SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics

presenter: Le Chen

Nanyang Technological University

lechen0213@gmail.com

December 28, 2013

Reference

- M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. *SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics*, in 19th USENIX Security Symposium, August 2010.

Shamir Secret Sharing

- A random polynomial f of degree $t = \lfloor (m-1)/2 \rfloor$ over a prime field $\mathbb{Z}_p$ with $p > s$, such that $f(0) = s$.
- Each player $i = 1...m$ receives an evaluation point $s_i = f(i)$ of f. Then s_i is called the share of player i.
- The secret s can be reconstructed from any $t+1$ shares using Lagrage interpolation but is completely undefined for t or less shares.

Shamir Secret Sharing

- $s_i = f(i)$
- $s = \sum_i \lambda_i s_i = f(0)$
- where $\lambda_i = \prod_{j, j \neq i} \frac{j}{j-i}$
- We use $[s]$ to denote the vector of shares $(s_1, ..., s_m)$ and call it *sharing* of s. In addition, we use $[s]_i$ to refer to s_i.

Addition

- Given two sharings $[a]$ and $[b]$.
- Since Shamir's scheme is linear, addition of two sharings, denoted by $[a] + [b]$, can be computed by having each player locally add his shares of the two values: $[a + b]_i = [a]_i + [b]_i$.
- $a + b = \sum_i \lambda_i a_i + \sum_i \lambda_i b_i = \sum_i \lambda_i (a_i + b_i)$.
- For a public constant c, $[a + c]_i = [a]_i + c$, $[ca]_i = c[a]_i$.

Multiplication

- What about multiplication?
- $a \times b = (\sum_i \lambda_i a_i) \times (\sum_i \lambda_i b_i)$.
- Each player only knows a_i and b_i.

Multiplication

- Turn back to the polynomial: $a = f_1(0)$, $b = f_2(0)$.
- Let $f = f_1 f_2$, then $ab = f(0)$. Note that the degree of f is $2t$, thus $2t + 1$ shares are needed to reconstruct ab.
- If $2t + 1$ shares are not available, then each player needs to share $d_i = [a]_i[b]_i$ to all players. (communication overhead m^2)

Comparison

- Unlike addition and multiplication, comparison (equality check, less-than comparison) of two shared secrets is a very expensive operation.
- Let $l = \log_2(p)$, Damgard et al.'s scheme: comparison with $205l + 188l \log_2 l$ multiplications in 44 rounds, equality test with $98l + 94l \log_2 l$ multiplications in 39 rounds.
- Nishide and Ohta's scheme: comparison with $279l + 5$ multiplications in 15 rounds, equality test with $81l$ multiplications in 8 rounds.

Comparison

- Our key observation for improving efficiency is the following: For scenarios with many parallel protocol invocations it is possible to build much more practical protocols by not enforcing the constant-round property.
- We design protocols that run in $O(l)$ rounds and are therefore not constant-round, although, once the field size p is defined, the number of rounds is also fixed, i.e., not varying at runtime. e.g. IPv4 address $l = 32$.

Comparison

The overall local running time of a protocol is determined by

- i) the local CPU time spent on computations,
- ii) the time to transfer intermediate values over the network,
- iii) delay experienced during synchronization.

Designing constant-round protocols aims at reducing the impact of iii) by keeping the number of rounds fixed and usually small. To achieve this, high multiplicative constants for the number of multiplications are often accepted (e.g., 279l).
Yet, both i) and ii) directly depend on the number of multiplications.

Equality Test

- Fermat's little theorem states

$$c^{p-1} = 0 \ \texttt{if} \ \ c = 0$$

$$c^{p-1} = 1 \ \texttt{if} \ \ c \neq 0$$

- equality test:

$$equal([a],[b]) := 1 - ([a] - [b])^{p-1}$$

The output of equal is [1] in case of equality and [0] otherwise.

Equality Test

- Using square-and-multiply for the exponentiation, we implement equal with $l + k - 2$ multiplications in l rounds, where k denotes the number of bits set to 1 in $p - 1$.
- By using carefully picked prime numbers with $k \leq 3$, we reduce the number of multiplications to $l + 1$. In the above example for comparing IPv4 addresses, this reduces the multiplication count by a factor of 76 from 2592 to 34.

Less Than

- The protocol uses the observation that $a < b$ is determined by the three predicates $a < p/2$, $b < p/2$, and $a - b < p/2$.
- Each predicate is computed by a call of the LSB(Least-significant bit) protocol for $2a$, $2b$, and $2(a - b)$.
- If $a < p/2$, no wrap-around modulo p occurs when computing $2a$, hence $\mathrm{LSB}(2a) = 0$. However, if $a > p/2$, a wrap-around will occur and $\mathrm{LSB}(2a) = 1$.

Less Than

- A call of $lessThan([a],[b])$ outputs [1] if $a < b$ and [0] otherwise.
- The overall complexity of *lessThan* is $24l + 5$ multiplications in $2l + 10$ rounds as compared to Nishide's version with $279l + 5$ multiplications in 15 rounds.

Short Range Check

- Consider one wanted to compute $[a] < T$, where T is a small public constant, e.g., $T = 10$. Instead of invoking $lessThan([a], T)$ one can simply compute the polynomial $[R] = [a]([a] - 1)([a] - 2)...([a] - (T - 1))$.

- We define a protocol for checking short public ranges that returns $[1]$ if $x \leq [a] \leq y$ and $[0]$ otherwise:

$$shortRange([a], x, y) := equal(0, \prod_{i=x}^{y}([a] - i))$$

Conclusion & Discussion

- This paper uses Shamir's secret sharing to compute $[a] - [b], [a][b]$ efficiently.
- By reducing the number of multiplications, this paper improves efficiency of computing equality test, less than, and short range check.
- The details of proposed protocol is omitted, please refer to the paper if interested.