



Session 7: Two-Party Secure Computation for Malicious Adversaries

Yehuda Lindell
Bar-Ilan University

This Session



Bar-Ilan University
Dept. of Computer Science

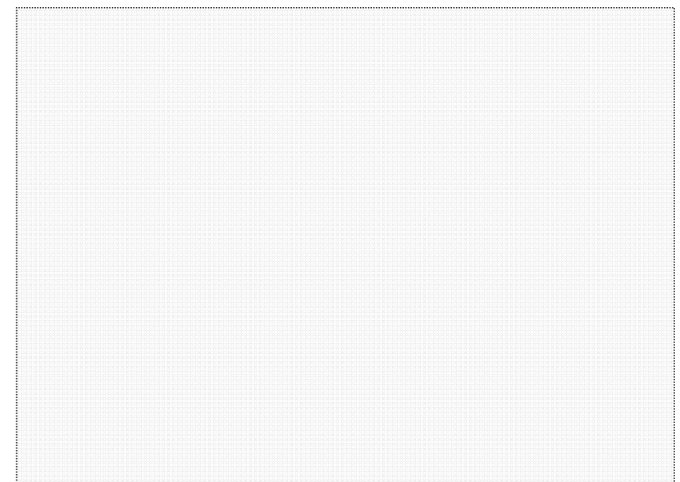
- ▶ **Constructing efficient secure two-party protocols for malicious adversaries**
 - In principle, this problem is solved by GMW but is not efficient
 - Important: there is no honest majority here and so BGW techniques don't work
- ▶ **Session outline**
 - Survey known approaches to the problem
 - Focus in detail on the cut-and-choose approach
 - Personal bias 😊

Yao's Protocol and Malicious



Bar-Ilan University
Dept. of Computer Science

- ▶ **Malicious P_1 in Yao's Protocol**
 - A malicious P_1 can construct an incorrect circuit
 - This can harm correctness, privacy, and independence of inputs
 - A malicious P_1 can carry out a “selective input attack”
 - P_1 can input an incorrect key for the 0-value on the 1st bit of P_2 's input
 - This causes P_2 to abort if $y_1=0$ and to successfully compute output if $y_1=1$
 - In the ideal world, P_1 cannot make the abort depend on P_2 's input



Yao's Protocol and Malicious



Bar-Ilan University
Dept. of Computer Science

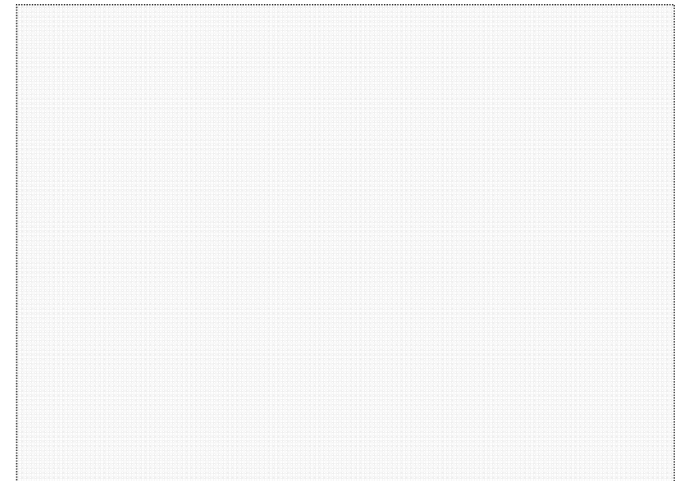
- ▶ Aim: force the circuit constructor to behave honestly
- ▶ This can be achieved using general ZK proofs, but this won't be efficient
- ▶ What other ways can this be done?
 - It turns out that there are many other ways...

Approaches



Bar-Ilan University
Dept. of Computer Science

1. Prove correctness of circuit construction using zero-knowledge
2. LEGO: prove correctness of gate construction and then solder gates together
3. Virtual MPC
4. From multiplication tuples to arithmetic circuit construction
5. Cut-and-choose to prove correctness of Yao circuits

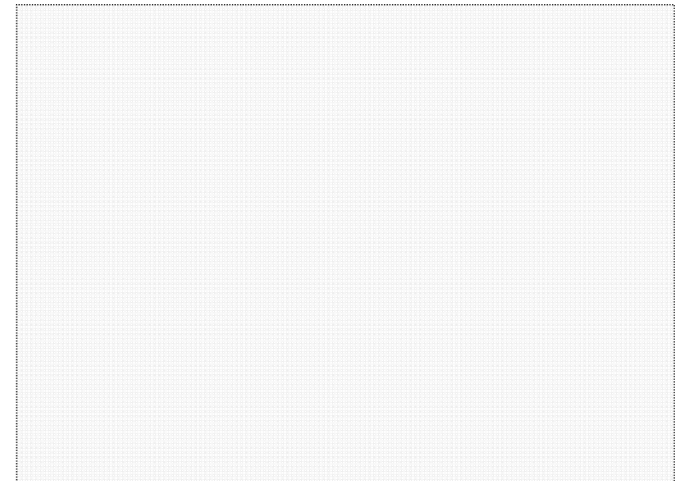


Boolean vs Arithmetic



Bar-Ilan University
Dept. of Computer Science

- ▶ Boolean circuits: AND/OR/XOR etc.
- ▶ Arithmetic circuits: ADD/MULT over some defined finite field
- ▶ What is better?
 - It depends on the application
 - AES:
 - 33,000 gates in a Boolean circuit
 - 2,400 gates over $GF[2^8]$
 - Branching is better in Boolean...



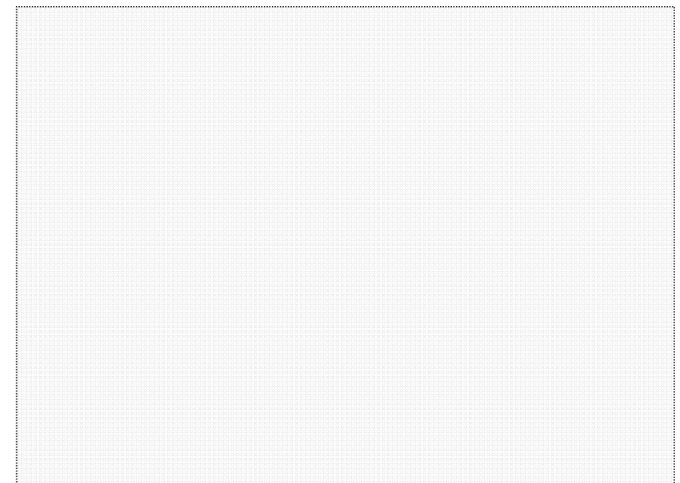
ZK Proving (Boolean Circuits)



Bar-Ilan University
Dept. of Computer Science

Jarecki-Shmatikov (Eurocrypt 2007)

- ▶ **Encrypt gates using asymmetric encryption with algebraic structure**
 - Use Camenisch-Shoup based on DCR (N-residuosity); two exponentiations mod N^2
- ▶ **Use structure to prove in zero knowledge that circuit is correctly constructed**
 - Used correct keys
 - Gate has correct structure
 - And so on...



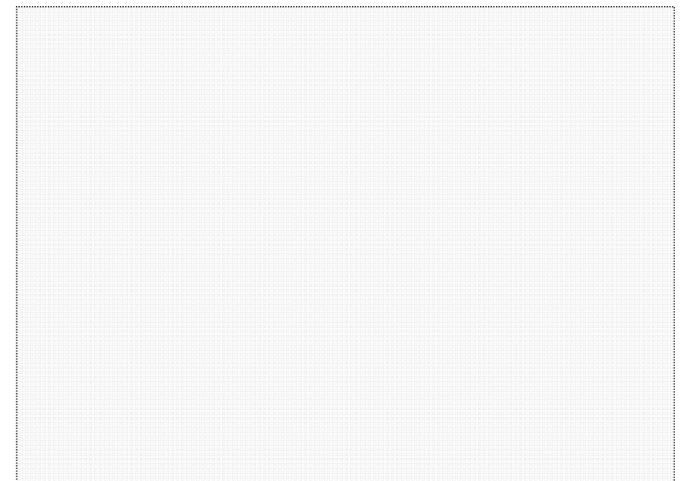
ZK Proving



Bar-Ilan University
Dept. of Computer Science

- ▶ **$O(1)$ exponentiations per gate**
 - What is $O(1)$? Here: 720
 - Also, these are N^2 exponentiations which are much more expensive than DH exponentiations which can be run in an Elliptic curve group

- ▶ **Optimizing the approach**
 - More efficient ZK protocols
 - Challenge: how to build the gates so that they yield efficient proofs
 - Batching of ZK protocols



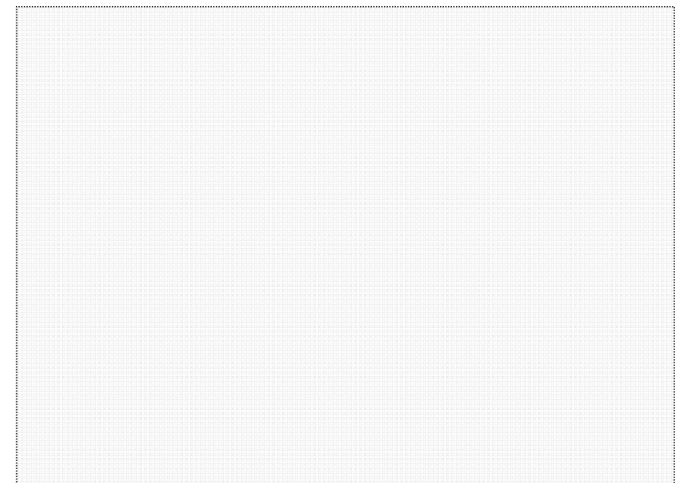
LEGO (Boolean Circuits)



Bar-Ilan University
Dept. of Computer Science

Nielsen–Orlandi (TCC 2009)

- ▶ Generate many encrypted gates using homomorphic commitments
- ▶ Open half of the gates to check that they are correctly formed
 - This guarantees that the majority of the remaining gates are correct
- ▶ Combine the remaining gates in a fault tolerant circuit
 - Use homomorphic property to “solder” the gates
- ▶ Compute the circuit



LEGO Efficiency



Bar-Ilan University
Dept. of Computer Science

- ▶ **Size of fault tolerant circuit $O(s |C|/\log |C|)$**
 - Statistical security parameter s
 - Error is 2^{-s} , so can set $s=40$
- ▶ **Number of exponentiations per gate is 32**
 - Number of exponentiations is $1280|C|/\log|C|$
 - Exponentiations are regular Diffie–Hellman

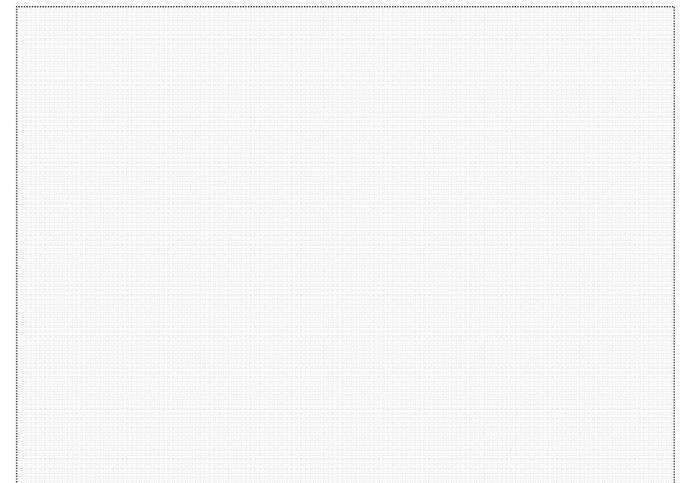
Virtual MPC (Arithmetic Circuits)



Bar-Ilan University
Dept. of Computer Science

Ishai-Prabhakaran-Sahai (Crypto 2008)

- ▶ Parties emulate a multiparty protocol with honest majority
 - Such protocols are much more efficient for arithmetic circuits
- ▶ Parties run 2-party protocols to simulate every step of the parties in the honest majority protocol
 - The parties use semi-honest protocols and “watchlists” to catch cheating



Multiplication Tuples

(Arithmetic Circuits)

- ▶ Damgard–Orlandi (Crypto 2010)
- ▶ The protocol
 - Share the inputs
 - Addition: locally add shares (like BGW)
 - Multiplication: as in BGW, this is the hard part
- ▶ Based on an idea by Beaver from 1991

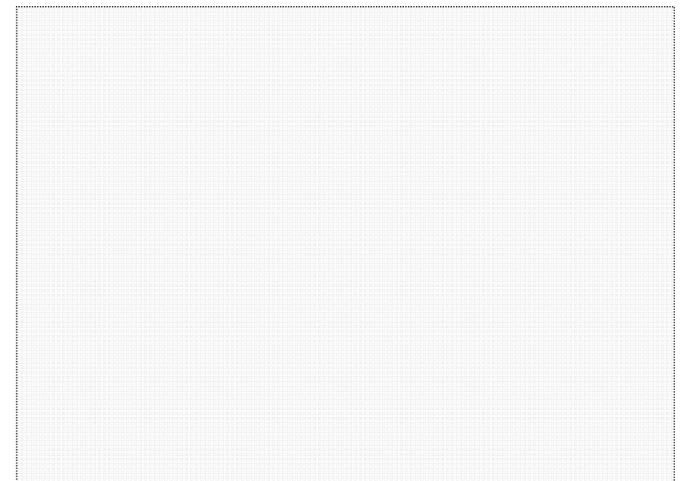
Multiplication Tuples



Bar-Ilan University
Dept. of Computer Science

► Setup

- Assume that the parties have many tuples of the form $[\text{Com}(x), \text{Com}(y), \text{Com}(z)]$ where $x=y \cdot z$ together with additive shares (x_1, x_2) , (y_1, y_2) and (z_1, z_2) of (x, y, z) , respectively
- In addition, **Com** is homomorphic
 - Can compute shares of $\text{Com}(x+y)$ given shares of $\text{Com}(x)$ and $\text{Com}(y)$
 - Can compute shares of $\text{Com}(a \cdot x)$ given shares of a and shares of $\text{Com}(x)$



Multiplication Using Tuples



Bar-Ilan University
Dept. of Computer Science

► Multiplication

- Wire 1: P_1 and P_2 have additive shares u_1, u_2 of u
- Wire 2: P_1 and P_2 have additive shares v_1, v_2 of v
- Aim: compute shares of $w = u \cdot v$; i.e. compute w_1, w_2 such that $w_1 + w_2 = (u_1 + u_2) \cdot (v_1 + v_2)$

Multiplication Using Tuples

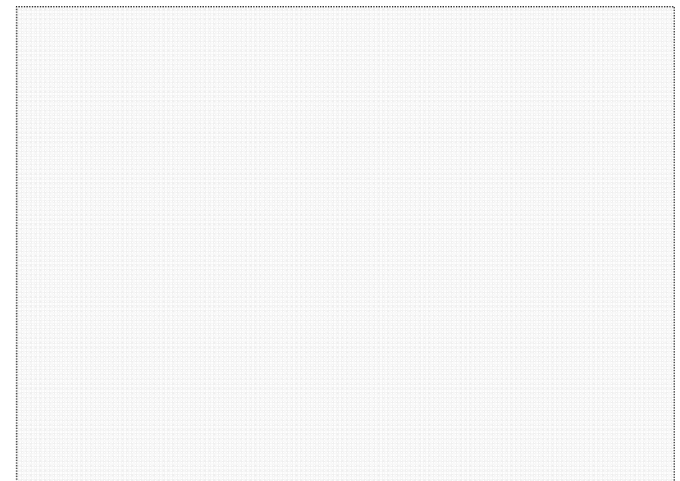
► Computation:

- Parties have additive shares of $\text{Com}(x)$, $\text{Com}(y)$, $\text{Com}(z)$ where $x=y \cdot z$
- Compute shares of $\text{Com}(u-y)$, and open; denote u'
- Compute shares of $\text{Com}(v-z)$, and open; denote v'
- Compute shares of

$$\text{Com}(u' \cdot v) + \text{Com}(v' \cdot u) + \text{Com}(x) - u' \cdot v'$$

- What does it equal? Shares of:

$$\begin{aligned} & (u-y) \cdot v + (v-z) \cdot u + y \cdot z - (u-y)(v-z) \\ &= uv - yv + vu - zu + yz - uv + zu + yv - yz \\ &= u \cdot v \end{aligned}$$

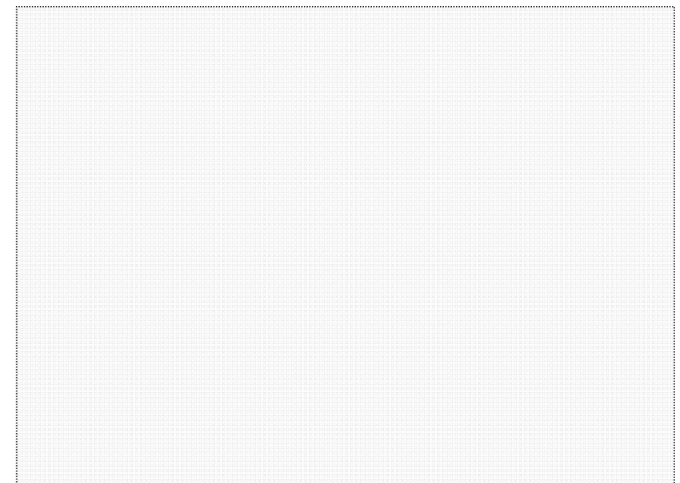


The Protocol



Bar-Ilan University
Dept. of Computer Science

- ▶ Run a specific two-party computation to generate multiplication tuples
 - This uses a special-purpose protocol, secure for malicious adversaries
- ▶ Share the inputs using the homomorphic commitments
- ▶ Locally add shares for addition
- ▶ Use tuples as shown for multiplication



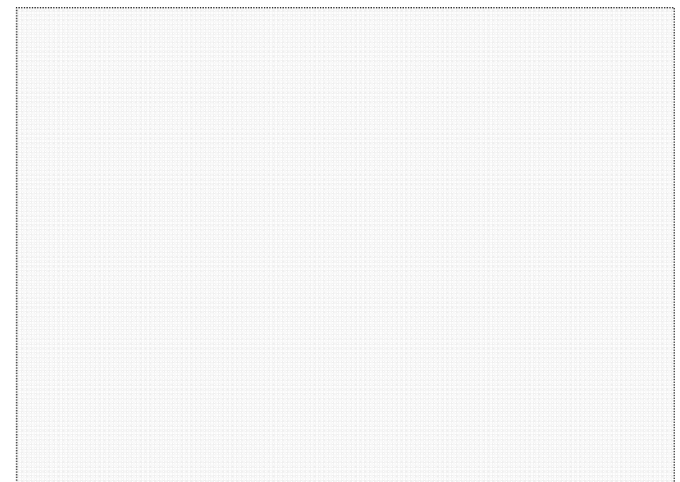
Cut-and-Choose (Boolean Circuits)



Bar-Ilan University
Dept. of Computer Science

Lindell–Pinkas (Eurocrypt 2007, TCC 2011)

- ▶ The basic idea – prove that the Yao circuit is correctly constructed as follows:
 - P_1 constructs s garbled circuits and sends them to P_2
 - P_2 chooses a random subset of $\frac{1}{2}$ and sends it to P_1
 - P_1 “opens” these circuits by sending all of the garbled keys
 - P_2 checks that the circuits are correctly constructed

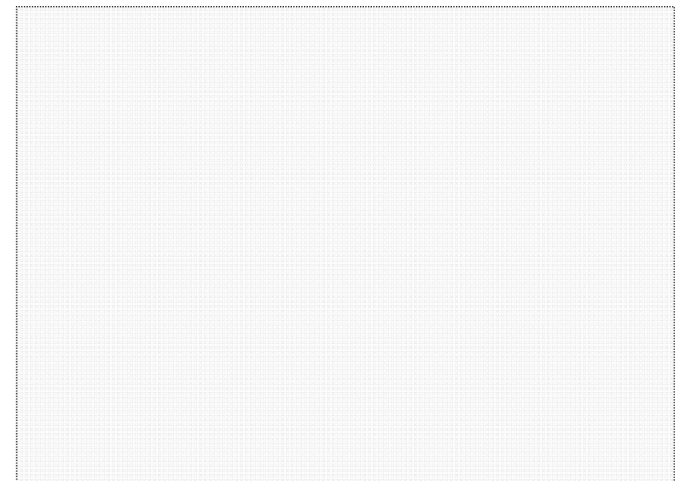


Cut-and-Choose



Bar-Ilan University
Dept. of Computer Science

- ▶ **What is guaranteed?**
 - A majority of the remaining circuits are correctly constructed
- ▶ **The rest of the protocol**
 - The parties compute all of the remaining garbled circuits
 - It is not enough to compute one because it is only guaranteed that the majority are fine

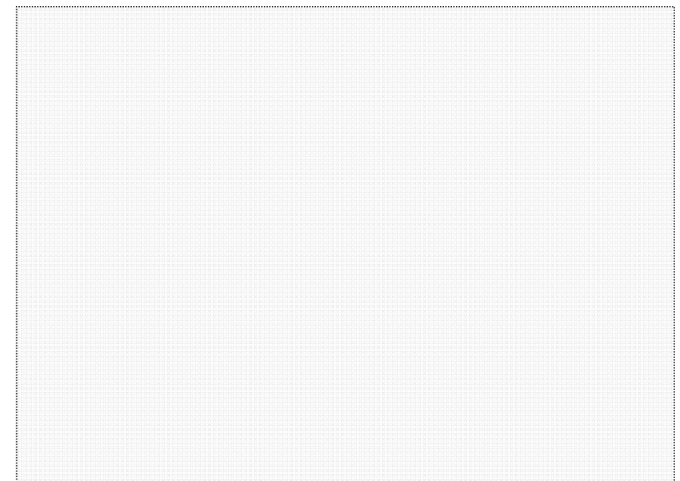


Difficulties and Attacks



Bar-Ilan University
Dept. of Computer Science

- ▶ What does P_2 do if it obtains different outputs?
 - Option 1: it detects P_1 cheating and so aborts
 - Attack: P_1 can use this to cheat:
 - P_1 constructs one circuit that outputs garbage if the first bit of P_2 's input equals 0 (otherwise, computes f)
 - If P_2 aborts, P_1 knows that P_2 's 1st input bit equals 0
 - Option 2: output majority value
 - This is the correct option; sometimes need to be quiet even when cheating is detected!

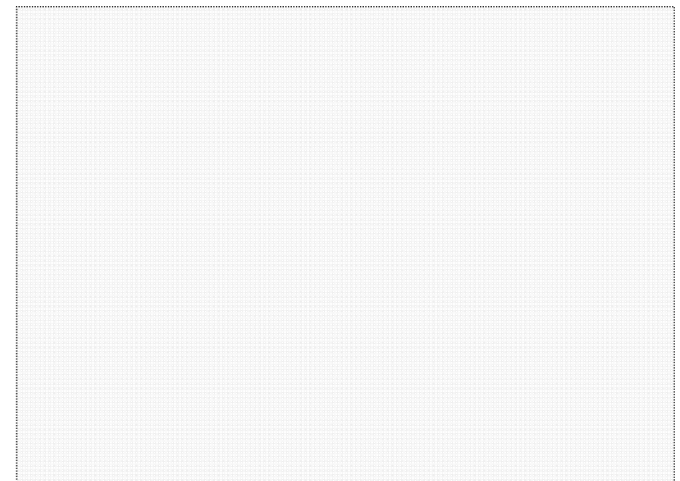


Difficulties and Attacks



Bar-Ilan University
Dept. of Computer Science

- ▶ It may be possible for P_1 to construct a garbled circuit G with 2 different sets of garbled values/keys K, K' such that
 - The keys in K decrypt G to the correct circuit C
 - The keys in K' decrypt G to an incorrect circuit C'
- ▶ This can be solved by having P_1 also commit to the keys



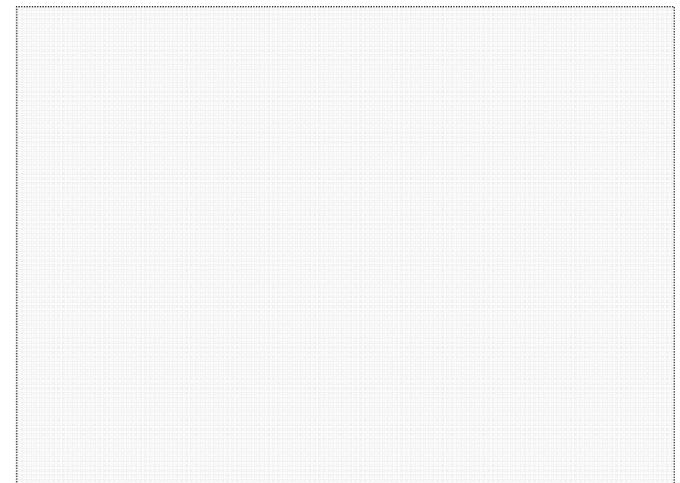
Difficulties and Attacks



Bar-Ilan University
Dept. of Computer Science

► Input consistency

- P_2 may use different inputs y_1, y_2, \dots in different circuits, in order to get $f(x, y_1), f(x, y_2), \dots$
- P_1 may use different inputs x_1, x_2, \dots in different circuits in order to get $f(x_1, y), f(x_2, y), \dots$
 - But won't this be detected by P_2 who gets the output?
 - Not necessarily; it depends on the function

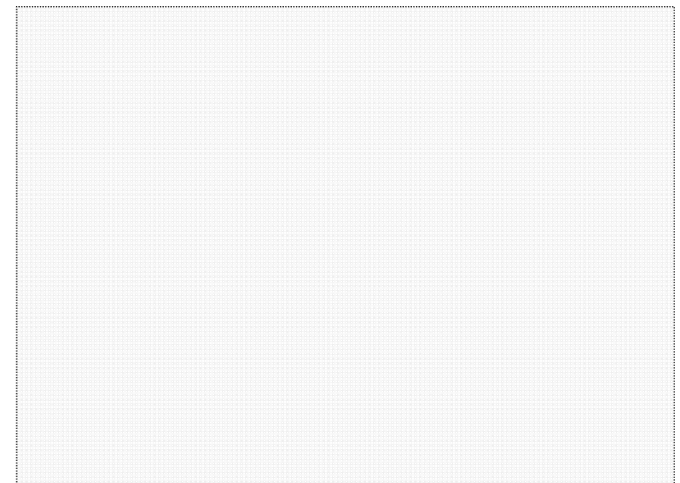


Solutions – Protocol 2007



Bar-Ilan University
Dept. of Computer Science

- ▶ Cut-and-choose on the circuit does not prevent a selective-input attack
- ▶ Preventing selective-input attacks
 - Split each input bit y of P_2 into s random bits y_1, \dots, y_s such that $y_1 \oplus \dots \oplus y_s = y$
 - Change the circuit to first compute the XOR of these bits and then the function
- ▶ Why does this help?
 - Each input bit is now random (the correlation between y_1, \dots, y_s and the actual bit y can be guessed w.p. 2^{-s})
 - Thus, any attack on the input bits is not correlated to the actual input

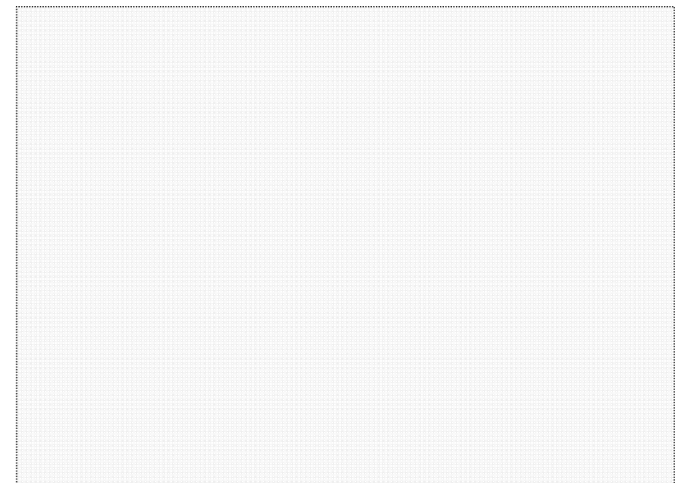


Selective-Input Attacks



Bar-Ilan University
Dept. of Computer Science

- ▶ **The drawback:**
 - Increases the size of the circuit
 - Increases the number of oblivious transfers
 - Need an oblivious transfer for each input bit
- ▶ Using randomized encoding of the input, this can be improved, but still costs

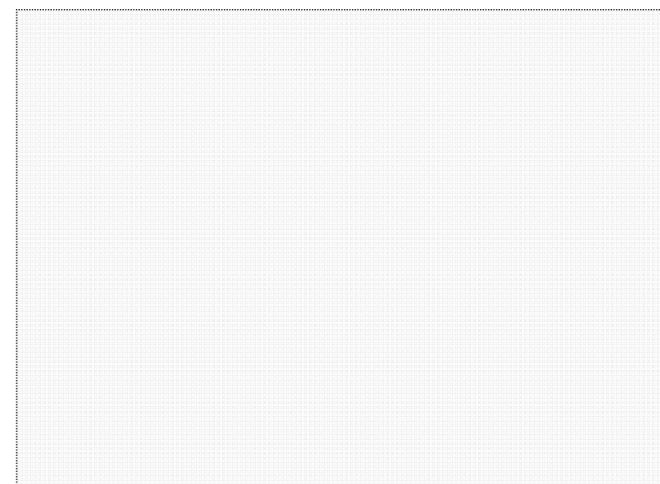


Input Consistency



Bar-Ilan University
Dept. of Computer Science

- ▶ **Forcing P_2 to use the same inputs in every circuit**
 - Carry out the oblivious transfers on all circuits at once (also more efficient)
- ▶ **In the i^{th} oblivious transfer**
 - P_1 (sender) inputs (K_0^i, K_1^i) where K_0^i is the vector of 0-keys in ALL circuits on the wire associated with P_2 's i^{th} input bit
 - P_2 (receiver) inputs its i^{th} input bit

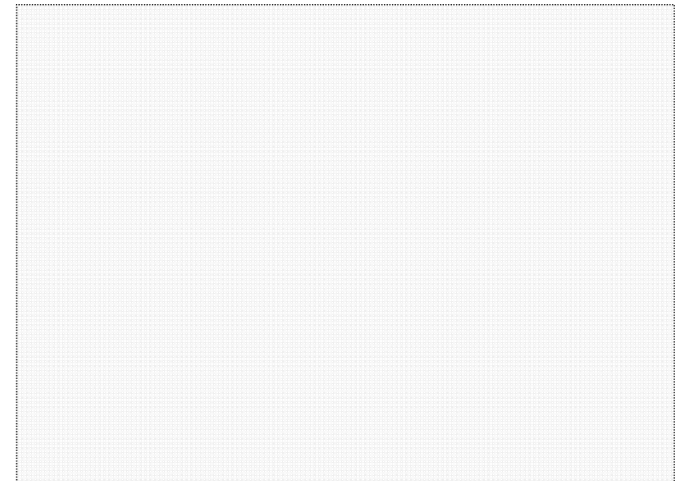


Input Consistency



Bar-Ilan University
Dept. of Computer Science

- ▶ **Forcing P_1 to use the same inputs in every circuit**
 - Use zero-knowledge – expensive
 - Use cut-and-choose on commitments
- ▶ **P_1 sends many sets of commitments to its input keys**
 - P_1 opens all commitments of opened circuits to show that correctly constructed
 - P_1 opens some commitments of computed circuits to show that it sent consistent keys

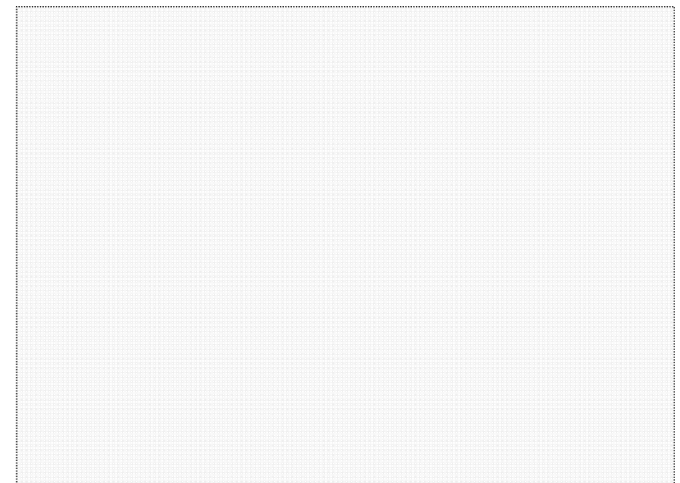


Input Consistency



Bar-Ilan University
Dept. of Computer Science

- ▶ **Cost: $2s^2L$ commitments are needed** (s is a statistical security parameter, L is the input length)
 - For $s = 160$, $n = 128$, this constitutes 6,553,600 commitments
 - In addition to significant computation (even if just hashing), this involves sending and processing a gigabit of data (if 160-bits is the size of each commitment)
- ▶ **This was a mistake...**

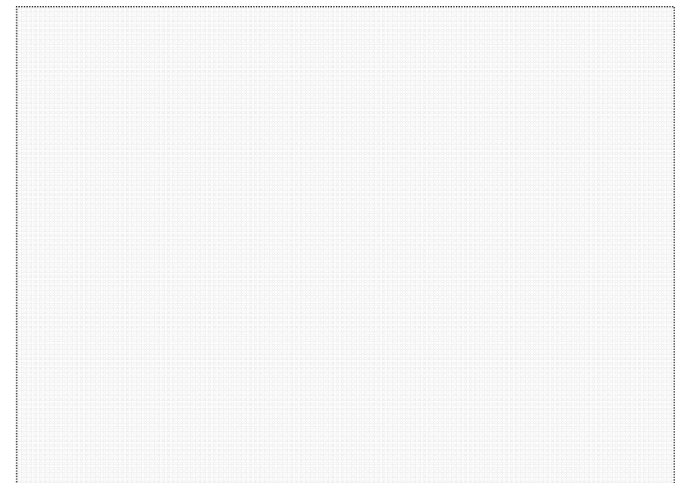


Security Parameter



Bar-Ilan University
Dept. of Computer Science

- ▶ **On the importance of tight proofs**
 - This protocol has a proven error of $2^{-s/17}$
 - The number of circuits sent and more is s
 - Thus, to obtain an error of 2^{-40} , we need to take $s=680$
- ▶ **This is a huge number of circuits**
 - It also means that the commitment sets are 20 gigabits)
- ▶ **We conjectured that the error is really $2^{-s/4}$ but are not sure**



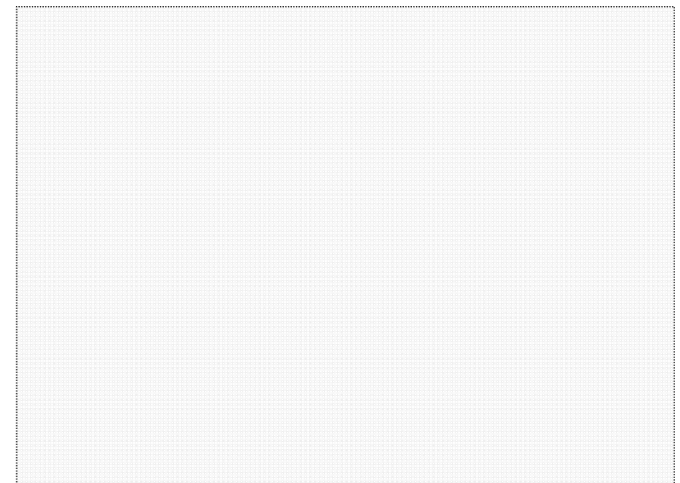
Efficiency...



Bar-Ilan University
Dept. of Computer Science

- ▶ **Efficiency means many things**
 - **Theoretical efficiency:** constant number of rounds, sublinear bandwidth, minimal number of oblivious transfers,...
 - **Concrete efficiency:** actual running time in comparison to other protocols

- ▶ **Both areas of research are important, but if you are doing concrete efficiency, then be concrete**

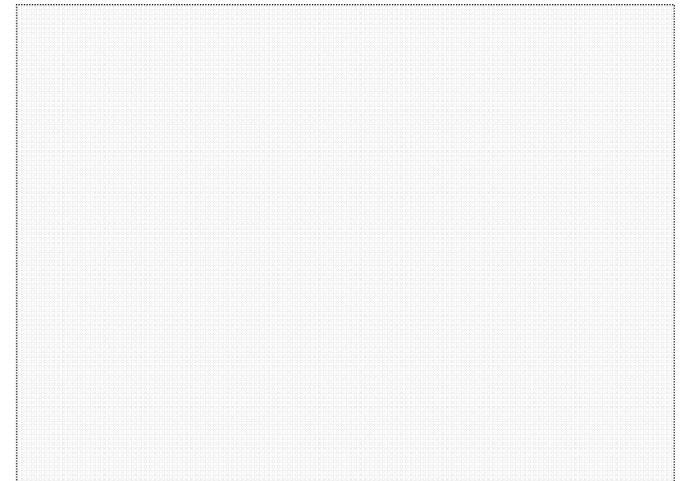


Implementations are Important



Bar-Ilan University
Dept. of Computer Science

- ▶ In [LP07], our aim was to reduce the number of oblivious transfers to a minimum
 - Symmetric operations, like commitments were assumed to be almost free
- ▶ In reality: the commitments are the bottleneck
 - They cost much more than the OTs

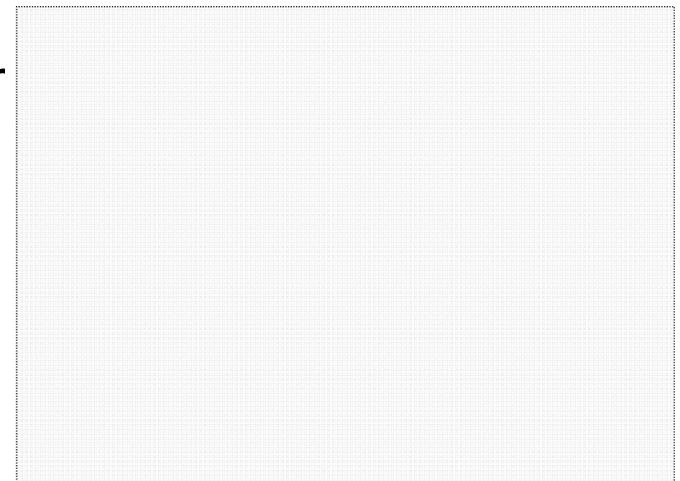


Solutions – Protocol 2011



Bar-Ilan University
Dept. of Computer Science

- ▶ Solution based on cut-and-choose, but using a very different approach
- ▶ More oblivious transfers and more exponentiations
 - No commitment sets
 - No selective-input attack is possible so don't need to split the inputs
 - Proven concrete error of $2^{-0.31s}$
 - Suffices to take $s=128$ for 2^{-40} error
 - Many less circuits – very important!

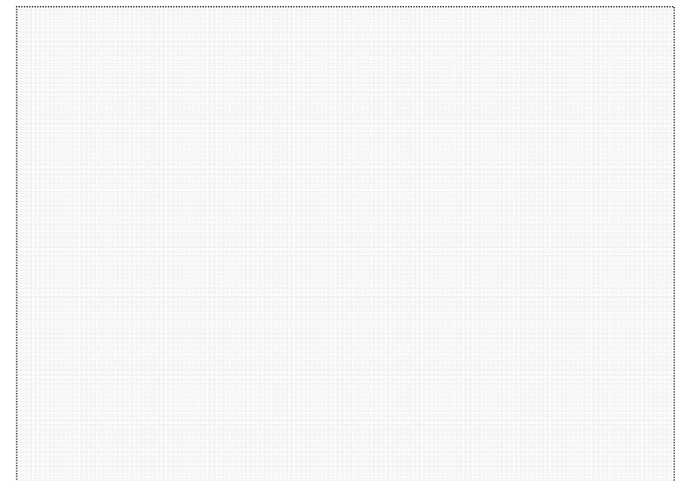


Consistency Proof



Bar-Ilan University
Dept. of Computer Science

- ▶ The keys on the wires associated with P_1 's input are chosen in a special way
 - Let r_1, \dots, r_s be random values (one for each circuit)
 - Let a_i^0, a_i^1 be random values (for the i^{th} bit of P_1 's input)
 - The keys for wire associated with the i^{th} bit of P_1 's input in the j^{th} circuit are $g^{a_i^0 \cdot r_j}, g^{a_i^1 \cdot r_j}$
 - P_1 sends $g^{r_1}, \dots, g^{r_s}, g^{a_1^1}, g^{a_1^0}, \dots, g^{a_L^0}, g^{a_L^1}$
 - These are commitments to all of the values on these wires
 - By DDH, the values are hidden



Consistency Proof



Bar-Ilan University
Dept. of Computer Science

► The proof

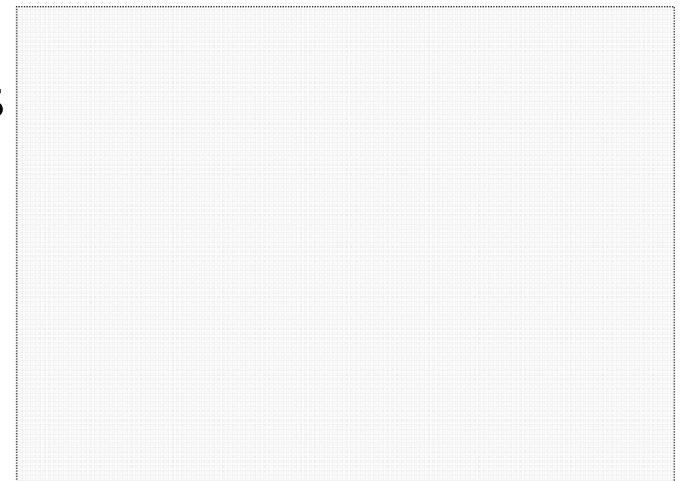
- Given $g^{r_1}, \dots, g^{r_s}, g^{a_1^1}, g^{a_1^0}, \dots, g^{a_L^0}, g^{a_L^1}$ and keys $k_i^1, k_i^2, \dots, k_i^s$ prove that there exists a bit $b \in \{0, 1\}$ such that

$$k_i^1 = g^{a_i^b \cdot r_1}, k_i^2 = g^{a_i^b \cdot r_2}, \dots, k_i^s = g^{a_i^b \cdot r_s}$$

- In other words, the key used for the i^{th} bit in all s circuits relates to the same bit (0 or 1)

► This looks complicated, but...

- This is an OR between two “extended Diffie–Hellman tuples”
- Using Sigma protocols, this can be proven with just $s+18$ exponentiations
 - First combine to one tuple (randomly), then prove OR of two DH tuples

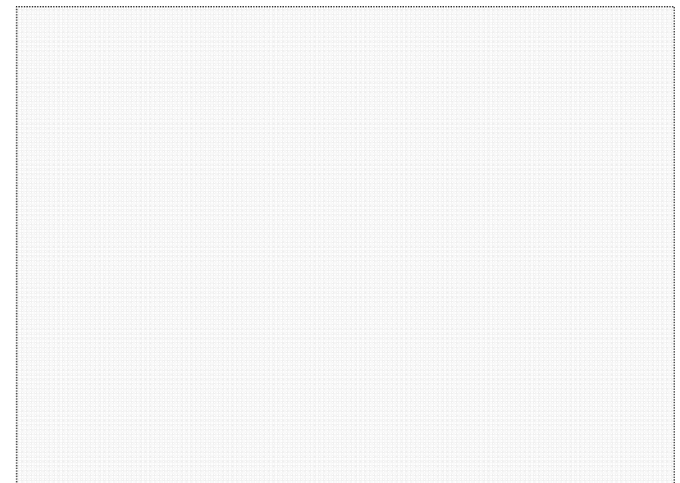


Cut-and-Choose OT



Bar-Ilan University
Dept. of Computer Science

- ▶ In the previous protocol, cut-and-choose on the circuits is separate from the OT
 - This enables P_1 to carry out a selective input attack because P_1 can use different keys in the OT to what are used in the opening
- ▶ In this protocol, we define cut-and-choose oblivious transfer to intertwine the two



Cut-and-Choose OT



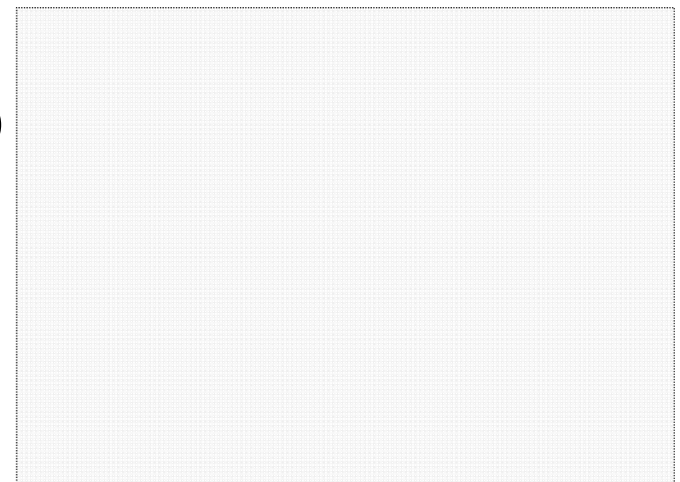
Bar-Ilan University
Dept. of Computer Science

► Input:

- The sender has a vector of s pairs
 - These are the keys for a wire associated with P_2 's input in all circuits
- The receiver has a bit
 - This is P_2 's input bit for this wire
- The receiver also has a set J of $s/2$ indices

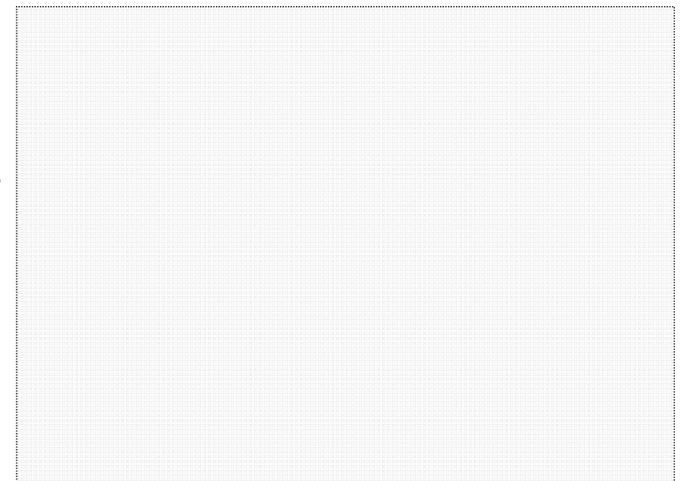
► Output:

- The receiver obtains the 1st or 2nd value in every pair (as per its input)
- The receiver obtains **both** values for every index in J



Using Cut-and-Choose OT

- ▶ P_1 sends the garbled circuits and the “commitments” to its own input wires
- ▶ P_1 and P_2 run cut-and-choose OT for the input wires of P_2 's input
- ▶ P_2 asks P_1 to send r_j for every $j \in J$
 - P_2 proves J by sending both values on some wire
 - This enables P_2 to compute all of the values on P_1 's input wires in the circuit
 - From the cut-and-choose OT it has all the values on its input wires
 - Thus, this is a full “opening”

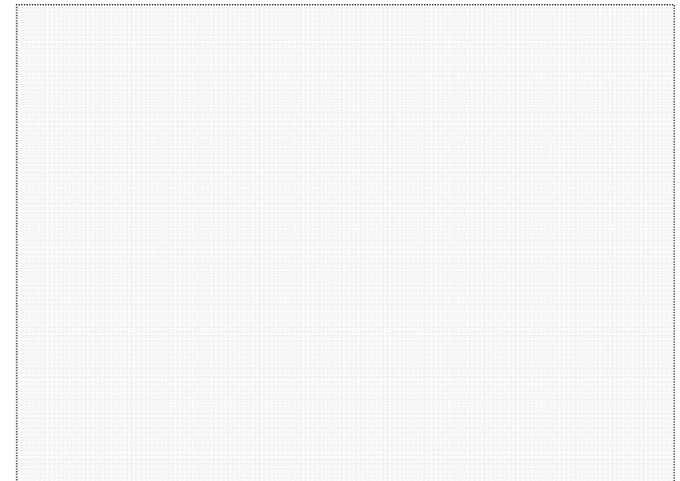


Using Cut-and-Choose OT



Bar-Ilan University
Dept. of Computer Science

- ▶ The circuit checks and the oblivious transfers are now intertwined
- ▶ Any incorrect value used in the oblivious transfers is either used few times (and so doesn't affect the majority) or used many times, and will be detected
- ▶ This also enables a much cleaner proof of security and analysis
 - There aren't different sources of error

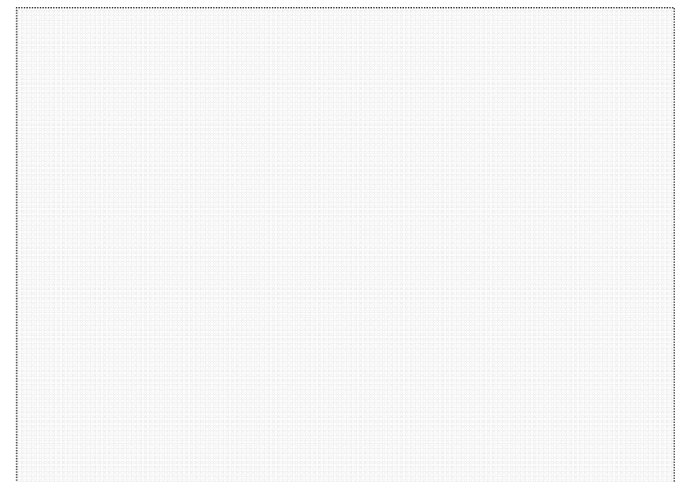


Cut-and-Choose OT



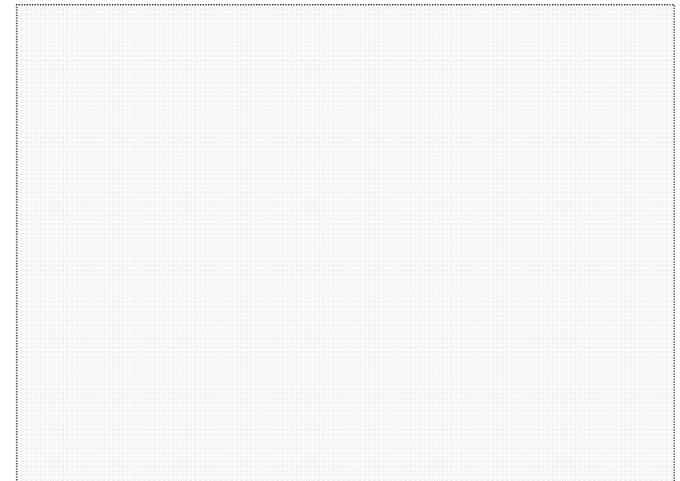
Bar-Ilan University
Dept. of Computer Science

- ▶ **Background – Oblivious Transfer of [PVW]**
 - RAND function: $\text{RAND}(w,x,y,z) = (u,v) = (w^s y^t, x^s z^t)$
 - If (w,x,y,z) is a DH tuple: $x=w^a, z=y^a$
 - $v = x^s z^t = w^{as} y^{at} = (w^s y^t)^a$ and so $v=u^a$
 - Thus, given $(u,v')=(u,v \cdot m)$ can compute $m = v'/u^a$
 - If (w,x,y,z) are not a DH tuple: $x=w^a, z=y^b$ ($a \neq b$)
 - $v = x^s z^t = w^{as} y^{bt}$; let $y = w^c$
 - Then $v = w^{as+cbt}, u=w^{s+ct}$
 - $as+cbt$ and $s+ct$ are linearly indep. equations and so for every m , there exist s,t such that $(u,v')=(u,v \cdot m)$



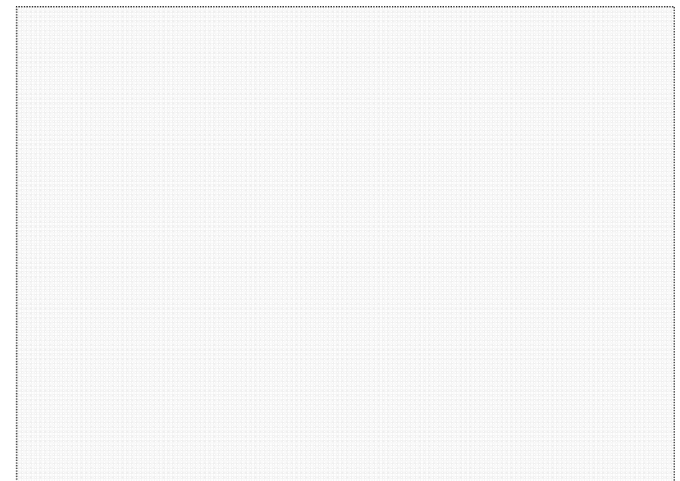
[PVW] Oblivious Transfer

- ▶ Inputs: $(m_0, m_1), \sigma$
 - ▶ Receiver R sends (g_0, g_1, h_0, h_1) that is not a DH tuple ($h_0 = g_0^a, h_1 = g_1^b, a \neq b$)
 - ▶ R chooses random r ; computes $g = g_\sigma^r, h = h_\sigma^r$
 - ▶ R sends (g, h) to S
 - ▶ S computes $(u_0, v_0) = \text{RAND}(g_0, g, h_0, h)$
 - ▶ S computes $(u_1, v_1) = \text{RAND}(g_1, g, h_1, h)$
 - ▶ S sends $(u_0, v_0 \cdot m_0), (u_1, v_1 \cdot m_1)$
- ▶ Only one of $(g_0, g, h_0, h), (g_1, g, h_1, h)$ is a Diffie-Hellman tuple



[PVW] Oblivious Transfer

- ▶ Only one of (g_0, g, h_0, h) , (g_1, g, h_1, h) is a Diffie–Hellman tuple
 - Recall: (g_0, g_1, h_0, h_1) is not a DH tuple; $h_0 = g_0^a$, $h_1 = g_1^b$
 - Thus, for every (g, h) , if $g = g_0^c$ and $h = h_0^c$, then it cannot be that $g = g_1^c$ and $h = h_1^c$
- ▶ **Security**
 - By what we have seen, this means that at least one of m_0, m_1 is perfectly hidden
 - The simulator can choose (g_0, g_1, h_0, h_1) as a DH tuple and so can extract both
 - By the DDH assumption, the sender also cannot know if (g, h) equals (g_0^r, h_0^r) or (g_1^r, h_1^r)



[PVW] Oblivious Transfer



Bar-Ilan University
Dept. of Computer Science

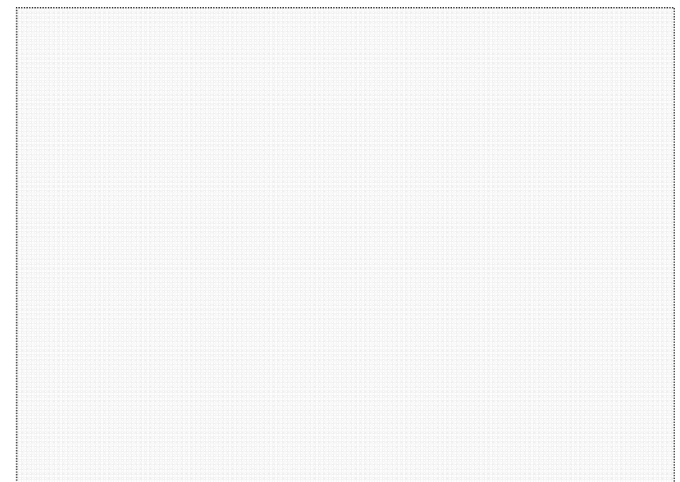
- ▶ What prevents R from sending a Diffie–Hellman tuple?
- ▶ R can prove in ZK that it's not a DH tuple
 - How can this be done efficiently?
- ▶ **Alternative: R computes $(g_0, g_1, h_0 = g_0^a, h_1 = g_1^{a+1})$**
 - Then, R proves that $(g_0, g_1, h_0, h_1 / g_1)$ is a DH tuple
 - This guarantees that (g_0, g_1, h_0, h_1) is not a DH tuple

Cut-and-Choose OT



Bar-Ilan University
Dept. of Computer Science

- ▶ We demonstrate this on two executions
 - Choose 1-out-of-2; same principle for many
- ▶ R chooses 2 tuples, one is DH and one is not
- ▶ R proves in ZK that 1 of 2 tuples is not DH
 - Use OR of sigma protocols
- ▶ R and S run the rest of [PVW] on each tuple
 - The execution for which the tuple is not DH is a regular OT
 - In the other execution, R receives both values, as required

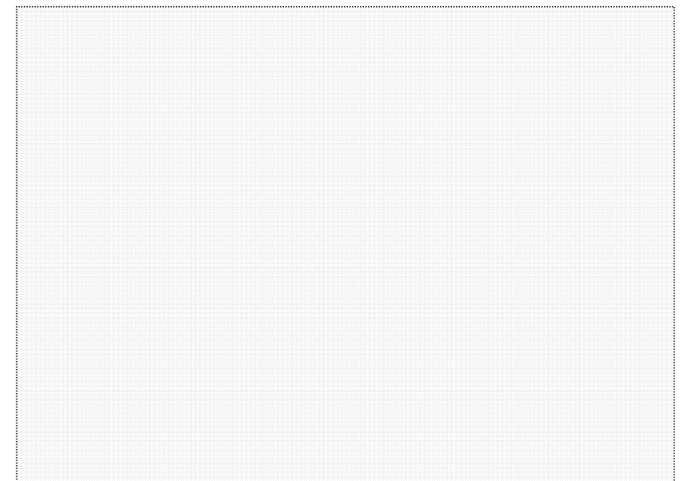


Lessons



Bar-Ilan University
Dept. of Computer Science

- ▶ It is possible to improve efficiency using ZK proofs intelligently
 - It's all about setting up the inputs in a way that is amenable to efficient proving
- ▶ Tight security reductions and proofs are crucial when considering concrete efficiency
- ▶ Constants are crucial for concrete efficiency
 - We didn't discuss this too much; except for the protocol of ZK-proving of Jarecki-Shmatikov (there $O(1) = 720$)

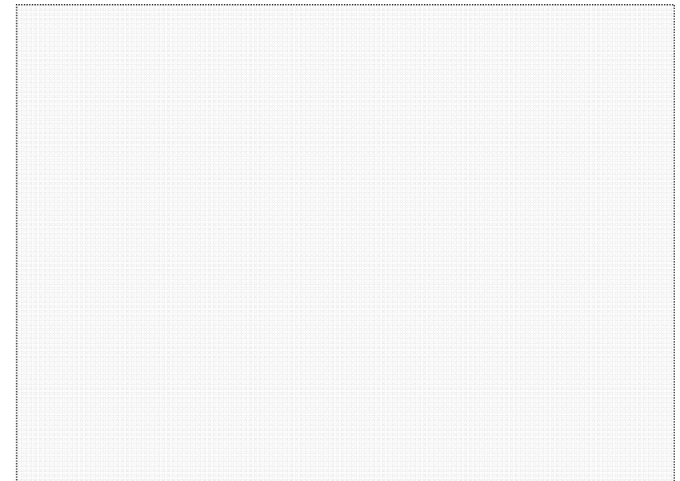


There is Much More



Bar-Ilan University
Dept. of Computer Science

- ▶ There are many considerations regarding concrete efficiency
 - We often count exponentiations, but:
 - A Paillier and RSA exponentiation is much more expensive than an Elliptic curve exponentiation
 - A pairing exponentiation is like an RSA exponentiation (plain DH is best out of these)
 - Multi-exponentiations of the type $g^s h^r$ cost about 1.33 regular exponentiations
 - This is just one example



Conclusion



Bar-Ilan University
Dept. of Computer Science

- ▶ We can compute any function for malicious adversaries with reasonable efficiency
- ▶ There is still a long way to go
 - The blowup of 128 times Yao is problematic
 - Other solutions requiring $O(1)$ or more exponentiations per gate are also problematic
- ▶ This is currently a very active research area
 - In 2006, there was nothing, now there are at least 5 different approaches

