University of BRISTOL

15th May 2015

Secure Two Party Computation

A practical comparison of recent protocols

Nick Tutte

University of Bristol

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.
  - How can mutually distrusting parties collaborate compute a function without revealing their inputs?

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.
  - How can mutually distrusting parties collaborate compute a function without revealing their inputs?
- A subset of this problem is Secure Two Party Computation (S2C).

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.
  - How can mutually distrusting parties collaborate compute a function without revealing their inputs?
- A subset of this problem is Secure Two Party Computation (S2C).
- Yao Garbled Circuits are one approach to S2C, but naive Yao Circuits only provide security against passive adversaries.

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.
  - How can mutually distrusting parties collaborate compute a function without revealing their inputs?
- A subset of this problem is Secure Two Party Computation (S2C).
- 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.

Project Overview

- Secure Multiparty Computation is a long standing problem in cryptography.
  - How can mutually distrusting parties collaborate compute a function without revealing their inputs?
- A subset of this problem is Secure Two Party Computation (S2C).
- 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

- Given n parties who each have an input x_i

Secure Multiparty Computation

- Given n parties who each have an input x_i
- Parties wish to compute a function $(y_1, ..., y_n) = f(x_1, ..., x_n)$

Secure Multiparty Computation

- Given n parties who each have an input x_i
- Parties wish to compute a function $(y_1, ..., y_n) = f(x_1, ..., x_n)$
- But they want the following security properties,

Secure Multiparty Computation

- Given n parties who each have an input x_i
- Parties wish to compute a function $(y_1, ..., y_n) = f(x_1, ..., x_n)$
- But they want the following security properties,
  - Privacy - Participating parties only learn their output.

Secure Multiparty Computation

- Given n parties who each have an input x_i
- Parties wish to compute a function $(y_1, ..., y_n) = f(x_1, ..., x_n)$
- 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*.

Secure Multiparty Computation

- Given n parties who each have an input x_i
- Parties wish to compute a function $(y_1, ..., y_n) = f(x_1, ..., x_n)$
- 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 - Participating parties only learn their output.

Yao's Protocol

- Proposed by Andrew Yao in 1986. Whilst it's primarily for Two Party computation it can be extended to Multiparty computation.

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,

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'.

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.

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.

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.

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	**Sender**
Inputs : $b \in \{0,1\}$	Inputs : $x_0, x_1 \in \{0,1\}^l$
Outputs : x_b	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]

[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)

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.

[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)

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,

[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)

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.

[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)

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.

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.

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.

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.

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 - Issues

- Cut and Choose seems trivially simple, however it actually raises several new problems.