Oblivious Transfer and Secure Multi-Party Computation With Malicious Parties

Vitaly Shmatikov
Reminder: Oblivious Transfer

• A inputs two bits, B inputs the index of one of A’s bits
• B learns his chosen bit, A learns nothing
  - A does not learn which bit B has chosen
  - B does not learn the value of the bit that he did not choose
• Generalizes to bitstrings, M instead of 2, etc.
Reminder: Semi-Honest OT Protocol

Assume the existence of some family of one-way trapdoor permutations

Choose a one-way permutation F; T is a trapdoor, H is the hard-core bit of F

A chooses his input i (0 or 1)

B chooses random x, \( y_{\text{not } i} \)
Computes \( y_i = F(x) \)

\( y_0, y_1 \)

\( b_0 \oplus H(T(y_0)), b_1 \oplus H(T(y_1)) \)

Computes \( b_i = m_i \oplus H(x) \)

How do we force malicious B to follow the protocol and pick \( y_{\text{not } i} \) randomly?
OT With Malicious Parties (Attempt)

A chooses his input $i$ (0 or 1)

Chooses a one-way permutation $F$; $T$ is a trapdoor, $H$ is the hard-core bit of $F$

Picks random $x$

Sets $y_i = F(x)$

Sets $y_{\neg i} = y' \oplus y''$

Chooses a one-way permutation $F$; $T$ is a trapdoor, $H$ is the hard-core bit of $F$

Picks random $y'$

Picks random $y''$

$F, y''$

Picks random $x$

Sets $y_i = F(x)$

Sets $y_{\neg i} = y' \oplus y''$

B

Commit($y'$)

$y_0, y_1$

B picks random $y'$

Computes $b_i = m_i \oplus H(x)$

ZK proof that $\exists i \text{ s.t. } c = \text{commit}(y_{\neg i} \oplus y'')$

$m_0$

$m_1$

$b_0 \oplus H(T(y_0)), b_1 \oplus H(T(y_1))$
Proving Security for Chooser

◆ Chooser in this protocol gives out much more information than in the original protocol
  • Commitment to a random value
  • ZK proof that he used this value in computing $y_{\text{not } i}$

◆ To prove security for Chooser, construct a simulator for Sender’s view (details omitted)
  • Main idea: distribution of $\{c, y_0, y_1\}$ is independent of the bit $i$ indicating Chooser’s choice
  • Intuition: commitment $c$ hides all information about $y'$
  • Intuition: Chooser’s proof is zero-knowledge, thus there exists a simulator for Sender’s view of the proof
Proof of Sender Security (Attempt)

Simulating malicious Chooser’s view of protocol

- Ideal OT functionality
- Sim must extract bit \( j \) that B is using as his real choice

What if B cheated and used a different bit \( j \) in real protocol?!

B’s input bit \( i \)

Pick random \( F, y'' \)

\[ b_i \oplus H(T(y_i)), \text{ random } m_{\not i} \]

ZK proof that, for some \( j \),
\[ c=\text{commit}(y'), y_{\not j}=y'\oplus y'' \]

B will detect the difference between real protocol and simulation if \( i \neq j \)
Extracting Chooser’s Bit (Sketch!)

- Ideal OT functionality
  - $b_k$
  - B’s input bit $i$

Simulator ($\text{Sim}$)

- Pick random $F, y''$
- $y_0, y_1$
- $F, y''$
- $c = \text{commit}(y')$,
  $y_{\not j} = y' \oplus y''$
- ZK proof that, for some $j$,
- ZK proof that, for some $k$
  - $c = \text{commit}(y')$
  - $z_{\not k} = y' \oplus z''$

- One of $y_0, y_1$ is equal to $y' \oplus y''$
- One of $z_0, z_1$ is equal to $y' \oplus z''$

- There is a unique pair of bits $(j, k)$ such that
  $y_{\not j} \oplus y'' = z_{\not k} \oplus z'' = y'$

- Simulator learns $k$!

- Must be same $y'$ because commitment is binding
  - $b_k \oplus H(T(z_k)), \text{ random } m_{\not k}$
  - rewind $B$
So, Is This Protocol Secure?

A

- Chooses a one-way permutation $F$; $T$ is a trapdoor, $H$ is the hard-core bit of $F$

B

- Chooses his input $i$ (0 or 1)

C

- Commit(y')
- Picks random $y'$
- Picks random $y''$
- $F, y''$
- $m_0$
- $m_1$
- $b_0 \oplus H(T(y_0)), b_1 \oplus H(T(y_1))$
- Computes $b_i = m_i \oplus H(x)$

ZK proof that $\exists i$ s.t. $c = \text{commit}(y_{not\ i} \oplus y'')$
Oops!

A

Chooses a one-way permutation $F$; $T$ is a trapdoor, $H$ is the hard-core bit of $F$

C

Commit($y'$)

Picks random $y'$

F, $y''$

Picks random $y''$

$y_0, y_1$

B

Chooses his input $i$ (0 or 1)

This is true, so proof passes!

Note that $y_0 = y_1 = y$

If these values are the same, B learns that $b_0 = b_1$

Would he be able to learn this in the ideal world?

This proof is NOT enough for security against malicious chooser

$y_{not\ i} = y' \oplus y''$
OT Protocol with Malicious Parties

Chooses a one-way permutation $F$; $T$ is a trapdoor, $H$ is the hard-core bit of $F$

Chooses his input $i$ (0 or 1)

F, $y''$

Picks random $y''$

ZKPK($y'$, $x$, $i$) s.t. $c$=commit($y'$), $y_i$=$F(x)$, $y_{not \ i}$=$y'\oplus y''$

Can A learn $y'$, $x$ or $i$ from this proof?

Computes $b_i=m_i\oplus H(x)$

B proves that he executed previous steps correctly
Proving Sender Security

Simulating malicious Chooser’s view of protocol

Ideal OT functionality

Because this is a ZK proof of knowledge, there is an extractor that allows simulator to extract $y'$, $x$ and $i$ (but ZKPK proof system must be secure against a malicious verifier – why?)
Naor-Pinkas Oblivious Transfer

Setting: order-q subgroup of $\mathbb{Z}^*p$, $p$ is prime, $q$ divides $p-1$
$g$ is a generator group for which CDH assumption holds

Messages $m_0$ and $m_1$

Chooser: bit $\sigma$

$S$

$C$

Chooses random $k$
Sets $PK_\sigma = g^k$, $PK_{1-\sigma} = C/PK_\sigma$

$g^r$, $m_0 \oplus \text{Hash}((PK_0)^r, 0)$, $m_1 \oplus \text{Hash}((PK_1)^r, 1)$

Computes $(g^r)^k = (PK_\sigma)^r$ and decrypts $m_\sigma$

Chooser does not know the discrete log of $PK_{1-\sigma}$, thus cannot distinguish between a random value $g_z$ and $(PK_{1-\sigma})^r$ - why?
1-out-of-4 Oblivious Transfer

- Very similar to 1-out-of-2 oblivious transfer
- How to construct a 1-out-of-4 OT protocol given an 1-out-of-2 protocol?
Boolean Circuits

- Alice and Bob want to compute function $f(a,b)$
  - Assume for now Alice and Bob are semi-honest
- First, convert the function into a **boolean circuit**

![Boolean Circuit Diagram]

- Next, parties securely **share** their inputs
Input Sharing

Knows $x_1 \ldots x_n$  \hspace{5cm} Knows $x_{n+1} \ldots x_{2n}$

\begin{align*}
\text{A} & \quad \text{Random } r_1 \ldots r_n \\
& \quad \text{Let } a_i = x_i \oplus r_i \text{ for } 1 \leq i \leq n \\
& \quad \text{Let } b_i = r_i \\
\text{B} & \quad \text{Random } r_{n+1} \ldots r_{2n} \\
& \quad \text{Let } a_i = r_i \text{ for } n+1 \leq i \leq 2n \\
& \quad \text{Let } b_i = x_i \oplus r_i
\end{align*}

\textbf{After this exchange, for all inputs } x_i \ldots \textbf{
\[x_i = a_i \oplus b_i\]}

- Alice still doesn’t know Bob’s inputs, and vice versa
- This is information-theoretically secure
Evaluating an AND Gate

Inputs $x_1$ and $x_2$ are shared between Alice and Bob
- $x_1 = a_1 \oplus b_1$, $x_2 = a_2 \oplus b_2$

At the end of the protocol, Alice learns bit $c_1$ and Bob learns bit $c_2$ such that...
- Bits $c_{1,2}$ are “random”
- $c_1 \oplus c_2 = x_1 \land x_2 = (a_1 \oplus b_1) \land (a_2 \oplus b_2)$
  - Output of the gate is shared between A and B just like the inputs
Use Oblivious Transfer

Pick random $c_1$

- In every case, $c_1 \oplus c_2 = (a_1 \oplus b_1) \land (a_2 \oplus b_2) = x_1 \land x_2$
- Can use similar tricks for other gate types
- Why do I give the ideal functionality only, not the actual protocol?
Is Secure OT Enough?

- We saw an oblivious transfer protocol which is secure even if parties are malicious
  - Chooser commits to his input and proves in zero knowledge that he performed his computations correctly
- Suppose each gate of the circuit is evaluated using an oblivious transfer protocol which is secure with malicious parties...
- Do we obtain a protocol for securely computing any function with malicious parties?
Oops!

When output of Gate 1 is used as input into Gate 2, both parties must use the same pair of bits that was obtained from evaluating Gate 1 (why?)
Security with Malicious Parties

- Details omitted (this is less than a sketch!)
- Intuition: every party commits to its inputs and proves in ZK that it did its computation correctly
  - This includes proving that output of one OT protocol is consistent with input of another OT protocol
- Main advantage: secure building blocks compose into secure protocols
  - Each building block (commitment, OT, etc.) is indistinguishable from its ideal functionality (IF)
  - IFs can be composed without compromising security
  - Can build a secure protocol from secure primitives
Issues

- **Parallel composition** of zero-knowledge proofs does not work in general
  - One proof is zero-knowledge, but two proofs executed concurrently are no longer zero-knowledge

- **How can cryptographic primitives be formulated as secure multi-party problems?**
  - Many protocols use encryption, digital signatures, etc.

- **Adaptive corruptions**: adversary may corrupt an honest party in the middle of protocol execution
  - Proofs with “rewinding” don’t work any more (why?)
ZKPK of Discrete Log

Suppose we have some ZK protocol $\Pi$ for proving knowledge of discrete log.

Here is another protocol $\Pi'$.

$\Pi'$ is a sound zero-knowledge protocol of discrete-log knowledge (why?)
Concurrent Composition

- V runs two instances of protocol $\Pi'$ in parallel

- Run $\Pi$ with V as prover
  - 1 (i.e., “I know w”)
  - Replays $P_2$’s messages to $P_1$
  - and $P_1$’s messages to $P_2$
  - Convinced!

- Run $\Pi$ with $P_2$ as prover
  - 0 (i.e., “I don’t know w”)

Effectively, $P_2$ who knows $w$ convinced $P_1$, but $P_1$ thinks he was convinced by $V$.

Cheating verifier learns $w$!

This protocol is clearly NOT zero-knowledge (why?)
SMC In This Course

This is how much we cover in this class
To Learn More