NISC: Non-Interactive Secure Computation

Yuval Ishai
Eyal Kushilevitz
Rafail Ostrovsky
Manoj Prabhakaran
Amit Sahai
NISC: A motivation
NISC: A motivation

Dating for cryptographers
NISC: A motivation

Dating for cryptographers

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
</tr>
<tr>
<td>--------</td>
</tr>
</tbody>
</table>

status: looking

encrypted preferences
NISC: A motivation

Dating for cryptographers

Profile
Name:
Age:
Sex:
Interests:
Contact info

Prefs
Age range:
Sex:
Interests include:

status: looking
encrypted preferences
NISC: A motivation

Dating for cryptographers

Functionality: Inputs: Preferences from Alice & Profile from Bob
Output: If match, then give Bob’s contact information to Alice (Bob learns nothing)
NISC: A motivation

- Dating for cryptographers

Functionality: Inputs: Preferences from Alice & Profile from Bob

Output: If match, then give Bob’s contact information to Alice (Bob learns nothing)

Feature: Alice can post her preferences and go offline
Yao’s Garbled Circuit
Yao’s Garbled Circuit

Allows Alice to compute $f(x,y)$ without learning anything more about $y$ (and without Bob learning about $x$) \cite{Y86}
Yao’s Garbled Circuit

- Allows Alice to compute $f(x,y)$ without learning anything more about $y$ (and without Bob learning about $x$) [Y’86]

- Combined with a 2-message OT protocol: A non-interactive scheme, where Alice publishes an encoding of $x$ and Bob can transfer $f(x,y)$ to her.
Yao’s Garbled Circuit

 Allows Alice to compute $f(x,y)$ without learning anything more about $y$ (and without Bob learning about $x$) [Y’86]

 Combined with a 2-message OT protocol: A non-interactive scheme, where Alice publishes an encoding of $x$ and Bob can transfer $f(x,y)$ to her

 Security only against honest-but-curious Bob (even if ideal OT)
Yao’s Garbled Circuit

- Allows Alice to compute $f(x,y)$ without learning anything more about $y$ (and without Bob learning about $x$) [Y’86]

- Combined with a 2-message OT protocol: A non-interactive scheme, where Alice publishes an encoding of $x$ and Bob can transfer $f(x,y)$ to her

- Security only against honest-but-curious Bob (even if ideal OT)

- Our Problem: Obtain the same result against malicious players
  - With blackbox PRG (and as little overhead as possible)
Yao’s Garbled Circuit

- Allows Alice to compute $f(x,y)$ without learning anything more about $y$ (and without Bob learning about $x$) [Y’86]

Combined with a 2-message OT protocol: A non-interactive scheme, where Alice publishes an encoding of $x$ and Bob can transfer $f(x,y)$ to her

- Security only against honest-but-curious Bob (even if ideal OT)

Our Problem: Obtain the same result against malicious players

- With blackbox PRG (and as little overhead as possible)
NISC/OT
(NISC over OT)
NISC/OT
(NISC over OT)

**Functionality** (single instance version): Alice and Bob give x and y respectively. Alice gets f(x,y).

**Structure of protocol:**
- Alice and Bob invoke several instances of OT in parallel with Alice as receiver
- Alice then carries out a local computation, and outputs f(x,y) (or “abort”)

**Security:** UC security (against active corruption) in the OT-hybrid model
NISC/OT

(NISC over OT)

- **Functionality** (single instance version): Alice and Bob give x and y respectively. Alice gets f(x,y).

- **Structure of protocol:**
  - Alice and Bob invoke several instances of OT in parallel with Alice as receiver.
  - Alice then carries out a local computation, and outputs f(x,y) (or "abort").

- **Security:** UC security (against active corruption) in the OT-hybrid model.
NISC/OT
(NISC over OT)

**Functionality** (single instance version): Alice and Bob give x and y respectively. Alice gets f(x,y).

**Structure of protocol:**
- Alice and Bob invoke several instances of OT in parallel with Alice as receiver.
- Alice then carries out a local computation, and outputs f(x,y) (or “abort”)

**Security:** UC security (against active corruption) in the OT-hybrid model.

Similarly NISC/H for other (non-reactive, one-sided-output) H.
Functionality (single instance version): Alice and Bob give x and y respectively. Alice gets f(x,y).

Structure of protocol:
- Alice and Bob invoke several instances of OT in parallel with Alice as receiver
- Alice then carries out a local computation, and outputs f(x,y) (or “abort”)

Security: UC security (against active corruption) in the OT-hybrid model

Similarly NISC/H for other (non-reactive, one-sided-output) H

NISC/OT can be converted to NISC/CRS using [PVW’08]

Alice doesn’t get any output until she gives inputs to all OT instances
NISC Results
NISC Results

- NISC considered in [RAD’78, Y’86, SYY’99,...]
NISC Results

- NISC considered in [RAD’78,Y’86,SYY’99,...]
- NISC for general (poly-time computable) functions:
  - Honest-but-curious players:
    - NISC/OT using Yao’s garbled circuit [Y’86]
    - NISC from fully homomorphic encryption [RAD’78,G’09, GHV’10,...]: Low communication (but currently less practical); uses more than PRG+OT
  - Malicious players:
    - Use a NIZK to prove correctness of messages sent [CCKM’00, HK’07]: Expensive, and non-blackbox use of PRG (used for encryption in Yao)
    - [IPS’08 (full version)]: using “MPC-in-the-head.” Non-blackbox use of PRG
NISC Results

NISC considered in [RAD’78, Y’86, SYY’99, ...]

NISC for general (poly-time computable) functions:

Honest-but-curious players:
- NISC/OT using Yao’s garbled circuit [Y’86]
- NISC from fully homomorphic encryption [RAD’78, G’09, GHV’10, ...]: Low communication (but currently less practical); uses more than PRG+OT

Malicious players:
- Use a NIZK to prove correctness of messages sent [CCKM’00, HK’07]: Expensive, and non-blackbox use of PRG (used for encryption in Yao)
- [IPS’08 (full version)]: using “MPC-in-the-head.” Non-blackbox use of PRG
- Today: NISC/OT using PRG as a black-box
NISC Results

- NISC considered in [RAD’78,Y’86,SYY’99,...]
- NISC for general (poly-time computable) functions:
  - Honest-but-curious players:
    - NISC/OT using Yao’s garbled circuit [Y’86]
    - NISC from fully homomorphic encryption [RAD’78,G’09,GHV’10,...]: Low communication (but currently less practical); uses more than PRG+OT
  - Malicious players:
    - Use a NIZK to prove correctness of messages sent [CCKM’00,HK’07]: Expensive, and non-blackbox use of PRG (used for encryption in Yao)
    - [IPS’08 (full version)]: using “MPC-in-the-head.” Non-blackbox use of PRG
    - Today: NISC/OT using PRG as a black-box

Wide Open: Statistically secure NISC/OT (even constant round MPC) possible for general functions?

Open for honest-majority and honest-but-curious settings too
Our Results
Our Results

- Today: NISC/OT using PRG as a black-box
- Also, few PRG calls: polylog(κ) per gate of the function’s (large) circuit (previously Ω(κ) even for interactive constant-round SFE [LP’07])
- A relaxed security notion allows constant number of PRG calls per gate
Our Results

- **Today:** NISC/OT using PRG as a black-box
  - Also, few PRG calls: polylog(\(\kappa\)) per gate of the function's (large) circuit (previously \(\Omega(\kappa)\) even for interactive constant-round SFE [LP’07])
  - A relaxed security notion allows constant number of PRG calls per gate
- Also, **Reusable NISC in CRS model** (using PRG + OT protocol): One reusable “public-key” that Alice publishes and uses in many executions.
Our Results

Today: NISC/OT using PRG as a black-box

- Also, few PRG calls: polylog(\(\kappa\)) per gate of the function’s (large) circuit (previously \(\Omega(\kappa)\) even for interactive constant-round SFE [LP’07])

- A relaxed security notion allows constant number of PRG calls per gate

Also, Reusable NISC in CRS model (using PRG + OT protocol): One reusable “public-key” that Alice publishes and uses in many executions.

- Issue: public-key must be refreshed each time Alice interacts with the environment (possibly after receiving messages from many Bobs)

- We show how to allow \(t\) such interactions before refreshing, with public-key much shorter than \(t\) times the original
Input-Dependent Abort
Input-Dependent Abort

- Intermediate security notions (also useful by themselves) [K’88,LP’07,...]
Input-Dependent Abort

- Intermediate security notions (also useful by themselves) \([K'88,LP'07,...]\)
- General input-dependent abort security: Corrupt Bob can enforce that Alice aborts the protocol iff \(P(x)=1\) for a predicate \(P\) he specifies
Input-Dependent Abort

Intermediate security notions (also useful by themselves) [K’88,LP’07,...]

General input-dependent abort security: Corrupt Bob can enforce that Alice aborts the protocol iff $P(x)=1$ for a predicate $P$ he specifies

- **Input Value Disjunction (IVD) predicate:** $P_S(x)=1$ iff $x_i=b_i$ for some $(i,b_i) \in S$

- **Wire Value Disjunction (WVD) predicate:** $P_{T,y}(x)=1$ iff in circuit $C(x,y)$, wire $w$ has value $b_w$ for some $(w,b_w) \in T$
Input-Dependent Abort

Intermediate security notions (also useful by themselves) [K’88,LP’07,...]

General input-dependent abort security: Corrupt Bob can enforce that Alice aborts the protocol iff $P(x)=1$ for a predicate $P$ he specifies

- **Input Value Disjunction (IVD) predicate:**
  
  $P_S(x)=1$ iff $x_i=b_i$ for some $(i,b_i) \in S$

- **Wire Value Disjunction (WVD) predicate:**
  
  $P_{T,y}(x)=1$ iff in circuit $C(x,y)$, wire $w$ has value $b_w$ for some $(w,b_w) \in T$

Maybe good enough in practice: leaks at most one bit (or less, if Alice aggregates many executions before taking any action) of information about Alice’s input
Roadmap for NISC/OT
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC$^0$ functions (with IVD-abort)
Roadmap for NISC/OT

Using Yao’s garbled circuit and Oblivious MAC

- NISC for NC^0 with IVD-abort
- Lean NISC/NC^0 with inp.dep-abort
- NISC/NC^0 using cut&choose
- NISC/NC^0 with WVD-abort

**Step 1**: NISC/OT for NC^0 functions (with IVD-abort)

**Step 2**: NISC/H for NC^0 function H. Use H to compile Yao’s garbled circuit. (Three variants.)
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC⁰ functions (with IVD-abort)
- **Step 2**: NISC/H for NC⁰ function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC⁰ into NISC/NC⁰ schemes

Using Yao’s garbled circuit and Oblivious MAC
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC⁰ functions (with IVD-abort)
- **Step 2**: NISC/H for NC⁰ function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC⁰ into NISC/NC⁰ schemes
- **Step 4**: Handle IVD/WVD-aborts
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC⁰ functions (with IVD-abort)
- **Step 2**: NISC/H for NC⁰ function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC⁰ into NISC/NC⁰ schemes
- **Step 4**: Handle IVD/WVD-aborts

Using Yao’s garbled circuit and Oblivious MAC

NISC for NC⁰ with IVD-abort

Lean NISC/NC⁰ with inp.dep-abort

NISC/NC⁰ using cut&choose

NISC/NC⁰ with WVD-abort

NISC with inp.dep-abort

NISC with IVD-abort

NISC with WVD-abort

Input-encoding

NISC
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC$^0$ functions (with IVD-abort)
- **Step 2**: NISC/H for NC$^0$ function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC$^0$ into NISC/NC$^0$ schemes
- **Step 4**: Handle IVD/WVD-aborts
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC^0 functions (with IVD-abort)
- **Step 2**: NISC/H for NC^0 function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC^0 into NISC/NC^0 schemes
- **Step 4**: Handle IVD/WVD-aborts

Using Yao’s garbled circuit and Oblivious MAC
Roadmap for NISC/OT

- **Step 1**: NISC/OT for NC⁰ functions (with IVD-abort)
- **Step 2**: NISC/H for NC⁰ function H. Use H to compile Yao’s garbled circuit. (Three variants.)
- **Step 3**: Plug-in NISC/OT for NC⁰ into NISC/NC⁰ schemes
- **Step 4**: Handle IVD/WVD-aborts

**C**: polylog(κ) factor more comm./PRG calls over Yao

For smaller circuits, B may be better
Roadmap for NISC/OT

- Step 1: NISC/OT for NC$^0$ functions (with IVD-abort)
- Step 2: NISC/H for NC$^0$ function $H$. Use $H$ to compile Yao's garbled circuit. (Three variants.)
- Step 3: Plug-in NISC/OT for NC$^0$ into NISC/NC$^0$ schemes
- Step 4: Handle IVD/WVD-aborts

- C: polylog($\kappa$) factor more comm./PRG calls over Yao
  - For smaller circuits, B may be better

- All use PRG in a black-box manner (like Yao)
Step 1

Using Yao's garbled circuit and Oblivious MAC

- NISC for NC^0 with IVD-abort
- Lean NISC/NC^0 with inp.dep-abort
- NISC/NC^0 using cut&choose
- NISC/NC^0 with WVD-abort

NISC with inp.dep-abort
NISC with IVD-abort
NISC with WVD-abort

Input-encoding

Private circuits

NISC
Step 1

- NISC for NC$^0$ with IVD-abort
- Lean NISC/NC$^0$ with inp.dep-abort
- NISC/NC$^0$ using cut&choose
- NISC/NC$^0$ with WVD-abort

Using Yao’s garbled circuit and Oblivious MAC

- NISC with inp.dep-abort
- NISC with IVD-abort
- NISC with WVD-abort

Input-encoding

Private circuits

NISC
For NC^0 functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient.
For NC⁰ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient.
Step 1

For $\text{NC}^0$ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient

First build NISC/OT with IVD-abort for “certified-OT”
Step 1

- For $NC^0$ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient
- First build NISC/OT with IVD-abort for "certified-OT"
- Implemented using "MPC-in-the-head" (a la [IKOS’07])
For $\text{NC}^0$ functions, unconditionally secure $\text{NISC/OT}$ [Kilian’88, IPS’08], but not very efficient

First build $\text{NISC/OT}$ with IVD-abort for “certified-OT”

Implemented using “MPC-in-the-head” (a la [IKOS’07])

Uses MPC protocol from [DIK’10] to keep views small
Step 1

- Implemented using “MPC-in-the-head” (a la [IKOS’07])
- Uses MPC protocol from [DIK’10] to keep views small

For $NC^0$ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient

First build NISC/OT with IVD-abort for “certified-OT”

Using Yao’s garbled circuit and Oblivious MAC

- Lean NISC/$NC^0$ with inp.dep-abort
- NISC/$NC^0$ using cut&choose
- NISC/$NC^0$ with WVD-abort

NISC for $cert$-OT with IVD-abort
NISC for $NC^0$ with IVD-abort
NISC with inp.dep-abort
NISC with IVD-abort
NISC with WVD-abort

O($\kappa$) overhead
polylog($\kappa$) overhead

Input-encoding
Private circuits

MPC
MPC in the head
Honest-Majority MPC
Lean NISC/NC$^0$ with inp.dep-abort
NISC/NC$^0$ using cut&choose
NISC/NC$^0$ with WVD-abort
NISC/NC$^0$ for $cert$-OT with IVD-abort
NISC for $NC^0$ with IVD-abort
NISC with inp.dep-abort
NISC with IVD-abort
NISC with WVD-abort
NISC
Step 1

For $NC^0$ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient

First build NISC/OT with IVD-abort for “certified-OT”
For $\text{NC}^0$ functions, unconditionally secure NISC/OT [Kilian’88,IPS’08], but not very efficient

First build NISC/OT with IVD-abort for “certified-OT”

Use certified-OT (instead of OT) in perfectly secure Yao’s garbled secure protocol for $\text{NC}^0$ functions
Step 2

- Honest-Majority MPC
- Semi-honest perfectly secure NISC for NC$^0$
  - NISC for cert-OT with IVD-abort
  - NISC for NC$^0$ with IVD-abort
    - Lean NISC/NC$^0$ with inp.dep-abort
    - NISC/NC$^0$ using cut&choose
    - NISC/NC$^0$ with WVD-abort
    - NISC with inp.dep-abort
    - NISC with IVD-abort
    - NISC with WVD-abort

Using Yao’s garbled circuit and Oblivious MAC

- MPC in the head
- NISC for cert-OT with IVD-abort
- Input-encoding
- Private circuits
- NISC
All variants rely on Yao’s Garbled Circuit and “oblivious MAC”
Yao’s Garbled Circuit
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.
Yao’s Garbled Circuit

- Bob sends a garbled circuit to Alice
- Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.

For each gate $G$, and each pair of masked values $(a’, b’)$ of inputs to the gate, let $c’$ be the masked output.
Yao's Garbled Circuit

- Bob sends a garbled circuit to Alice
- Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice's input wires and the output wires.
- For each gate $G$, and each pair of masked values $(a', b')$ of inputs to the gate, let $c'$ be the masked output.
Yao's Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice's input wires and the output wires.

For each gate $G$, and each pair of masked values $(a', b')$ of inputs to the gate, let $c'$ be the masked output
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire \( w \) has a secret mask \( r_w \), and two encryption keys \( K_w(0) \) and \( K_w(1) \). \( r_w = 0 \) for Alice’s input wires and the output wires.

For each gate \( G \), and each pair of masked values \( (a', b') \) of inputs to the gate, let \( c' \) be the masked output

Store \( [c', K_w(c')]_{G, a', b'} : (c', K_w(c')) \) encrypted using \( K_u(a') \) & \( K_v(b') \)

<table>
<thead>
<tr>
<th>Gate</th>
<th>a'</th>
<th>b'</th>
<th>c' = F_G(a, b')</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.

For each gate $G$, and each pair of masked values $(a',b')$ of inputs to the gate, let $c'$ be the masked output.

Store $[c',K_w(c')]_{G,a',b'} : (c',K_w(c'))$ encrypted using $K_u(a') \& K_v(b')$

Alice can evaluate the circuit if $(z_w',K_w(z_w'))$ known for all input wires $w$, with value $z_w$
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

Each wire w has a secret mask \( r_w \), and two encryption keys \( K_w(0) \) and \( K_w(1) \). \( r_w = 0 \) for Alice’s input wires and the output wires.

For each gate \( G \), and each pair of masked values \((a', b')\) of inputs to the gate, let \( c' \) be the masked output

\[ [c', K_w(c')]_{G, a', b'} : (c', K_w(c')) \text{ encrypted using } K_u(a') & K_v(b') \]

Alice can evaluate the circuit if \((z_w', K_w(z_w'))\) known for all input wires \( w \), with value \( z_w \)

Bob sends \((y_w', K_w(y_w'))\) for his input wires

<table>
<thead>
<tr>
<th>Gate</th>
<th>a'</th>
<th>b'</th>
<th>c' = F_G(a, b')</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

...
Bob sends a garbled circuit to Alice

Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.

For each gate $G$, and each pair of masked values $(a', b')$ of inputs to the gate, let $c'$ be the masked output.

Store $[c', K_w(c')]_{G, a', b'} : (c', K_w(c'))$ encrypted using $K_u(a')$ & $K_v(b')$

Alice can evaluate the circuit if $(z_w', K_w(z_w'))$ known for all input wires $w$, with value $z_w$

Bob sends $(y_w', K_w(y_w'))$ for his input wires

Alice picks up $K_w(x_w)$ for her input wires using OT
Yao’s Garbled Circuit

Bob sends a garbled circuit to Alice

- Each wire $w$ has a secret mask $r_w$, and two encryption keys $K_w(0)$ and $K_w(1)$. $r_w = 0$ for Alice’s input wires and the output wires.

- For each gate $G$, and each pair of masked values $(a', b')$ of inputs to the gate, let $c'$ be the masked output.

- Store $[c', K_w(c')]_{G, a', b'} : (c', K_w(c'))$ encrypted using $K_u(a') & K_v(b')$

Alice can evaluate the circuit if $(z_w, K_w(z_w'))$ known for all input wires $w$, with value $z_w$

- Bob sends $(y_w, K_w(y_w'))$ for his input wires

- Alice picks up $K_w(x_w)$ for her input wires using OT

<table>
<thead>
<tr>
<th>Gate</th>
<th>$a'$</th>
<th>$b'$</th>
<th>$c'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Oblivious MAC (OM)

Verifier gets a receipt that can be used to verify the MAC when the message and tag are delivered
Oblivious MAC (OM)

Verifier gets a receipt that can be used to verify the MAC when the message and tag are delivered.

**Obliviousness:** If tag chosen at random, then receipt by itself reveals no information about msg.

**Correctness:** Verify(msg, tag; key, rcpt) = 1

**Unforgeability:** can’t find (msg, tag) and (msg’, tag’) s.t. msg’ ≠ msg, and for rcpt=OM_key(msg, tag), Verify(msg’, tag’; key, rcpt) = 1
Oblivious MAC (OM)

Verifier gets a receipt that can be used to verify the MAC when the message and tag are delivered

**Obliviousness**: If tag chosen at random, then receipt by itself reveals no information about msg

**Correctness**: Verify(msg,tag;key,rcpt) = 1

**Unforgeability**: can’t find (msg,tag) and (msg’,tag’) s.t. msg’ ≠ msg, and for rcpt=OM_key(msg,tag), Verify(msg’,tag’;key,rcpt) = 1

Concretely implemented using a **one-time** (statistically) secure MAC
Oblivious MAC (OM)

Verifier gets a receipt that can be used to verify the MAC when the message and tag are delivered

\[ \text{rcpt} = \text{MAC}_{\text{key}}(\text{msg}) \oplus \text{tag} \]

- **Obliviousness**: If tag chosen at random, then receipt by itself reveals no information about msg
- **Correctness**: Verify(msg, tag; key, rcpt) = 1
- **Unforgeability**: can’t find (msg, tag) and (msg’, tag’) s.t. msg’ \( \neq \) msg, and for \( \text{rcpt} = \text{OM}_{\text{key}}(\text{msg}, \text{tag}) \), Verify(msg’, tag’; key, rcpt) = 1
- Concretely implemented using a **one-time** (statistically) secure MAC
A Lean NISC/NC\(^0\)
With Input-Dependent Abort Security
A Lean NISC/NC^0
With Input-Dependent Abort Security

NC^0 functionality H:
A Lean NISC/NC$^0$

With Input-Dependent Abort Security

NC$^0$ functionality $H$:
- Takes from Bob the wire masks and computes the bit $c'$ in each garbled gate $(G,a',b',c')$

\[ c' = F_G(a' \oplus r_u, b' \oplus r_v) \oplus r_w \]
A Lean NISC/NC$^0$

With Input-Dependent Abort Security

**NC$^0$ functionality H:**
- Takes from Bob the wire masks and computes the bit $c'$ in each garbled gate $(G,a',b',c')$
- For each $(G,a',b')$ carries out OM for the bit $c'$ (using independent keys from Alice)

\[ c' = F_G(a' \oplus r_u, b' \oplus r_v) \oplus r_w \]

\[ \text{rcpt } = \text{MAC}_{\text{key}}(c') \oplus \text{tag} \]
A Lean NISC/NC⁰

With Input-Dependent Abort Security

NC⁰ functionality H:

- Takes from Bob the wire masks and computes the bit c’ in each garbled gate (G,a’,b’,c’)
- For each (G,a’,b’) carries out OM for the bit c’ (using independent keys from Alice)
- Also, as in Yao’s scheme, lets Alice pick up her input wires’ keys

\[ c’ = F_G(a’ \oplus r_u, b’ \oplus r_v) \oplus r_w \]

\[ \text{rcpt} = \text{MAC}_{\text{key}}(c’) \oplus \text{tag} \]

OT for input-keys
A Lean NISC/NC^0
With Input-Dependent Abort Security

NC^0 functionality H:
- Takes from Bob the wire masks and computes the bit c' in each garbled gate (G,a',b',c')
- For each (G,a',b') carries out OM for the bit c' (using independent keys from Alice)
- Also, as in Yao’s scheme, lets Alice pick up her input wires’ keys
- Bob sends the garbled circuit to Alice, with encryptions [c',K_w(c'),tag]_{G,a',b'}

\[ c' = F_G(a' \oplus r_u, b' \oplus r_v) \oplus r_w \]

\[ \text{rcpt} = \text{MAC}_{\text{key}}(c') \oplus \text{tag} \]

OT for input-keys

Garbled circuit (with encrypted tags)

Input keys

OM-key

rcpt

r_u, r_v, r_w, ...

Tag_{G,a',b'}

Key pairs

Input
A Lean NISC/NC$^0$

With Input-Dependent Abort Security

NC$^0$ functionality $H$:
- Takes from Bob the wire masks and computes the bit $c'$ in each garbled gate $(G,a',b',c')$
- For each $(G,a',b')$ carries out OM for the bit $c'$ (using independent keys from Alice)
- Also, as in Yao's scheme, lets Alice pick up her input wires' keys
- Bob sends the garbled circuit to Alice, with encryptions $[c',K_w(c'),\text{tag}]_{G,a',b'}$
A Lean NISC/NC⁰

With Input-Dependent Abort Security

NC⁰ functionality H:
- Takes from Bob the wire masks and computes the bit c’ in each garbled gate (G,a’,b’,c’)
- For each (G,a’,b’) carries out OM for the bit c’ (using independent keys from Alice)
- Also, as in Yao’s scheme, lets Alice pick up her input wires’ keys
- Bob sends the garbled circuit to Alice, with encryptions [c’,K_w(c’),tag]_{G,a’,b’}
- Prevents Bob from changing c’ that Alice obtains by decrypting, but Bob can cause Alice to abort (if a wrong c’ or tag is kept encrypted)
A Lean NISC/NC$^0$

With Input-Dependent Abort Security

NC$^0$ functionality $H$:
- Takes from Bob the wire masks and computes the bit $c'$ in each garbled gate $(G,a',b',c')$
- For each $(G,a',b')$ carries out OM for the bit $c'$ (using independent keys from Alice)
- Also, as in Yao's scheme, lets Alice pick up her input wires' keys

Bob sends the garbled circuit to Alice, with encryptions $[c',K_w(c'),\text{tag}]_{G,a',b'}$

Prevents Bob from changing $c'$ that Alice obtains by decrypting, but Bob can cause Alice to abort (if a wrong $c'$ or tag is kept encrypted)

But if no abort, then throughout the evaluation, for each $(G,a',b')$ the $c'$ value recovered is correct, and hence output is correct
A Lean NISC/NC⁰

With Input-Dependent Abort Security

NC⁰ functionality H:
- Takes from Bob the wire masks and computes the bit c’ in each garbled gate (G,a’,b’,c’)
- For each (G,a’,b’) carries out OM for the bit c’ (using independent keys from Alice)
- Also, as in Yao’s scheme, lets Alice pick up her input wires’ keys

Bob sends the garbled circuit to Alice, with encryptions [c’,KW(c’),tag]_{G,a’,b’}

Prevents Bob from changing c’ that Alice obtains by decrypting, but Bob can cause Alice to abort (if a wrong c’ or tag is kept encrypted)

But if no abort, then throughout the evaluation, for each (G,a’,b’) the c’ value recovered is correct, and hence output is correct

Input-dependent abort security: since abort can depend on the inputs in a fairly complicated way
Step 2

- Semi-honest perfectly secure NISC for NC^0
- NISC for cert-OT with IVD-abort
- NISC for NC^0 with IVD-abort
- Using Yao's garbled circuit and Oblivious MAC
  - Lean NISC/NC^0 with inp.dep-abort
  - NISC/NC^0 using cut&choose
  - NISC/NC^0 with WVD-abort
- NISC with inp.dep-abort
- NISC with IVD-abort
- NISC with WVD-abort
- Input-encoding
- Private circuits
- NISC

Honest-Majority MPC
MPC in the head
To restrict to WVD-abort, more complex NC$^0$ function

1. Honest-Majority MPC
2. Semi-honestly perfectly secure NISC for NC$^0$
3. NISC for cert-OT with IVD-abort
4. NISC for NC$^0$ with IVD-abort
5. Lean NISC/NC$^0$ with inp.dep-abort
6. NISC/NC$^0$ using cut&choose
7. NISC/NC$^0$ with WVD-abort
8. NISC with inp.dep-abort
9. NISC with IVD-abort
10. NISC with WVD-abort

Using Yao’s garbled circuit and Oblivious MAC

MPC in the head

Private circuits

Input-encoding
To restrict to WVD-abort, more complex NC\(^0\) function
- creates the garbled circuit, using purported PRG values given by Bob
- applies OM to those PRG values: using “NC\(^0\) MAC” [IKOS’08]
To restrict to WVD-abort, more complex NC$^0$ function
- creates the garbled circuit, using purported PRG values given by Bob
- applies OM to those PRG values: using “NC$^0$ MAC” [IKOS’08]
- WVD-abort as wrong PRF values can be given for certain keys and not others

Using Yao’s garbled circuit and Oblivious MAC

NISC for NC$^0$ with IVD-abort
Lean NISC/NC$^0$ with inp.dep-abort
NISC/NC$^0$ using cut&choose
NISC/NC$^0$ with WVD-abort

NISC with inp.dep-abort
NISC with IVD-abort
NISC with WVD-abort

NISC with WVD-abort
NISC with IVD-abort
Lean NISC/NC$^0$ with inp.dep-abort
NISC/NC$^0$ using cut&choose
NISC/NC$^0$ with WVD-abort

Honest-Majority MPC
Semi-honest perfectly secure NISC for NC$^0$
MPC in the head

Input-encoding
Private circuits
To restrict to WVD-abort, more complex $\text{NC}^0$ function

- creates the garbled circuit, using purported PRG values given by Bob
- applies OM to those PRG values: using "NC$^0$ MAC" [IKOS'08]
- WVD-abort as wrong PRF values can be given for certain keys and not others
- Can avoid WVD-abort using cut&choose, but $O(\kappa)$ overhead
Step 3

Semi-honest perfectly secure NISC for NC^0

NISC for cert-OT with IVD-abort

NISC for NC^0 with IVD-abort

Using Yao’s garbled circuit and Oblivious MAC

Lean NISC/NC^0 with inp.dep-abort

NISC/NC^0 using cut&choose

NISC/NC^0 with WVD-abort

NISC with inp.dep-abort

NISC with IVD-abort

NISC with WVD-abort

Input-encoding

Private circuits

MPC in the head

NISC
Step 3

Plug-in NISC with IVD-abort security into the NISC/NC\(^0\) constructions

- Semi-honest perfectly secure NISC for NC\(^0\)
- MPC in the head
- NISC for cert-OT with IVD-abort
- NISC for NC\(^0\) with IVD-abort
- Lean NISC/NC\(^0\) with inp.dep-abort
- NISC/NC\(^0\) using cut&choose
- NISC/NC\(^0\) with WVD-abort
- NISC with inp.dep-abort
- NISC with IVD-abort
- NISC with WVD-abort

Using Yao’s garbled circuit and Oblivious MAC

Input-encoding

Private circuits

NISC
Step 4

- Semi-honest perfectly secure NISC for NC⁰
  - NISC for cert-OT with IVD-abort
  - NISC for NC⁰ with IVD-abort

- MPC in the head

- Using Yao's garbled circuit and Oblivious MAC
  - Lean NISC/NC⁰ with inp.dep-abort
  - NISC/NC⁰ using cut&choose
  - NISC/NC⁰ with WVD-abort

- Input-encoding
  - NISC with inp.dep-abort
  - NISC with IVD-abort
  - NISC with WVD-abort

- Private circuits

- NISC
Step 4

- Semi-honest perfectly secure NISC for NC<sup>0</sup>
- IVD-abort handled by using a κ-wise independent encoding of x

Using Yao’s garbled circuit and Oblivious MAC

Lean NISC/NC<sup>0</sup> with inp.dep-abort

NISC/NC<sup>0</sup> using cut&choose

NISC/NC<sup>0</sup> with WVD-abort

NISC with inp.dep-abort

NISC with IVD-abort

NISC with WVD-abort

MPC in the head

Private circuits

Input-encoding

NISC
Step 4

- IVD-abort handled by using a $\kappa$-wise independent encoding of $x$
- WVD-abort handled by using a “private-circuit” encoding of $C$ and input $x$
Step 4

- Semi-honest perfectly secure NISC for NC^0

- NISC for cert-OT with IVD-abort

- NISC for NC^0 with IVD-abort

- Lean NISC/NC^0 with inp.dep-abort

- NISC/NC^0 using cut&choose

- NISC/NC^0 with WVD-abort

- NISC with inp.dep-abort

- NISC with IVD-abort

- NISC with WVD-abort

- Using Yao's garbled circuit and Oblivious MAC

- MPC in the head

- IVD-abort handled by using a \( \kappa \)-wise independent encoding of \( x \)

- WVD-abort handled by using a "private-circuit" encoding of \( C \) and input \( x \)

- Reading up to \( \kappa \) wires gives no information about \( x \)
IVD-abort handled by using a $\kappa$-wise independent encoding of $x$

WVD-abort handled by using a “private-circuit” encoding of $C$ and input $x$

Reading up to $\kappa$ wires gives no information about $x$

Private-circuit constructed from MPC (a la [ISW'03])
IVD-abort handled by using a $\kappa$-wise independent encoding of $x$
WVD-abort handled by using a “private-circuit” encoding of $C$ and input $x$
- Reading up to $\kappa$ wires gives no information about $x$
- Private-circuit constructed from MPC (a la [ISW’03])
- Size of private-circuit proportional to “work” in the MPC. polylog($\kappa$) overhead using MPC in [DIK’10]
Summary

- Semi-honest perfectly secure NISC for NC₀
  - NISC for cert-OT with IVD-abort
    - MPC in the head
  - NISC for NC₀ with IVD-abort
  - Lean NISC/NC₀ with inp.dep-abort
    - NISC/NC₀ using cut&choose
    - NISC with inp.dep-abort
    - NISC/NC₀ using cut&choose
  - NISC/NC₀ with WVD-abort
  - NISC with IVD-abort
  - NISC/NC₀ with WVD-abort
    - NISC

Using Yao's garbled circuit and Oblivious MAC

Input-encoding

Private circuits
Semi-honest perfectly secure NISC for NC^0

NISC for cert-OT with IVD-abort

MPC in the head

NISC for NC^0 with IVD-abort

Using Yao's garbled circuit and Oblivious MAC

NISC/NC^0 with inp.dep-abort

Lean NISC/NC^0 with inp.dep-abort

NISC/NC^0 using cut&choose

NISC/NC^0 with WVD-abort

NISC with inp.dep-abort

NISC with IVD-abort

NISC with WVD-abort

Input-encoding

Private circuits

NISC/OT using blackbox PRG

NISC
NISC/OT using blackbox PRG

Few PRG calls, low communication overhead over Yao
NISC/OT using blackbox PRG

- Few PRG calls, low communication overhead over Yao

Open questions: Better efficiency (in OT usage, in reusability, ...)? Statistical security?