![f(x)]\!] \in [a_1, a_1 + 2^{k_1})$ and ended with much finer knowledge that $[\![f(x)]\!] \in [a_2, a_2 + 2^{k_2})$.
- We can iterate!

2^sary search

- Note that we began with knowledge that $[\![f(x)]\!] \in [a_1, a_1 + 2^{k_1})$ and ended with much finer knowledge that $[\![f(x)]\!] \in [a_2, a_2 + 2^{k_2})$.
- We can iterate!
- (Good if we reach bounds of parallelisation)

2^sary search

- Note that we began with knowledge that $[\![f(x)]\!] \in [a_1, a_1 + 2^{k_1})$ and ended with much finer knowledge that $[\![f(x)]\!] \in [a_2, a_2 + 2^{k_2})$.
- We can iterate!
- (Good if we reach bounds of parallelisation)
- If we can perform m operations in parallel, then for n-bit increase in accuracy we will need $O(\frac{n}{m})$ time.