15th May 2015

Secure Two Party Computation

A practical comparison of recent protocols

Nick Tutte

University of Bristol

Project Overview

- Secure Two Party Computation (S2C) is a long standing problem in cryptography.
  - How can two mutually distrusting parties collaborate compute a function without revealing their inputs?
- Yao Garbled Circuits are one approach to S2C, but naive Yao Circuits only provide security against passive adversaries.
- Recently many protocols have been proposed to use Yao Circuits to provide security in the presence of active adversaries.
- Our aim was to implement some of these recent protocols so we could compare their performance.

Secure Multiparty Computation

- Parties wish to compute a function $(y_1, y_2) = f(x_1, x_2)$
- But they want the following security properties,
  - Privacy - Participating parties only learn their output.
  - Correctness - Output is correct evaluation of the function *given the inputs*.
  - Independence of inputs - Parties only learn their output from participating.

Yao's Protocol

- Proposed by Andrew Yao in 1986. Whilst it's primarily for Two Party computation it can be extended to Multiparty computation.
- The basic idea,
  - One party is labelled the 'Builder' and the other is the 'Executor'.
  - The Builder constructs a binary circuit representing the function.
  - The Builder then 'garbles' this circuit, encrypting it in such a way that it can be evaluated without revealing information about the inputs, and sends it to the Executor.
  - The Builder's inputs are hardcoded, the Executor's inputs are obtained by Oblivious Transfers.
  - Using these inputs the Executor can then evaluate the circuit and so the function.

Oblivious Transfer

Receiver
Inputs : $b \in \{0, 1\}$
Outputs : x_b

Sender
Inputs : $x_0, x_1 \in \{0, 1\}^l$
Outputs : $\emptyset$

Formal definition of the functionality of a one-out-of-two OT protocol. The Receiver should learn nothing about the value of x_{1-b} and the Sender should learn nothing about b.

We will not dwell on the details of Oblivious Transfer, suffice to say it is possible, if anyone is interested in seeing a concrete protocol I suggest the Naor-Pinkas Oblivious Transfer.

The Security of Yao's Protocol

- As already mentioned Yao Circuits are only secure in the presence of passive adversaries. [1]
- In naive Yao Circuits the Executor has no way to know if the circuit given to it by the Builder actually computes the function it purports to.
- There are two main approaches to overcoming this,
  - Cut and Choose.
  - Commit and Prove.

[1] For a formal proof of this see "A Proof of Security of Yao's Protocol for Two-Party Computation" by Lindell and Pinkas (2006).

Cut and Choose - Concept

- All three protocols we implemented are based on Cut and Choose.
- The Builder now generates many garbled circuits and sends them to the Executor.
- The Executor then picks a set of these circuits and asks the Builder to open them so they can be checked for correctness.
- If all check circuits pass then the rest of the circuits are evaluated.
- So a malicious Builder must now guess which circuits will be checked.

Cut and Choose - Pitfalls

- Cut and Choose seems trivially simple, however it actually raises several new problems. The main ones are,
  - Consistency of Builder's inputs.
  - Consistency of Executor's inputs.
  - Output determination - Now we have many circuits what do we do if their outputs disagree?
- Protocols aiming to provide active security

Lindell-Pinkas 2010

- Based around a new primitive, a Cut and Choose Oblivious Transfer.
- Using an elegant approach to consistency using Zero Knowledge proofs.
- Half of the circuits are check circuits, the other half are evaluation circuits.
- Return majority output of the evaluation circuits.
- Statistical security $2^{-0.311 \cdot S}$ where S is the number of circuits.
- So we need 130 circuits to achieve statistical security of 2^{-40}.

Lindell 2013

- Very similar to the Lindell-Pinkas 2010, uses the same basic ideas.
- Modifies the Cut and Choose Oblivious Transfer.
- Runs a small sub-computation (using Lindell-Pinkas) such that only one circuit needs to be correct.
- So it's expected to do well for large circuits, badly for small circuits.
- Statistical security 2^{-S} , so we need 40 circuits to achieve statistical security of 2^{-40}.

Huang-Katz-Evans 2013

- Developed concurrently to Lindell 2013.
- Uses Symmetric cut and choose. Both parties build circuits, both circuits execute the other's circuits.
- Uses a logarithm based approach to the consistency of inputs.
- Output determination is such that,
  - For each output wire a value is output if at least one of your circuits gives that value *and* at least one of you partner's circuits gives the same value.
- Statistical security $2^{-S+log(S)}$, so we need 46 circuits to achieve statistical security of 2^{-40}.

Merging Lindell and HKE

- The obvious question raised by Lindell 2013 is can we improve it by changing the protocol used for the sub-computation.
- We changed the sub-computation to use HKE instead.
- Not a trivial task,
  - Lindell provides several levels of optimisations to the sub-computation some of which are incompatible with HKE.
  - The Lindell-Pinkas Zero Knowledge Proof approach to cannot work here, we have to switch the whole protocol to use the HKE logarithm based approach.
  - Additionally the output of the sub-computation needs to be hidden from the Builder.

32-bit Addition Circuit (439 gates)

Builder	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	113.96	27.41	7,648,074	737,109
L2013	171.21	42.03	4,693,761	980,193
HKE	45.59	6.77	3,143,383	3,143,366
L-HKE	145.77	25.47	5,995,366	3,299,399

Executor	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	55.90	27.45	737,109	7,648,074
L-2013	101.69	42.05	980,193	4,693,761
L-HKE	132.51	25.83	3,299,399	5,995,366

HKE runs faster by a very clear margin, while L-2013 uses less bandwidth.

32-bit Addition Circuit (439 gates)

Builder	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	113.96	27.41	7,648,074	737,109
L2013	171.21	42.03	4,693,761	980,193
HKE	45.59	6.77	3,143,383	3,143,366
L-HKE	145.77	25.47	5,995,366	3,299,399

Executor	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	55.90	27.45	737,109	7,648,074
L-2013	101.69	42.05	980,193	4,693,761
L-HKE	132.51	25.83	3,299,399	5,995,366

L-2013 run slower than LP-2010 due to the size of the circuit.

32-bit Addition Circuit (439 gates)

Builder	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	113.96	27.41	7,648,074	737,109
L2013	171.21	42.03	4,693,761	980,193
HKE	45.59	6.77	3,143,383	3,143,366
L-HKE	145.77	25.47	5,995,366	3,299,399

Executor	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP-2010	55.90	27.45	737,109	7,648,074
L-2013	101.69	42.05	980,193	4,693,761
L-HKE	132.51	25.83	3,299,399	5,995,366

Some observations

- A clear victory for HKE in Wall and CPU time. Surprising given it requires many more circuits than the L-2013 and L-HKE protocols.
- L-2013 needs much less bandwidth.

AES-128 Encryption Circuit (33,872 gates)

Builder	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP 2010	480.82	114.98	668,935,684	2,798,517
L 2013	399.27	119.25	210,537,538	1,609,692
HKE	185.47	32.95	238,300,835	238,300,840
L-HKE	417.84	78.22	214,725,419	7,868,176

Executor	CPU Time	Wall Time	Bytes Sent	Bytes Recv
LP 2010	227.91	116.15	2,798,517	668,935,684
L 2013	270.99	119.27	1,609,692	210,537,538
L-HKE	363.46	80.49	7,868,176	214,725,419