MPC – Lecture 7:

OTTT & BeDoZa against malicious adversary

DR. ADI AKAVIA

UNIVERSITY OF HAIFA

High Level Approach

Force semi-honest behavior

Discuss: How to secure protocols again a malicious adversary?

Idea: <u>Force</u> adversary to behave semi-<u>honestly</u>

How? 1) Focus on security-with-abort

2) Devise protocols that **identify** deviations from protocol's specifications.

3) If deviation occurs, **abort**

Essentially, security is reduced to **identifying** deviation from protocol's specifications

How to identify deviation from protocol?

Approach 1 [GMW]: <u>**Prove**</u> each step is computed according to protocols specification.

| later in this course |

Is it possible? Yes, it's an NP statement, can be proved.

Use **ZK-proofs** so that the proof doesn't leak information.

Approach 2 [……]: Use authenticated messages (**MAC**s) so that deviation from protocol will fail to authenticate

| today |

OTTT against Malicious Adversary

OTTT for Passive Adversary (Recap)

Offline (Dealer has truth table $F[x,y] = f(x,y)$):

1) Let $F_{s,r}[i,j] = F[i-r,j-s]$ be a rotating F's rows (cols.) by r (s) for random r,s.

2) Send to Bob: (M_B , s) for M_B uniformly random matrix.

3) Send to Alice: (M_A , r) for $M_A = F_{s,r} – M_B \bmod p$.

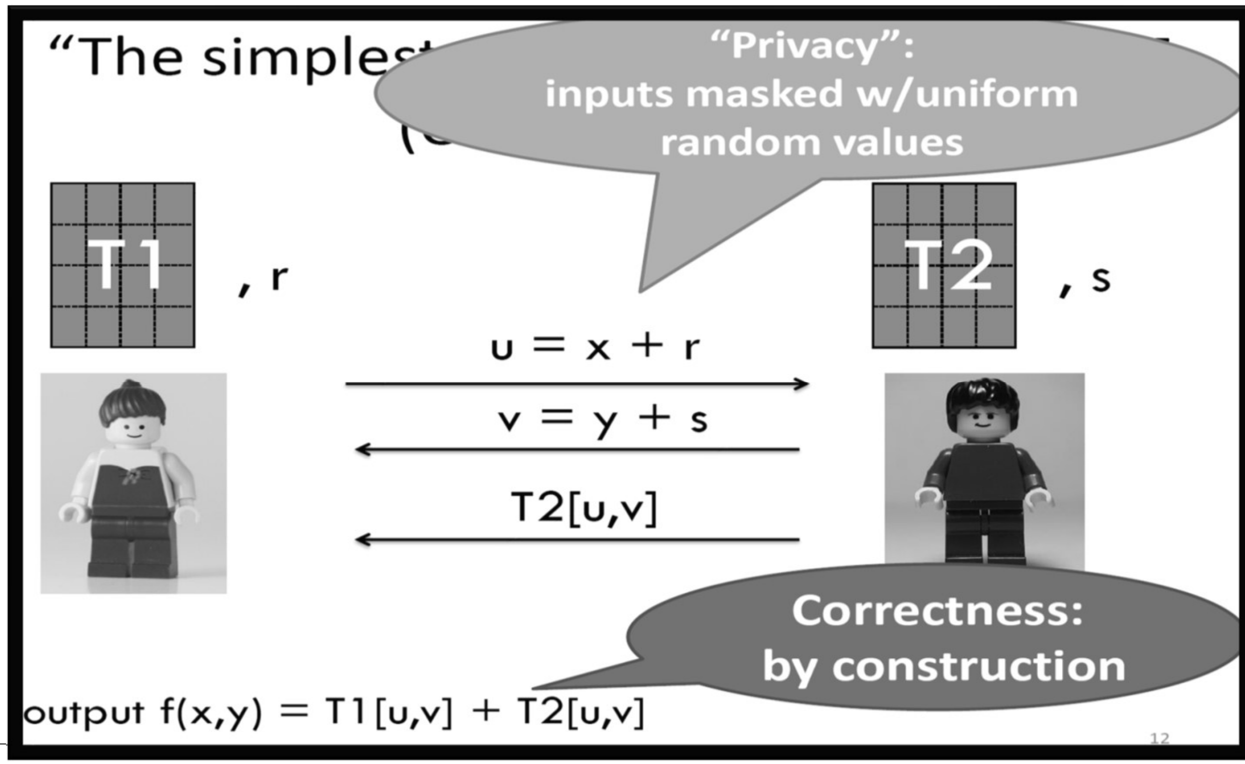
Online (Alice has (x, M_A, r), Bob has (y, M_B, s)):

1) Alice sends to Bob $u = x+r$

2) Bob sends to Alice $v = y+s$ and $M_B[u,v]$

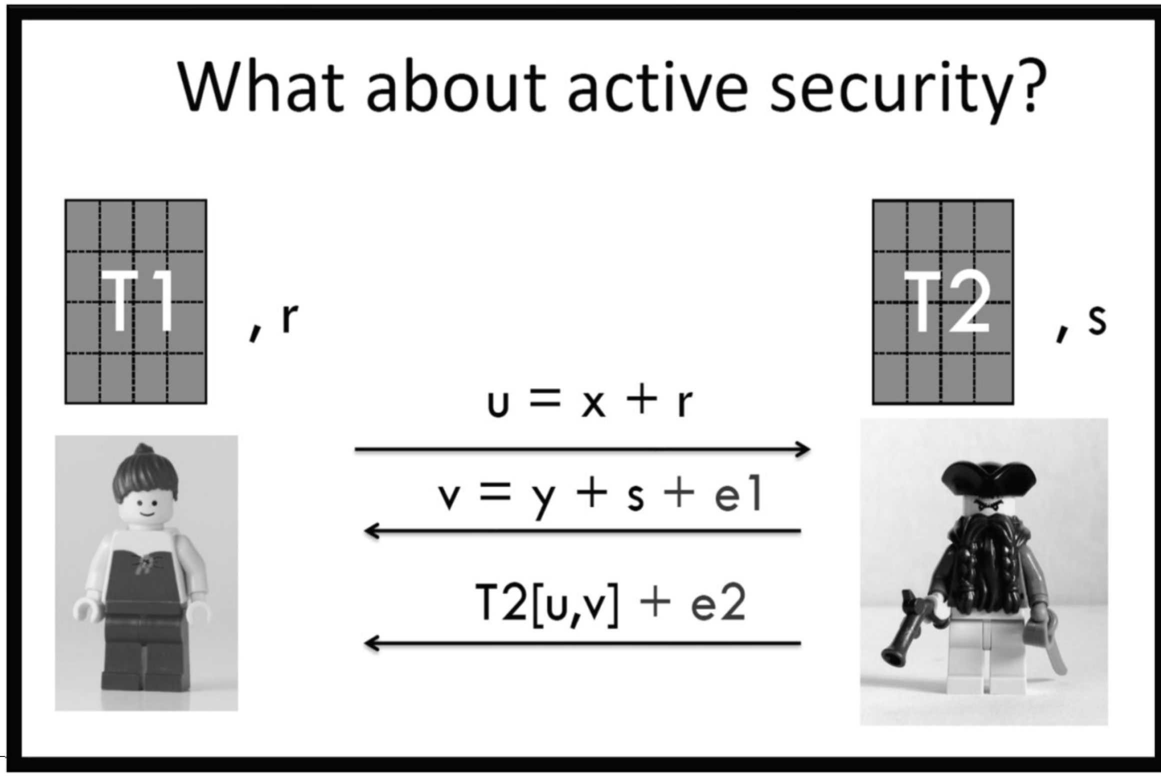
3) Alice outputs $M_A[u,v]+M_B[u,v]$

Passive OTTT, pictorially



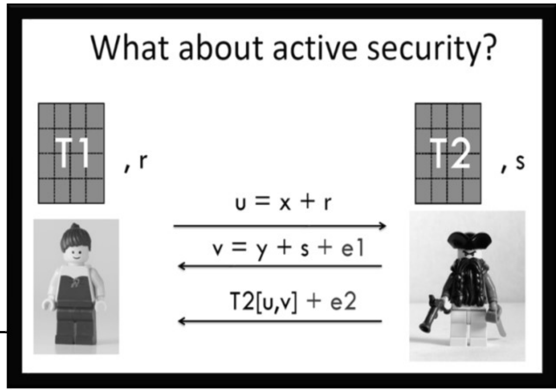
Imgs from Claudio Orlandi

Passive OTTT: What can malicious Bob do?



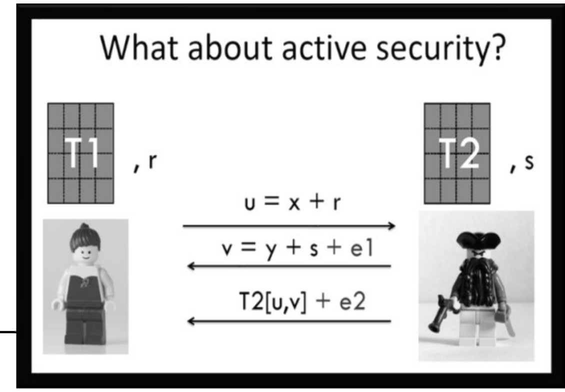
Imgs from Claudio Orlandi

Passive OTTT:
What can malicious Bob do?



Q: Is this “cheating”?

Passive OTTT: What can malicious Bob do?



Q: Is this “cheating”?

A1: $v = y + s + e_1$

$= (y+e_1) + s = y' + s$

Equiv. to input substitution **– not cheating**, as feasible also in Ideal-World.

A2: $M_B[u,v] + e_2$

Changes the output to $z' = f(x,y) + e_2$

A breach of correctness!

How to force Bob to send right value?

Challenge: Bob can send the wrong shares

Solution approach: Employ **MAC**s to let Alice detect wrong shares.

Message Authentication Codes

Message Authentication Codes (MAC)

Definition: A MAC scheme has three algorithms (Gen,Tag,Ver) where

- K ← Gen(sec param) produce a MAC key k
- t ← Tag(k, x), given a key k and message x, outputs a tag t.
- accept/reject ← Ver(k, t, x), given a key k, a tag t and message x, outputs accept if t is a valid tag for x with key k or reject otherwise.

Definition: MAC Security

The Security Game: Challenger C samples a key k,

Adversary A can (adaptively) send to C a sequence or q messages $x_1, \ldots, x_q$ of his choice and receive corresponding tags $t_1, \ldots, t_q$. where $t_i \leftarrow \mathrm{Tag}(k, x_i)$

A outputs (t' , x').

A **wins** if Ver(k, t', x')=accept and $x' \notin \{x_1, .., x_q\}$

Definition [MAC security]: A MAC scheme is *(q,ε)-secure* if for every adversary A, A wins the security game with probability at most ε.

Construction: One-time MAC

One-Time MAC.

- Gen(p): Sample $k = (a,b) \leftarrow Z_p^2$ for a prime p.
- Tag(k=(a,b),x): Output $t = ax + b$.
- Ver(k=(a,b),t,x): Output accept if $t = ax + b$, reject o/w.

Exercise. Prove that this is a (1,1/p)-secure MAC scheme (even against unbounded adversary).

OTTT with MACs

OTTT with MACs: Offline Phase

Offline (Dealer has truth table $F[x,y] = f(x,y)$ for $x,y \in \{0,1\}^n$):

1) Set $F_{s,r}[i,j] = F[i-r,j-s]$ for $r,s \leftarrow_R [2^n]$
$M_B \leftarrow_R \{0,1\}^{2^n \times 2^n}$ and $M_A = F_{s,r} - M_B \bmod p$

2) Set $K[i,j] \leftarrow Gen(p)$ $\forall i,j \in [2^n]$ (keys for a $(1,\varepsilon)$-secure MAC)

3) Set $T[i,j] \leftarrow tag(K[i,j] , M_B[i,j])$ $\forall i,j \in [2^n]$ (tags for all M_B's entries)

4) Output (r, M_A, K) to Alice and (s, M_B, T) to Bob.

OTTT with MACs: Online Phase

Online (Alice has (x, r, M_A, K), Bob has (y, s, M_B, T)):

1) Alice sends to Bob $u = x+r \bmod 2^n$.

2) Bob sends to Alice $v = y+s \bmod 2^n$ and $z_B = M_B[u,v]$ and $t_B = T[u,v]$.

3) Alice computes $Ver(K[u,v], t_B, z_B)$

   If verification fails, Alice outputs $z = f(x,y_0)$

   Else Alice outputs $z = M_A[u,v] + z_B$

Security against corrupt Bob (sketch)

Simulator S_B(Adv)**:** // Adv real-world adversary controlling Bob
// S_B(Adv) ideal-world adversary controlling Bob

1) Replacing the trusted dealer: Send to adv. Adv controlling Bob (s, M_B, T) for:

uniformly random col. shift s and matrix M_B,
freshly generated MAC keys K,
corresponding MAC tags $T = Tag(K, M_B)$ (entry-by-entry).

2) Replacing the honest Alice: Send to adv. Adv random row u.

Security against corrupt Bob (sketch)

Simulator $S_B(A)$ – cont'd**:**

1) Replacing the trusted dealer: Send to adv. Adv controlling Bob (s, M_B, T)

2) Replacing the honest Alice: Send to adv. Adv random row u.

3) Input extraction: If adversary Adv (controlling Bob) sent (v, z_B, t_B)

s.t. $Ver(K[u,v], z_B, t_B)$=accept and $z_B = M_B[u,v]$

Then S_B sets Bob's input to: $y = v - s$ (& *deliver)*

Else default input value: y_0 (& *abort)*

4) Output (as Bob's view): (s, M_B, T, u)

Security against corrupt Bob (sketch)

We next show that $\mathrm{REAL}_{\mathrm{Adv}} \equiv_{1/p} \mathrm{IDEAL}_{\mathrm{S(Adv)}}$ where:

◦ REAL_{A} = (	(s, M_B, T, u),	$M_A[u,v] \oplus z_B$	or *abort*	)
protocol's:	Bob's view	Alice's output in protocol		
◦ $\mathrm{IDEAL}_{S(A)}$ = (	(s, M_B, T, u),	$f(x,y)$	or *abort*	)
ideal world's:	Bob's sim. view	Alice's output from $\mathcal{T}$		

Security against corrupt Bob (sketch)

Recall: $\text{REAL}_{\text{Adv}} = ((s, M_B, T, u),\ M_A[u,v] \oplus z_B))$ and $\text{IDEAL}_{S(\text{Adv})} = ((s, M_B, T, u),\ f(x,y))$

We sketch why $\text{REAL}_{\text{Adv}} \equiv_{1/p} \text{IDEAL}_{S(\text{Adv})}$:

- (s,M_B) : identically distributed in both worlds
- u : identically distributed in both worlds (uniform in IDEAL, and u=x+r for uniform r in REAL)
- Alice's output : identically distributed in both worlds, **<u>except</u>** if adv. Adv sent a triple (v', z_B', t_B') such that:

  $z_B' \neq M_B[u,v']$ while $Ver(K[u,v], z_B', t_B')=$*accept* .

In this case, Alice's output is incorrect in REAL WORLD but correct in IDEAL world.

Security against corrupt Bob (sketch)

Recall: $\text{REAL}_{\text{Adv}} = ((s, M_B, T, u), \quad M_A[u,v] \oplus z_B))$ and $\text{IDEAL}_{S(\text{Adv})} = ((s, M_B, T, u), \quad f(x,y))$

We sketch why $\text{REAL}_{\text{Adv}} \equiv_{1/p} \text{IDEAL}_{S(\text{Adv})}$:

- (s, M_B) : identically distributed in both worlds
- u : identically distributed in both worlds
  (uniform in IDEAL, and u=x+r for uniform r in REAL)
- Alice's output: identically distributed in both worlds,
  except if adv. Adv sent a triple (v', z_B', t_B') such that:
  $z_B' \neq M_B[u,v']$ while $\text{Ver}(K[u,v], z_B', t_B')=\textit{accept}$.

  But this breaks the MAC security
  $\Rightarrow$ occurs with probability 1/p.

Wrapping up.

Setting $p=2^{secParam}$ $\Rightarrow$ $1/p$ is negligible in secParam

$\Rightarrow$ $REAL_{Adv} \equiv_{stat} IDEAL_{S(Adv)}$

Conclusion: The OTTT-w-MAC protocol securely computes f against a malicious adversary.

BeDoZa against Malicious Adversary

BeDoZa for Passive Adversary (Recap)

Wires' values are secret shared between Alice and Bob:

- Secret share input wires
- Propagates secret sharing layer by layer,
- Once obtained a secret sharing of the output wire, open.

Passive BeDoZa: What can malicious adversary do?

Input wires: Send wrong shares in ShrA (sim. ShrB)

– Allowed: "input substitution".

XOR([x],[y]), AND(c,[y]): Modify local computation

– Not allowed ("additive attacks")

AND([x],[y]), Output Wire: Send wrong shares in OpenTo

– Not allowed ("additive attacks").

How to force sending correct value?

Approach: Use MACs to prevent (undetected) attacks.

Challenge: Want linear operations “for free” on shared value.

BEDOZa-w-MAC: Try 1

Try 1 – Use **"homomorphic MAC"** that

- Given tags t_1,t_2 for msgs x_1,x_2 (without key k !),
- Support generating a MAC for any linear combination of x_1,x_2.
- E.g. $t' = a\,t_1 + b\,t_2$ a valid MAC for msg $x' = a\,x_1 + b\,x_2$

Problem: "homomorphic MAC" is **insecure**!
allows adversary to **forge** MACs!

Better approach – MACs that support "limited homomorphism"

Homomorphic m-time MAC

Definition: m-hom MAC Security

The Security Game for m-hom MAC:

Adversary A can query challenger C on messages $x_1,\ldots, x_m$ of his choice (adaptively) and receive corresponding tags $t_1,\ldots,t_m$, where $t_i \leftarrow Tag(k_i, x_i)$

for $k_i = (\alpha, \beta_i)$ for $\alpha, \beta_1, \ldots, \beta_m \leftarrow_R Z_p$.

A outputs (i, t' , x') and A **wins** if Ver(k_i, t', x')=accept and $x' \neq x_i$

Definition [m-hom MAC security]: A m-hom MAC scheme is *(m,ε)-secure* if every adversary A wins the security game with probability at most ε.

Construction: m-hom MAC

m-Time MAC.

- Gen(p): Sample $\alpha, \beta_1,\ldots, \beta_m \leftarrow_R Z_p$ for a prime p, |p|=secParam.
  Output $k_1,..,k_m$ where $k_i=(\alpha, \beta_i)$
- Tag($k_i=(\alpha, \beta_i)$,x): Output $t = \alpha x + \beta_i$.
- Ver(k_i, t, x): Output accept if $t_i = \alpha x + \beta_i$, reject o/w.

Exercise 1. Prove that this is a (m,1/p)-secure m-hom MAC scheme (even against unbounded adversary).

Homomorphism

m-Time MAC.

- Gen(p): Sample $\alpha, \beta_1, \ldots, \beta_m \leftarrow_R Z_p$ for a prime p, $|p|$=secParam.
  Output $k_1,..,k_m$ where $k_i=(\alpha, \beta_i)$
- Tag($k_i=(\alpha, \beta_i)$,x): Output $t = \alpha x + \beta_i$.
- Ver(k_i, t, x): Output accept if $t_i = \alpha x + \beta_i$, reject o/w.

Exercise 2. Prove that given two tags $\mathbf{t_1,t_2}$ for msgs $\mathbf{x_1,x_2}$ and keys $\mathbf{k_1,k_2}$,
We can compute a valid tag **t'** for msg $\mathbf{x' = x_1 + x_2}$.

Hint: Use a new key **k'** .

Authenticated Secret Sharing

Unauthenticated Secret Sharing (Recap)

Recall: (unauthenticated) *secret sharing* for x (denoted [x]),

Alice held x_A
Bob held x_B

where 1) (x_A, x_B) are uniformly random
subject to: $x_A + x_B = x \mod p$.

Authenticated Secret Sharing

Def: *<u>authenticated</u> secret sharing* for value x (denoted [x]),

Alice holds $(x_A , k_{A,x} , t_{A,x})$

Bob holds $(x_B , k_{B,x} , t_{B,x})$

where 1) (x_A, x_B) are uniformly random
subject to: $x_A + x_B = x \mod p$.

2) $k_{A,x}$ and $k_{B,x}$ are fresh **MAC Keys**

3) $t_{A,x}$ and $t_{B,x}$ are corresponding **MAC Tags**: $t_{A,x} = Tag(k_{B,x}, x_A)$
$t_{B,x} = Tag(k_{A,x}, x_B)$

BeDoZa with MACs

BeDoZa with MACs: The Idea

Idea: Use <u>*authenticated*</u> *secret sharing* for all wire values x .

BeDoZa with MACs: The Invariant

Remarks:
- same α_A, α_B for all wires
- arithmetic circuit mod-p
- $x \in Z_p$

The Invariant: For each wire value $x \in Z_p$,

Alice holds $(x_A , k_{A,x} , t_{A,x})$

Bob holds $(x_B , k_{B,x} , t_{B,x})$

where 1) (x_A, x_B) are uniformly random subject to: $x_A + x_B = x \mod p$.

2) $k_{A,x}$ and $k_{B,x}$ are MAC Keys: $k_{A,x} = (\alpha_A, \beta_{A,x})$
$k_{B,x} = (\alpha_B, \beta_{B,x})$

3) $t_{A,x}$ and $t_{B,x}$ are corresponding Tags: $t_{A,x} = \mathrm{Tag}(k_{B,x}, x_A)$
$t_{B,x} = \mathrm{Tag}(k_{A,x}, x_B)$

BeDoZa with MACs: Subprotocols OpenTo

OpenTo(A, [x]): Bob sends x_B and $t_{B,x}$ to Alice
Alice outputs $x = x_A + x_B$ if $\mathrm{Ver}(k_{A,x}, t_{B,x}, x_B)$=accept (o/w abort)

OpenTo(B, [x]): Analogous.

Open([x]): Run both OpenTo(B, [x]) and OpenTo(A, [x]).

Denote: $(x,\perp) \leftarrow \mathrm{OpenTo}(A, [x])$

BeDoZa with MACs: Subprotocols Addition Gates

Add([x],[y]): Alice outputs $(z_A, k_{A,z}, t_{A,z})$ where
$$z_A = x_A + y_A$$
$$k_{A,z} = (\alpha_A, \ \beta_{A,x} + \beta_{A,y})$$
$$t_{A,z} = t_{A,x} + t_{A,y}$$

Bob outputs $(z_B, k_{B,z}, t_{B,z})$ where
$$z_B = x_B + y_B$$
$$k_{B,z} = (\alpha_B, \ \beta_{B,x} + \beta_{B,y})$$
$$t_{B,z} = t_{B,x} + t_{B,y}$$

Denote: $[z] \leftarrow \mathrm{Add}([x],[y])$

BeDoZa with MACs: Subprotocols
Addition Gates

Exercise: Write subprotocol for Add([x],c):

BeDoZa with MACs: Subprotocols
Addition Gates

Solution:

Add([x],c): Alice outputs $(z_A\ ,\ k_{A,z}\ ,\ t_{A,z})$ where $z_A = x_A + c$

$k_{A,z} =$ __________

$t_{A,z} =$ __________

Bob outputs $(z_B\ ,\ k_{B,z}\ ,\ t_{B,z})$ where $z_B = x_B$

$k_{B,z} =$ ____________

$t_{B,z} =$ ____________

BeDoZa with MACs: Subprotocols
Input wires sharing Shr(A,x) , Shr(B,x)

To share each of Alice's (resp. Bob's) **input wires**:

1) The **dealer** D outputs a random authenticated secret sharing [r]

2) Alice & Bob run $(r,\perp) \leftarrow \text{OpenTo}(A,[r])$

3) Alice sends Bob $d = x - r$

4) Alice & Bob compute $[x] = [r] + d$

Discuss: Why dealer chooses r ?

BeDoZa with MACs: Subprotocols Multiplication Gates

Similarly to passive BeDOZa,

<u>except</u> for using **authenticated Beaver triple** ([u],[v],[w]) from dealer.

Exercise: Write subprotocol for Mult([x],c) , Mult([x],[y]).
Argue correctness.

BeDoZa with MACs: Comments

Alice & Bob never generate MAC keys or tags in the protocol.

They only compute linear combinations of shares, MACs and keys, and verify correctness of received MACs.

All keys and tags in protocol are generated by trusted dealer:

for input wires in [r],

for multiplication gates in authenticated Beaver triples.

BeDoZa with MACs: Security (sketch)

1) Simulator S_A for corrupt Alice (resp. S_B):

- **Input wires –** Shr(A,x): S_A (plays dealer's role): Chooses random r, sends [r] to Alice
  S_A (plays Bob's role): Receives msg d from Alice, (internally) extracts Alice's input $x' = d + r$
- **Input wires –** Shr(B,x): S_A (plays Bob's role): Sends random share to Alice.
- **Internal gates:** Simulated similarly to passive secure case, but where simulator verifies Bob's msg and aborts if Ver(..) rejects.

2) Real ≡ Ideal

- If there was no undetected forged MAC (similarly to OTTT proof).
- The probability of undetected forged msg is $1/p$
- Take $p \sim 2^{secParam}$

BeDoZa with MACs: Wrapping up.

Setting $p=2^{secParam}$ $\Rightarrow$ $1/p$ is negligible in secParam

$\Rightarrow$ $REAL_A \equiv_{stat} IDEAL_{S(A)}$

Conclusion: The BeDoZa-w-MAC protocol securely evaluates C against a malicious adversary.

BeDoZa with MACs: Efficiency

Constant overhead over passive version in #stored values:

- Three values in Z_p for each wire with value
- But how large is p ?

BeDoZa with MACs: Efficiency cont'd

Constant overhead over passive version in #stored values:

- Three values in Z_p for each wire with value
- But how large is p ?

Problem: Security requires large p (~40-60 bits long)

- Bad for efficiency
- Also, how to securely evaluate circuits with arithmetic mod small p? E.g Boolean circuit?

Extensions: small field arithmetic

TinyOT: Idea: Use k MACs for each value, small p (e.g. p=2).

Pr[undetected forged msg] = $(1/p)^k$

Efficiency: ×3k bits over cleartext

MiniMAC: Idea: MAC together *vectors of bits* .

Efficiency: O(1) (amortized) overhead

Extensions: n parties

BeDOZa: Each party has key and MAC for each other party.
O(n) storage overhead for each party.

SPDZ: Instead of putting MACs on shares:
- MACs computed on values, and
- MACs and keys secret shared

Introduces new problems; details in papers.

Efficiency: each party stores only 3 values (x_i, k_i, t_i) s.t. $\Sigma_i\, t_i$ a valid tag on msg $\Sigma_i\, x_i$ with key $\Sigma_i\, k_i$.

Summary of Today's Class

1) OTTT secure against malicious adversary

2) BeDoZa secure against malicious adversary

3) Common approach: force honest behavior

4) Common technique: use MAC to detect deviation from honest behavior

… Next time: getting rid of the Trusted Dealer