Secure Two Party Computation

A practical comparison of recent protocols

Nick Tutte

University of Bristol

Project Background

- Secure Two Party Computation (S2C) - How can a pair of mutually distrusting parties collaborate to compute a function without revealing their inputs?
- Yao's Protocol is one approach to S2C, but in its naive form only provide security against passive adversaries.
- Recently several protocols have been proposed using Cut-and-Choose variants of Yao's protocol to provide security against active adversaries.
- Our aim was to provide the first practical comparison and analysis of these protocol's performance.

Project Motivation

- Without a good comparison of these protocols we cannot know which is the most promising for further research.
- In order to run practical comparisons we need implementations to compare. For two of the three protocols our implementations are the first.
- Furthermore we broke down our results to identify the bottlenecks for each protocol to guide future optimisation

Yao's Protocol

- Proposed by Andrew Yao in 1986. Primarily used for Two Party computation but it can be extended to Multiparty computation.
- The parties are labelled as the 'Builder' and the 'Executor'.
- The Builder constructs a binary circuit representing the function and 'garbles' this circuit.
- Such a garbled circuit can be evaluated without revealing information about the inputs.
- This is only secure if we trust the Builder garbled the circuit honestly.

Cut and Choose - Concept

- The Builder generates *many* garbled circuits and sends them to the Executor.
- The Executor then picks a subset of these circuits and asks the Builder to open them so they can be checked for correctness.
- If all check circuits pass then the remaining circuits are evaluated.
- This gives us statistical security as a malicious Builder must guess which circuit the Executor will open.

Cut and Choose - Pitfalls

- Cut and Choose seems trivially simple, however it actually raises several new problems. The main ones are,
  - Consistency of inputs - Ensuring the parties give the same input to the many circuits, without leaking the inputs to the other party.
  - Output determination - Now we have many circuits what do we do if their outputs disagree?

Lindell-Pinkas 2010 (LP-2010)

- Using Zero Knowledge proofs for the input consistency checks.
- Half of the circuits are check circuits, the other half are evaluation circuits.
- Simply returns 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 (L-2013)

- Builds on Lindell-Pinkas 2010 using the same basic ideas.
- Main difference, runs a small sub-computation (using Lindell-Pinkas) such that only one circuit needs to be correct.
- We expected it to perform well for large circuits and poorly for small circuits.
- Statistical security 2^{-S} , so we need 40 circuits to achieve statistical security of 2^{-40}.

Huang-Katz-Evans 2013 (HKE)

- Uses Symmetric cut and choose. Both parties build circuits, both parties 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 *each* party needs 46 circuits to achieve statistical security of 2^{-40}.

Merging L-2013 and HKE (L-HKE)

- The natural question raised by L-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
L-2013	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 fastest by a very clear margin, while L-2013 uses the least 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
L-2013	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 runs substantially slower than LP-2010 due to the relative cost of the sub-computation being high.

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
L-2013	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-HKE's results are very hopeful compared to LP-2010 and L-2013.

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

HKE still has a commanding lead. While L-2013 closes the gap to LP-2010.

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

At first glance L-HKE is hopeful. However, the breaking down the timings reveals cause for concern.

Some Conclusions

- A clear victory for HKE in Wall and CPU time, despite needing to build/check more circuits.
- Symmetric Cut and Choose merits further work.
- L-2013 is the protocol of choice if bandwidth is at a premium, e,g, when on a metered connection.
- L-HKE shows promise on smaller circuits over L-2013, but is hampered on large circuits by the optimisation roll backs.

Achievements

- We provide the first prototype implementations of LP-2010 and L-2013
- We propose and prototype a variant of L-2013 using HKE for the sub-computation.
- We provide the first practical comparison of LP-2010, L-2013, HKE and our variant. Along with breakdown of results.
- We suggest optimisations for LP-2010 and L-2013.

Future Work

- Change the LP-2010 and L-2013 protocols to use the Logarithm consistency.
- Testing on a wider variety of circuits and with parties with unequal computational capabilities.
- Further research into optimising HKE.
- Reconcile some of the L-2013 sub-computation optimisations with HKE in L-HKE.
- Optimisation the sub-computation output hiding in L-HKE.