A Quick Tour of Secure Multiparty Computation (MPC)
int main (int argc,
char** argv) {
...
}

// F.c
Trusted Third Party

// F.c
int main (int argc, char** argv) {
    ...
}

x

y

z
int main (int argc, char** argv) {
    ...
}

// F.c

Trusted Third Party

x

y

z
Trusted Third Party

\[ F(x, y, z) \]
Trusted Third Party

$F(x, y, z)$

$x$

Confidentiality

Integrity

$y$

“Learn nothing but the output”
Trusted Third Party

\[ F(x, y, z) \]
Confidentiality
Integrity
Secure Multiparty Computation

How can we use cryptography to emulate the existence of a trusted third party so that we can run arbitrary programs on joint private inputs?
Secure Auctions

1 Introduction and History

In multiparty computation (MPC), we consider a number of players $P_1, \ldots, P_n$, who initially each hold inputs $x_1, \ldots, x_n$, and we want to securely compute some function $f(x_1, \ldots, x_n)$ on these inputs, where $f$ is a public polynomial. Each player $P_i$ has a secret input $x_i$, and no player knows the secret inputs of another player. Also, no player may deviate from the protocol. With this protocol we can, in principle, solve very general cryptographic problems. The general theory of MPC was developed in the late 70s (86,3,7,2). The theory was later developed in several ways — see for instance (31,5,8). An overview of the theoretical results known can be found in [10].

Despite the obvious potential that MPC has in solving a wide range of problems, we have seen very little practical application of MPC in the past. This is probably in part due to the fact that the security of the protocol is more general than the security of the protocol. In addition, the computational overhead is more general than the overhead in the protocol itself.

A different view of research has been taken from a range of economic applications. We are particularly interested in the practical use. This approach was taken, for instance, by one researcher who has the authors of the paper have been involved in SCRDE (Secure Computing, Coding and Trust) and MDAC (Secure Information and Management and Processing) [20], which has been responsible for the practical applications of MPC described in this paper. In the economic field of cryptography, there are several ways to implement a secure auction. One of these ways is to use a secure auctioneer. The auctioneer is responsible for the actual auction process. The auctioneer is responsible for the actual auction process. In addition, there is a need for a secure auctioneer.

Variants of secure auction exist under the following assumptions. The most well-known is probably the so-called happy last auction with secret bid, however, a term of interest another common variant is the so-called sealed-bid auction with many solvers and bidders. This auction handles common auction where two wants to sell a fair market price for a commodity given the existing supply and demand in the market.

* This work was sponsored by the Defense Strategic Research Council.
* "MPC" stands for "multiparty computation," a term that is used to describe the field of secure computation among multiple parties with incomplete information. It is essential for secure transactions in various areas such as e-commerce, finance, and distributed systems.
Secure Auctions

Privacy-preserving studies
Secure Auctions

Privacy-preserving studies
Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising
Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising

Privacy-preserving analytics

(Secure Machine Learning)
Secure Auctions

Privacy-preserving advertising

Privacy-preserving analytics

Secure Machine Learning

Financial Fraud Detection

Privacy-preserving studies

Differentially Private Secure Multi-Party Computation for Financial Applications

(Video-privacy-preserving analytics)
Secure Auctions

Privacy-preserving studies

Privacy-preserving advertising

Privacy-preserving analytics

(Finite Machine Learning)

Financial Fraud Detection

...and much more
H ow to run any program...

For parties that are honest but curious (semi-honest)
1-out-of-2 Oblivious Transfer
Sender

$m_0, m_1$

1-out-of-2 Oblivious Transfer

Receiver
$m_0, m_1 \quad b \in \{0,1\}$
Sender $m_0, m_1$ 1-out-of-2 Oblivious Transfer $b \in \{0,1\}$ Receiver $m_b$
\[ m_0, m_1 \]

1-out-of-2 Oblivious Transfer

\[ b \in \{0,1\} \]

\[ m_b \]

Sender

Receiver
OT is a standard cryptographic primitive, and there are many protocols that implement it.
A Boolean Circuit is a directed acyclic graph where
• Each node has fan-in two (and unbounded fan-out).
• Each node has a label $\land$ or $\oplus$
• There are two distinguished wires labelled 0 and 1
A Boolean Circuit is a directed acyclic graph where:

- Each node has fan-in two (and unbounded fan-out).
- Each node has a label $\land$ or $\oplus$.
- There are two distinguished wires labelled 0 and 1.
A Boolean Circuit is a directed acyclic graph where

- Each node has fan-in two (and unbounded fan-out).
- Each node has a label $\land$ or $\oplus$
- There are two distinguished wires labelled 0 and 1
A Boolean Circuit is a directed acyclic graph where
- Each node has fan-in two (and unbounded fan-out).
- Each node has a label $\land$ or $\oplus$
- There are two distinguished wires labelled 0 and 1
A Boolean Circuit is a directed acyclic graph where

- Each node has fan-in two (and unbounded fan-out).
- Each node has a label $\wedge$ or $\oplus$
- There are two distinguished wires labelled 0 and 1
A Boolean Circuit is a directed acyclic graph where

- **Each node has fan-in two (and unbounded fan-out).**
- **Each node has a label ∧ or ⊕**
- **There are two distinguished wires labelled 0 and 1**
Fact: \{ \land, \oplus, 1 \} is a complete Boolean basis.

For any Boolean function \( f : \{0,1\}^n \to \{0,1\}^m \), there exists a Boolean circuit over \{ \land, \oplus, 1 \} that computes \( f \).

I.e., Boolean circuits can compute any bounded function
GMW Protocol
GMW Protocol

Hint: Use a lot of Oblivious Transfer
Step 1 of GMW:
Express program $F$ as a Boolean circuit $C$
$a, b$

c, d

\[
\begin{array}{c}
\oplus \\
\oplus \\
\wedge
\end{array}
\]
$a, b, c, d$
a, b

\[ a, b \]

\[ c, d \]
\(a, b\)

\(c, d\)

\[\hat{a}, \hat{c}, \hat{b}, \hat{d}\]

\[\hat{a} \oplus \hat{c} \oplus \hat{b} \oplus \hat{d}\]
\(a, b\)  \(c, d\)

\[
\begin{align*}
\hat{a} & \quad (a \oplus c)(b \oplus d) \\
\hat{c} & \quad (a \oplus c)(b \oplus d) \\
\hat{b} & \quad (a \oplus c)(b \oplus d) \\
\hat{d} & \quad (a \oplus c)(b \oplus d)
\end{align*}
\]
The XOR secret sharing of a bit $x$ is a pair of bits $\langle x_0, x_1 \rangle$ where $P_0$ holds $x_0$ and $P_1$ holds $x_1$, and where $x_0 \oplus x_1 = x$
The XOR secret sharing of a bit $x$ is a pair of bits $\langle x_0, x_1 \rangle$ where $P_0$ holds $x_0$ and $P_1$ holds $x_1$, and where $x_0 \oplus x_1 = x$

We sometimes denote such a pair by $[x]$
The XOR secret sharing of a bit $x$ is a pair of bits $\langle x_0, x_1 \rangle$ where $P_0$ holds $x_0$ and $P_1$ holds $x_1$, and where $x_0 \oplus x_1 = x$

We sometimes denote such a pair by $[x]$

Intuition: $P_0$'s share $x_0$ acts as a mask, hiding $x$ from $P_1$ (and vice versa)
\[ a, b \quad c, d \]

\[ \hat{a} \quad \hat{c} \quad \hat{b} \quad \hat{d} \]

\[ (a \oplus c)(b \oplus d) \]
\[ (a \oplus c)(b \oplus d) \]
Each party in its head maintains a local copy of the circuit, placing its shares on the wires.
Where do input shares come from?
How do we XOR two shares?
How do we AND two shares?
How do we “decrypt” output shares?
Where do input shares come from?

Goal: put \([x]\) on the input wire
Where do input shares come from?

Goal: put $[x]$ on the input wire

$x$

$r \leftarrow \{0, 1\}$
Where do input shares come from?
Goal: put \([x]\) on the input wire
Where do input shares come from?
Goal: put \([x]\) on the input wire

\[\begin{align*}
    x & \quad \text{Input} \\
    r & \leftarrow \{0, 1\} \\
    x \oplus r & \quad \text{Output}
\end{align*}\]
How do we XOR two shares?

Goal: given gate input wires holding \([x], [y]\),
put \([x \oplus y]\) on the gate output

\[
\begin{align*}
& x_0 \\
& y_0
\end{align*}
\]
How do we XOR two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \oplus y]\) on the gate output
How do we XOR two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \oplus y]\) on the gate output

XOR is “free”
How do we “decrypt” output shares?

Goal: given wire holding $[x]$, reveal $x$ to each party

$x_0$ $x_1$
How do we “decrypt” output shares?

Goal: given wire holding $[x]$, reveal $x$ to each party
Where do input shares come from?
How do we XOR two shares?
How do we AND two shares?
How do we “decrypt” output shares?
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output

\[(x_0 \oplus x_1) \land (y_0 \oplus y_1)\]
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output

\[
(x_0 \oplus x_1) \land (y_0 \oplus y_1)
\]

\[
= (x_0 \land y_0) \oplus (x_0 \land y_1) \oplus (x_1 \land y_0) \oplus (x_1 \land y_1)
\]
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output

\[
\begin{align*}
(x_0 \oplus x_1) \land (y_0 \oplus y_1) &= (x_0 \land y_0) \oplus (x_0 \land y_1) \oplus (x_1 \land y_0) \oplus (x_1 \land y_1)
\end{align*}
\]
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output

\[
(x_0 \oplus x_1) \land (y_0 \oplus y_1) = (x_0 \land y_0) \oplus (x_0 \land y_1) \oplus (x_1 \land y_0) \oplus (x_1 \land y_1)
\]
Important Subgoal

Goal: given gate input bits $x, y$, compute random secret share $[x \land y]$ s.t. neither party learns $x \land y$
Goal: given gate input bits $x, y$, compute random secret share $[x \land y]$ s.t. neither party learns $x \land y$.
Goal: given gate input bits $x, y$, compute random secret share $[x \land y]$ s.t. neither party learns $x \land y$. 

Important Subgoal

$r \leftarrow \{0, 1\}$, $r, r \oplus x$
Goal: given gate input bits $x, y$, compute random secret share $[x \land y]$ s.t. neither party learns $x \land y$.

Important Subgoal

$r \leftarrow \{0, 1\}$

$r, r \oplus x$ \quad OT \quad \rightarrow \quad y$

$r \oplus (x \land y)$
Goal: given gate input bits $x, y$, compute random secret share $[x \land y]$ s.t. neither party learns $x \land y$.

Important Subgoal

$r \leftarrow \{0, 1\}$  \hspace{1cm} $r, r \oplus x$

$x \land y$

$r \oplus (x \land y)$

\[ \langle r, r \oplus (x \land y) \rangle = [x \land y] \]
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output

\[ r \leftarrow \{0,1\} \]

\[ s \leftarrow \{0,1\} \]

\[ r, r \oplus x_0 \quad \rightarrow \quad y_1 \quad \leftarrow \quad r \oplus (x_0 \land y_1) \]

\[ y_0 \quad \rightarrow \quad OT \quad \leftarrow \quad s, s \oplus x_1 \]

\[ s \oplus (x_1 \land y_0) \quad \rightarrow \quad OT \quad \leftarrow \quad x_0 \land y_1 \]
How do we AND two shares?

Goal: given gate input wires holding \([x], [y]\), put \([x \land y]\) on the gate output.

\[
r \leftarrow \{0, 1\}
\]

\[
s \leftarrow \{0, 1\}
\]

\[
\begin{align*}
    y_0 & \rightarrow r, r \oplus x_0 \\
    r \oplus (x_0 \land y_1) & \leftarrow y_1 \\
    s \oplus (x_1 \land y_0) & \rightarrow s, s \oplus x_1 \\
    (s \oplus (r \oplus (y_0 \land x_1) \oplus (x_0 \land y_0)), s \oplus (r \oplus (x_0 \land y_1) \oplus (x_1 \land y_1)) \rangle \\
    & = [x \land y]
\end{align*}
\]
Where do input shares come from?
How do we XOR two shares?
How do we AND two shares?
How do we “decrypt” output shares?

\[
\begin{align*}
[a] & \quad [a \oplus c] \\
[b] & \quad [(a \oplus c)(b \oplus d)] \\
[c] & \quad [b \oplus d] \\
[d] & 
\end{align*}
\]
GMW Protocol

Propagate secret shares from input wires to output wires

Use OT to implement AND gates

Cost:
\[ O(|C|) \text{ OTs} \]

Number of protocol rounds scales with the depth of \( C \)
Now we know how to run any program

What is the MPC field about?

More Parties

Stronger Security Notions

Improved Efficiency