Efficient Implementation of the Orlandi Protocol

Thomas P. Jakobsen[1], Marc X. Makkes[2], and **Janus Dam Nielsen**[1]

[1]**The Alexandra Institute**
[2]Eindhoven University of Technology

Applied Cryptography and Network Security, 2010

Outline

Outline

1. What (is it all about?)
   - What is Secure Multiparty Computation
   - What is the Orlandi Protocol
2. Why (is the Orlandi protocol interesting?)
   - Active security and self-trust
   - Its practical
   - Solves real-world problems
3. How (did we make it practical?)
   - The Orlandi Protocol in VIFF
   - Efficient Paillier is required
   - Rewrite key steps in C
4. Summary

Secure Multiparty Computation (SMC)

In Secure Multiparty Computation (SMC) we have:

- a number of parties $P_1, \ldots, P_n$
- each having input x_i
- the parties wish to jointly compute a function $y = f(x_1, \ldots, x_n)$
- s.t. x_i is not revealed to others than P_i and y is correct

The Millionaires Example

Two millionaires, want to know who is richer, without revealing the precise amount of their wealth.

Andrew C. Yao, “Protocols for Secure Computations” (1982).

What problems does SMC solve?

SMC enables joint computation on confidential information:

- information can be a resource of vital importance and considerable economic value
- confidentiality of information can be crucial
- significant value can often be obtained by combining confidential information

Real-world Examples

- Auctions
- Benchmarking (e.g. total CO_2 emission from all cargo ships)
- Online games (e.g. poker - only I should learn the value of my cards)
- Procurements
- Data mining

Outline

1. What (is it all about?)
   - What is Secure Multiparty Computation
   - What is the Orlandi Protocol
2. Why (is the Orlandi protocol interesting?)
   - Active security and self-trust
   - Its practical
   - Solves real-world problems
3. How (did we make it practical?)
   - The Orlandi Protocol in VIFF
   - Efficient Paillier is required
   - Rewrite key steps in C
4. Summary

High level Description

Protocol for secure multiparty computation:

- let $s = \sum_{i=1}^{n} s_i \bmod p$ where $s_i \in \mathbb{Z}_p$ then a share is (s_i, C)
- allows $+$, $-$, and $*$
- addition and subtraction are straight forward in an additive scheme
- multiplication is separated into a preprocessing and an online part
- preprocessing creates a set of triples (a, b, c) s.t. $a * b = c$
- online part does actual multiplication and one multiplication consumes one triple

Random Triple Generation

Random Triple Generation takes the security parameter s and a number M as input and generates M triples (a, b, c) s.t. $a * b = c$:

- generate a set of triples $\mathcal{D}$:
  - $\mathcal{D} = \emptyset$
  - For $i = 1, \ldots, \kappa M$ do: $\mathcal{D} = \mathcal{D} \cup \texttt{TripleTest}()$
    (where $\kappa > 1$ is an overhead factor depending on s)
- compute a random subset $\mathcal{T} \subset \mathcal{D}$ and check that they are correct
- use the rest to "distill" M triples

Triple Test and Triple Generation

- Triple Test
  - generates one triple a, b, c
  - uses two triples generated by Triple Generation
  - use one to check the other to reduce the probability for overflow
- Triple Generation:
  - generates one triple a, b, c
  - uses the homomorphic properties of the Paillier cryptosystem
  - encrypted computation could overflow
  - require that $N >> p$

Online Multiplication

Given a triple (a, b, c), multiplication $[x] * [y]$ is defined as:

1. $d = \texttt{Open}([x] - [a])$
2. $e = \texttt{Open}([y] - [b])$
3. $[z] = e[x] + d[y] - de + [c]$

uses *one* broadcast to every party and some local computations - fast.

Outline

Attractive Security Model

- Self-trust - All shares are required to reconstruct the secret values
- Active security - An adversary cannot change a share or deviate from the protocol without the other parties notices
- A corrupt party may block the computation
- 2 to n players

Outline

Experiment Setup

The benchmarks were performed by using 10 identical computers:

- 1 GHz dual-core AMD Opteron 2216 processors with 2x1 Mb level 2 cache
- 2 Gb RAM
- running Red Hat Enterprise Linux 5.2
- 64bit x86 architecture
- gigabit Ethernet, round-trip latency of 0.104 ms.
- 1024-bits key size for the Paillier cryptosystem

One of the machines were used as coordinator.

Online Multiplication

n	2	3	4	5	6	7	8	9
time	27.4	15.9	19.7	22.8	25.6	26.7	28.2	35.9
stdvar	0.1	3.5	4.7	6.7	7.4	6.8	8.1	8.3

Figure: The average *execution time* in ms. as function of the number of parties, n

Random Triple Generation

s	1	1	1	21	21	21
M	5	10	30	5	10	30
2	1.872	1.511	1.370	20.959	16.560	16.453
3	1.598	0.952	1.059	16.931	15.981	15.269
9	2.238	1.799	1.794	31.901	32.572	37.545

Figure: The average execution time in seconds of Random Triple Generation as a function of *parties* (2, 3, and 9), *security parameter* (1 and 21), and *number of triples* (5, 10, and 30)

Outline

1. What (is it all about?)
   - What is Secure Multiparty Computation
   - What is the Orlandi Protocol
2. Why (is the Orlandi protocol interesting?)
   - Active security and self-trust
   - Its practical
   - Solves real-world problems
3. How (did we make it practical?)
   - The Orlandi Protocol in VIFF
   - Efficient Paillier is required
   - Rewrite key steps in C
4. Summary

The Orlandi Protocol is good for

Scenarios which requires self-trust or are know and can be prepared in advance are well-suited for Orlandi Protocol:

- Auctions
- Benchmarking
- Online games (e.g. poker - self-trust)
- Procurements
- Data mining

Outline

Implemented in VIFF

- VIFF - The Virtual Ideal Functionality Framework
- Allows implementation of SMC protocols in a clean and easy way
- Provide means for communication
- Arithmetic with elements from Zp
- Extend the Runtime class and define operations input, add, mul, sub, and output
- Automatic parallel (asynchronous) execution
- Python

Outline

The Key to making the protocol practical

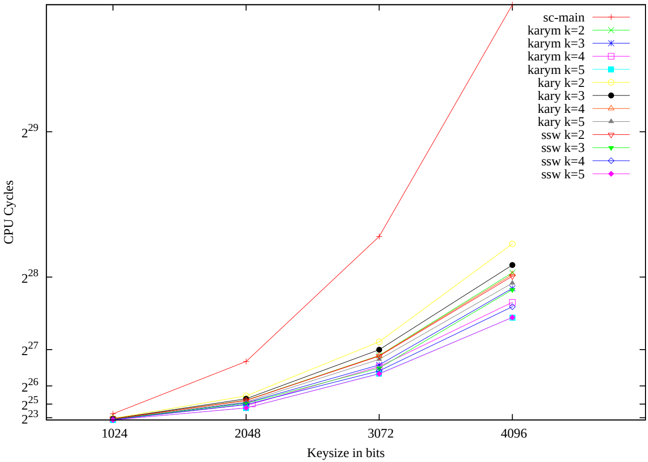
- Used extensively in the Orlandi Protocol
- Homomorphic property
  $\mathrm{Dec}_{dk}(\mathrm{Enc}_{ek}(m_1)\mathrm{Enc}_{ek}(m_2)) = m_1 + m_2$

Paillier

- Main variant ($c = g^m r^N \bmod N$)
- Subgroup variant ($c = g^{m+N+r} \bmod N$)
- Multiexponentiations is vital to performance
- 2^k-ary method uses two aux. tables to evaluate two powers
- 2^k-ary matrix method uses aux. matrix instead of tables - saves a multiplication but gives more pre-computation
- Simultaneous sliding window exponentiation method
- All three methods are benchmarked for varying key-sizes and windows size $1 < k \leq 5$

Efficient Paillier is required

Timings in CPU cycles vs. key size



Without precomputation.

Outline

1. What (is it all about?)
   - What is Secure Multiparty Computation
   - What is the Orlandi Protocol
2. Why (is the Orlandi protocol interesting?)
   - Active security and self-trust
   - Its practical
   - Solves real-world problems
3. How (did we make it practical?)
   - The Orlandi Protocol in VIFF
   - Efficient Paillier is required
   - Rewrite key steps in C
4. Summary

Preprocessing is Slow in Python

- step 2b of Triple Generation: $\gamma_{i,j} = \alpha_i^{b_j} \mathrm{Enc}_{\mathrm{ek}_i}(1;1)^{d_{i,j}}$, where $\alpha_i = Enc_{ek_i}(a_i)$, $a_i, b_j \in \mathbb{Z}_p$, $d_{i,j} \in \mathbb{Z}_{p^3}$, and ek_i, dk_i are public/private keys.
- step 2b also involves mulitexponentiation
- step 3a of Triple Generation:

$$c_i = \sum_j \mathrm{Dec}_{\mathrm{dk}_i}(\gamma_{i,j}) - \sum_j d_{i,j} \bmod p \quad (1)$$

$$= \mathrm{Dec}_{\mathrm{dk}_i}\left(\prod_j \gamma_{i,j} \bmod N^2\right) - \sum_j d_{i,j} \bmod p \quad (2)$$

- relatively small amount of code

ALEXANDRA INSTITUTE

Summary

- The Orlandi protocol is practical
- The Orlandi protocol can be used to solve interesting problems
- The Orlandi protocol requires fast Paillier
- Key parts have been rewritten in C
- Implementation is partly available as part of the open-source VIFF framework at: `http://www.viff.dk`

Questions

Questions?