Bar-Ilan University
Dept. of Computer Science

Session 6: Oblivious Transfer

Benny Pinkas
Bar-Ilan University

- Most of this talk is based on Ch. 7, “Efficient Secure Two-Party Protocols”, Hazay and Lindell, 2010.

1-out-of-2 Oblivious Transfer

Bar-Ilan University
Dept. of Computer Science

- **Two players: sender and receiver.**
  - Sender has two inputs, x_0, x_1.
  - Receiver has an input $j \in \{0,1\}$.
- **Output:**
  - Receiver learns x_j and nothing else.
  - Sender learns nothing about j.

- Depending on the OT variant, the inputs x_0,x_1 could be strings or bits.

- We examine the malicious setting.
- We consider the standard model and aim to get fully simulatable protocols
- More efficient protocols are possible if these requirements are relaxed
  - Random oracle model
  - Protocols which are not proved to be secure in the sense of full simulatability.

Why study OT?

Bar-Ilan University
Dept. of Computer Science

- Oblivious transfer is one of the basic primitives of secure computation
  - "Founding cryptography on oblivious transfer", J. Kilian, 1988.
  - OT alone, without any complexity-theoretic assumptions, can be used to construct non-interactive zero-knowledge proofs of statements in NP.
- The overhead of OT is often the bottleneck of the entire secure protocol.

Feasibility of constructing OT

Bar-Ilan University
Dept. of Computer Science

- There is no OT protocol which provides unconditional security for both parties.
  - Namely, with information theoretic security which does not depend on any computation assumption.

- We show this by showing that there is no AND protocol which provides unconditional security for both parties.

Computing "AND" privately

Bar-Ilan University
Dept. of Computer Science

- P_1 and P_2, have binary inputs a and b.
- They wish to securely compute a AND b.
  - Suppose that P_1's input is a=0, and he learns that (a AND b) = 0. Then he must not learn whether P_2's input is 0 or 1.

- Applications?
  - dating

Computing "AND" Privately using OT

Bar-Ilan University
Dept. of Computer Science

- P_1 is the sender, with inputs $x_0=0$, $x_1=a$.
- P_2 is the receiver, with input $j=b$.
  - They run an OT protocol, and output its output.
  - The output is $(1-j)\cdot x_0+j\cdot x_1 = (1-b)\cdot 0+b\cdot a = a\cdot b$.
- Privacy (semi-honest, hand-waving):
  - If b=0 then P_2 always learns 0, and therefore can be easily simulated.
  - If b=1 then the result obtained in the OT is equal to P_1's input a, but it is also equal to $a\cdot b$ which is the legitimate output of P_2.
  - Simulation is therefore easy.

Impossibility of achieving OT with unconditional security

Bar-Ilan University
Dept. of Computer Science

- Suppose that there is an AND protocol (between P_1 and P_2, with inputs a and b) with unconditional security.
  - Such a protocol could be constructed from an OT which has unconditional security.
- Let T be a transcript of all messages sent in the protocol.
- The parties use random inputs R_1 and R_2.
  - Given these inputs the transcript T is a deterministic function.

Impossibility of achieving OT with unconditional security

Bar-Ilan University
Dept. of Computer Science

- In a certain execution with P_1's input a=0, the protocol has transcript T and output "0".
  - If b=0, then P_2 must not learn P_1's input.
  - Therefore ∃ an R'_1 s.t. if P_1 has inputs a=1 and R'_1, the protocol would have produced the same transcript T.
  - If b=1, then output is 0. Therefore there is no R''_1 s.t. the protocol has transcript T for a P_1 input of a=1.
- P_1 can therefore
  - search over all possible values for R_1 and check if running them with input a=1 results in transcript T. If there is such an R_1 then it concludes that b=0.

Oblivious transfer
Privacy definition

- **We prefer to use protocols which are fully secure**
  - Can be easily compostable in higher level protocols
  - Especially important for oblivious transfer
- **Defining privacy only is difficult**
  - No correctness and independence of inputs.
  - E.g., do not ensure that the protocol implements the OT functionality.
  - Composition is not guaranteed.

- For oblivious transfer, we know how to define privacy only, for two-round protocols.

Privacy definition

Bar-Ilan University
Dept. of Computer Science

- **Why do 2 rounds help?**
  - Receiver sends one message – commits to its choice
  - Sender replies with one message
- **Privacy definition for a malicious sender**
  - Just need to prove indistinguishability of receiver's first message when **b=0** and when **b=1**
  - Namely, for any values of the sender's inputs x_0,x_1, the sender cannot distinguish between the case that the receiver's input is 0 and the case that it is 1.
  - This can be extended to many messages

Privacy definition

Bar-Ilan University
Dept. of Computer Science

- **Privacy definition for a malicious receiver**
  - More intricate, since the receiver obtains an output.
  - First message is generated before seeing anything. We would like that this message essentially commits the receiver to learning a specific message.
  - The definition requires that for every first message sent by the receiver, there exists a bit $\mathbf{b'}$ such that receiver learns nothing about $\mathbf{x_{b'}}$.

Preliminaries – The Decisional Diffie Hellman (DDH) assumption

Bar-Ilan University
Dept. of Computer Science

- The **Decisional Diffie-Hellman assumption (DDH)**, is that the following problem is hard:
  - The input to the problem contains
  - a group G of order q, and a generator g of G
  - a pair of tuples in random order,
    - (g^a, g^b, g^c) where $a,b,c \in_R [1,q]$
    - (g^a, g^b, g^{ab}) where $a,b \in_R [1,q]$
  - The task is to decide which of the two tuples is (g^a, g^b, g^{ab}).

OT satisfying the privacy only definition [NP]

Bar-Ilan University
Dept. of Computer Science

- **Input**: sender – x_0,x_1. receiver – $j\in\{0,1\}$.
- **Setting**: Group G of prime order q. Generator g.
- **Receiver**
  - chooses random $a,b,c_{1-j}\in[1,q]$, and defines $c_j=ab$ (mod q).
  - Sends to the sender the message $(g^a, g^b, g^{c0}, g^{c1})$.
- **The sender**
  - Checks that $g^{c0}\neq g^{c1}$. Chooses random $u_0,v_0,u_1,v_1\in[1,q]$.
  - Defines $w_0=(g^a)^{u0}g^{v0}$. Encrypts x_0 with the key $k_0=(g^{c0})^{u0}(g^b)^{v0}$.
  - Defines $w_1=(g^a)^{u1}g^{v1}$. Encrypts x_1 with the key $k1=(g^{c1})^{u1}(g^b)^{v1}$.
  - Sends w_0, w_1 and encs with k_0,k_1 to receiver.
- Receiver computes $(w_j)^b$ which is the key k_j with which x_j can be decrypted.

Properties

Bar-Ilan University
Dept. of Computer Science

- **Correctness**
  - Suppose $j=0$. R sends (g^a, g^b, g^{ab}, g^c).
  - S defines $w_0=(g^a)^{u0}g^{v0}$.
  - S encrypts x_0 with $k_0=(g^{ab})^{u0}(g^b)^{v0}$.
    - Note that encryption key is equal to $(w_0)^b$.
  - R computes $k_0=(w_0)^b$ and uses it for decryption.

- **Overhead:**
  - R computes 5 exponentiations.
  - S computes 8 exponentiations.

Privacy – malicious sender

Bar-Ilan University
Dept. of Computer Science

- **Receiver's security**
  - Based on the DDH assumption
  - Must show that sender's view is indistinguishable regardless of receiver's input.
    - Sender receives either (g^a, g^b, g^{ab}, g^c) or (g^a, g^b, g^c, g^{ab}).
    - Suppose that it can distinguish between the two cases.
    - We can construct a distinguisher for the DDH problem, which distinguishes between (g^a,g^b,g^{ab}) and (g^a,g^b,g^c):
    - The distinguisher receives (g^a,g^b,X) and (g^a,g^b,Y), and sends (g^a,g^b,X,Y) to S.

Privacy – malicious receiver

Bar-Ilan University
Dept. of Computer Science

- The security of the server is unconditional.
  - Does not depend on any cryptographic assumption.
- Suppose that $j=0$.
- Regarding x_1, server sees
  - $w_1=(g^a)^{u1}g^{v1}$.
  - x_1 encrypted with the key $k_1=(g^c)^{u1}(g^b)^{v1}$.
  - The values u_1,v_1 were chosen at random, and $ab\neq c_1$.
  - **Claim:** (w_1,k_1) are uniformly distributed.
  - Therefore message (w_1,k_1) sent by S about x_1 can be easily simulated.

Privacy – malicious receiver

Bar-Ilan University
Dept. of Computer Science

- Proof of claim:
  - $w_1=(g^a)^{u1}g^{v1}= g^{a\cdot u1+v1}$.
  - $k_1=(g^c)^{u1}(g^b)^{v1}= g^{c\cdot u1+b\cdot v1}= (g^{(c/b)\cdot u1+v1})^b$.
  - Define $F(x) = u_1 \cdot x + v_1$. F(x) is pair-wise independent:
    - $\forall x,y,s,t$ Prob(F(x)=s & F(y)=t) = $1/|G|^2$
  - $w_1=g^{F(a)}$.
  - $k_1= (g^{F(c/b)})^b$.
  - $c \neq ab$ and therefore F(a) and F(c/b) are uniformly distributed.
  - $\Rightarrow (w_1,k_1)$ are uniformly distributed.

One-sided simulation

Bar-Ilan University
Dept. of Computer Science

- The sender receives no output
  - Therefore we keep the previous requirement – that it cannot distinguish between different inputs of the receiver
- We require in addition the existence of a simulator that can fully simulate the receiver's view.
- Does not solve all problems:
  e.g., sender's input can depend on the first message it receives.

OT with one-sided simulation

Bar-Ilan University
Dept. of Computer Science

- A simple modification to the previous protocol:
  - When the receiver sends its message (g^a,g^b,g^{c0},g^{c1}), it adds a zero-knowledge proof of knowledge of a.
  - Namely, proves the relation $R_{DL}=\{ ((G,q,g,h,),a) \mid h=g^a\}$
  - Intuitively, this shows that the receiver "knows" which of x_0,x_1 it wishes to learn in the protocol.

OT with one-sided simulation

Bar-Ilan University
Dept. of Computer Science

- Add a ZK POK of discrete log.
  - 6 rounds of communication.
  - Additional 9 exponentiations.

- The idea behind the security proof:
  - Extract a from the ZK POK.
  - Find which of g^{c0}, g^{c1} is equal to $(g^b)^a$.
  - Define the input j of the receiver accordingly.
  - Send j to the TTP.
  - Learn x_j, and simulate.

OT with one-sided simulation
Security proof

Bar-Ilan University
Dept. of Computer Science

- The case of a malicious sender is as before.
- Simulator for a malicious receiver R^*:
  - Receive from R^* its first message (g^a,g^b,g^{c0},g^{c1}), and the ZK POK of discrete log of g^a.
  - Run the POK's simulator and extract R^*'s input a.
  - If $g^{c0}=(g^a)^b$ then set $j=0$. Otherwise set $j=1$.
  - Send j to the TTP and receive x_j.
  - Operate as S does on the message (g^a,g^b,g^{c0},g^{c1}). Return encryptions of the x_j received from the TTP, and of $x_{1-j}=1$.

OT with one-sided simulation Security proof

Bar-Ilan University
Dept. of Computer Science

- Must show that R^*'s view is indistinguishable from its view in the real execution.
  - Until the last message, R^* sees exactly the same messages as in a real execution.
  - In the last message, the only difference is that the simulator encrypted the value $x_{1-j}=1$ instead of the actual value of x_{1-j}.
  - But we proved before that for the receiver, the keys with which x_{1-j} is encrypted are uniformly distributed.
  - Therefore it cannot distinguish…

OT with full simulatability

- Why doesn't the previous protocol suffice?
  - For full simulatability, need to be able to extract the input of a malicious sender and send it to the TTP.
  - The sender receives a message (g^a,g^b,g^{c0},g^{c1}).
  - It checks that $g^{c0} \neq g^{c1}$, and therefore only one of c_0,c_1 is equal to ab. For the other c value, the message the sender sends is uniformly distributed, and the corresponding input cannot be extracted.
  - We can rewind S and send it another message (g^a,g^b,g^{c0},g^{c1}). But its answer might be different than before, so we might extract now a different message.

OT with full simulatability [HL]

Bar-Ilan University
Dept. of Computer Science

- An idea overcoming the previous problem:
  - Receiver sends a longer message $(g^a, g^b, g^{c0}, g^{c1})$, $(g^{a'}, g^{b'}, g^{c'0}, g^{c'1})$, and proves that either $c_0=ab$ or $c'_1=a'b'$, but not both.
  - Therefore receiver can only learn one message,
  - But in the simulation we can cheat in the proof and send a message which enables to learn both inputs of sender.
  - Since this is a *single* message for both inputs, we do not care if sender's behavior depends on the message it sees.

OT with full simulatability Basic ideas

Bar-Ilan University
Dept. of Computer Science

- R sends a single message (h_0,h_1,d,b_0,b_1)
- $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r+j}$, $b_1=g^{a1\cdot r+j}$
  - Recall, $j\in\{0,1\}$.
  - If $j=0$ then (h_0, d, b_0) is a DDH tuple.
  - If $j=1$ then $(h_1, d, b_1/g)$ is a DDH tuple.

  - R also needs to prove that it can't be that both (h_0, d, b_0) and $(h_1, d, b_1/g)$ are DDH tuples.

OT with full simulatability
Basic ideas

Bar-Ilan University
Dept. of Computer Science

- R sends a single message (h_0,h_1,d,b_0,b_1)
- $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r+j}$, $b_1=g^{a1\cdot r+j}$
- R proves that $(h_0/h_1,d,b_0/b_1)$ is a DDH tuple.
- Therefore cannot be that $b_0=g^{a0\cdot r}$ and $b_1=g^{a1\cdot r+1}$,
- Namely cannot be that both (h_0,d,b_0) and $(h_1,d,b_1/g)$ are DDH tuples.

OT with full simulatability Basic ideas

Bar-Ilan University
Dept. of Computer Science

- R sends a single message (h_0,h_1,d,b_0,b_1)
- $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r+j}$, $b_1=g^{a1\cdot r+j}$
  - When j=0 then (h_0,d,b_0) is a DDH tuple, but $(h_1,d,b_1/g)$ isn't.
  - When j=1 then $(h_1,d,b_1/g)$ is a DDH tuple, but (h_0,d,b_0) isn't.
- Use (h_0,d,b_0) to encrypt x_0, and $(h_1,d,b_1/g)$ to encrypt x_1.
- In the simulation, cheat in the POK s.t. (h_0,d,b_0) and $(h_1,d,b_1/g)$ are both DDH tuples.

The protocol

Bar-Ilan University
Dept. of Computer Science

- R chooses random $a_0,a_1,r \in [1,q]$ and sends the message (h_0,h_1,d,b_0,b_1)
  - $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0 \cdot r+j}$, $b_1=g^{a1 \cdot r+j}$
- R proves, using a ZK POK, that $(h_0/h_1,d,b_0/b_1)$ is a DDH tuple.
- S chooses random $u_0,v_0,u_1,v_1 \in [1,q]$, and sends
  - $w_0=d^{u0}g^{v0}$, and encrypts x_0 with $k_0=(b_0)^{u0}(h_0)^{v0}$.
  - $w_1=d^{u1}g^{v1}$, and encrypts x_1 with $k_1=(b_1/g)^{u1}(h_1)^{v1}$.
- R decrypts with $(w_j)^{aj}$

Correctness

Bar-Ilan University
Dept. of Computer Science

- R sends the message (h_0,h_1,d,b_0,b_1)
- $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r+j}$, $b_1=g^{a1\cdot r+j}$
- S chooses random $u_0,v_0,u_1,v_1\in[1,q]$, and sends
  - $w_0=d^{u0}g^{v0}$, and encrypts x_0 with $k_0=(b_0)^{u0}(h_0)^{v0}$.
  - $w_1=d^{u1}g^{v1}$, and encrypts x_1 with $k_1=(b_1/g)^{u1}(h_1)^{v1}$.
- R decrypts with $(w_j)^{aj}$
- When $j=0$, $(w_0)^{a0} = (d^{u0}g^{v0})^{a0} = (g^{r\cdot u0+v0})^{a0} = (g^{r\cdot a0})^{u0}(g^{a0})^{v0} = (b_0)^{u0}(h_0)^{v0} = k_0$
- When $j=1$, $(w_1)^{a1} = (d^{u1}g^{v1})^{a1} = (g^{r\cdot a1})^{u1}(g^{a1})^{v1} = (b_1/g)^{u1}(h_1)^{v1} = k_1$

Overhead

Bar-Ilan University
Dept. of Computer Science

- 6 rounds of communication, including ZK POK
- Sender computes 15 exponentiations
- Receiver computes 11 exponentiations

Security – malicious sender

Bar-Ilan University
Dept. of Computer Science

- Simulator
  - Computes $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r}$, $b_1=g^{a1\cdot r+1}$
    - Compared to $b_0=g^{a0\cdot r+j}$, $b_1=g^{a1\cdot r+j}$ in real execution
  - Sends to sender
  - Cheats in ZK POK to simulate a proof that the first message is well formed
  - Receives w_0,w_1 and two encryptions from sender
  - Computes $k_0=(w_0)^{a0}$ and $k_1= (w_1)^{a1}$
  - Decrypts encryptions using k_0,k_1
  - Sends results to TTP

Security – malicious sender

Bar-Ilan University
Dept. of Computer Science

- The only difference in the messages that sender sees, between real and simulated executions, is the first message
  - Real, j=0: $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r}$, $b_1=g^{a1\cdot r}$
    - (h_0,d,b_0) and (h_1,d,b_1) are DDH tuples
  - Real, j=1: $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r+1}$, $b_1=g^{a1\cdot r+1}$
    - $(h_0,d,b_0/g)$ and $(h_1,d,b_1/g)$ are DDH tuples
  - Simulated: $h_0=g^{a0}$, $h_1=g^{a1}$, $d=g^r$, $b_0=g^{a0\cdot r}$, $b_1=g^{a1\cdot r+1}$
    - (h_0,d,b_0) and $(h_1,d,b_1/g)$ are DDH tuples
- Can show that if server can distinguish, it can break DDH

Security – malicious receiver

Bar-Ilan University
Dept. of Computer Science

- Simulator
  - Receives from receiver (h_0,h_1,d,b_0,b_1)
  - Extracts from ZK POK the input r s.t. $d=g^r$
  - If $b_0=(h_0)^r$ then sets $j=0$. Otherwise sets $j=1$.
  - Sends j to TTP and receives x_j.
  - Computes w_0,k_0,w_1,k_1 as the sender would do.
  - Uses these values to encrypt the x_j received from TTP, and $x_{j-1}=1$.
  - Sends encryptions to receiver.

Security – malicious receiver

Tradition of Excellence
Bar-Ilan University
Dept. of Computer Science

- Proof:
  - Until the last message, the receiver's view is as in the real protocol. In the last message, the encryption of x_{1-j} is replaced with an encryption of 1.
  - If $b_0=(h_0)^r$ then $j=0$ and x_1 is replaced with 1.
  - From the ZK POK is follows that $b_1=(h_1)^r$, therefore
  - $w_1=d^{u1}g^{v1}=g^{r\cdot u1+v1}$, $k_1=(b_1/g)^{u1}(h_1)^{v1}=(h_1)^{r\cdot u1+v1}/g^{u1}$
  - Need to show that these values are uniformly distributed (and therefore receiver cannot decrypt)

Security – malicious receiver

Bar-Ilan University
Dept. of Computer Science

- $w_1 = d^{u1} g^{v1} = g^{r \cdot u1 + v1}$,
  $k_1 = (b_1/g)^{u1} (h_1)^{v1} = (h_1)^{r \cdot u1 + v1} / g^{u1}$
- Define $F(x) = u1 \cdot x + v1$.
- F(X) is pair-wise independent, since u_1, v_1 are uniformly distributed.
- $w_1 = g^{F(r)}$
- $k_1 = (g^{a1})^{F(r)} / g^{u1} = (g^{a1})^{F(r) - u1/a1} = (g^{a1})^{F(r - 1/a1)}$
- Therefore (w_1, k_1) are uniformly distributed

Conclusions

Bar-Ilan University
Dept. of Computer Science

- Fully simulatable OT (against malicious parties) can be efficiently implemented
- Batch OT – performing many Ots
  - Can perform a single ZK POK
  - Overhead is reduced to 14 exponentiations per OT + 23 for the initialization
- Peikert–Vaikuntanathan–Waters
  - Similar ideas to the OT protocol we presented
  - Batch OT overhead: 11 exponentiations per OT + 15 for the initialization

Conclusions

Bar-Ilan University
Dept. of Computer Science

- **We considered the standard model, and protocols which can be proved to be secure in the sense of full simulatability**
  - More efficient protocols are known if these requirements are relaxed
- **Extending OT**
  - [Beaver], [Ishai,kilian,Nissim,Petrank]
  - Precompute k (e.g. 128) OTs which can then be used to perform an arbitrary # of OTs
  - No proof if the sense we want here