Randomization Techniques for Secure Computation and Parallel Cryptography

Eyal Kushilevitz (Technion)

Course based on joint works with
Yuval Ishai and Benny Applebaum
(FOCS 00, ICALP 02, FOCS 04, CCC 05, RANDOM 06, CRYPTO 07)
The Basic Question

- **g** is a “randomized encoding” of **f**
  - Nontrivial relaxation of computing **f**
- **Hope:**
  - **g** can be “simpler” than **f**
    (meaning of “simpler” determined by application)
  - **g** can be used as a substitute for **f**
  - **g** inherits security properties of **f**

\[
Enc(y) \rightarrow y \\
\sim \rightarrow \sim \\
x \rightarrow y \\
x \rightarrow y \\
r \rightarrow y \\
Enc(y) \rightarrow x \\
Dec(g(x,r)) = f(x) \\
Sim(f(x)) = g(x,r)
\]
Applications at a Glance

Randomized encodings

- Secure computation
- Parallel cryptography

Hardness of approximation
Randomized Encoding - Syntax

\[ f(x) \text{ is encoded by } g(x,r) \]
Randomized Encoding - Semantics

- Correctness: \( f(x) \) can be efficiently decoded from \( g(x,r) \).
  \[ f(x) \neq f(w) \implies \]

- Privacy: \( \exists \) efficient simulator \( \text{Sim} \) such that \( \text{Sim}(f(x)) \equiv g(x,U) \)
  - \( g(x,U) \) depends only on \( f(x) \)
  \[ f(x) = f(w) \implies \]
Notions of Simplicity - I

- **Application:** “minimal model for secure computation” [Feige-Kilian-Naor 94, …]

- **2-decomposability:** \( g((x_A,x_B),r) = (g_A(x_A,r), g_B(x_B,r)) \)
Example: sum

- \( f(x_A, x_B) = x_A + x_B \) \( (x_A, x_B \in \text{finite group } G) \)
Example: equality

- \( f(x_A, x_B) = \text{equality} \quad (x_A, x_B \in \text{finite field } F) \)

\[
\begin{align*}
r_1 &\in \mathbb{R} F \setminus \{0\}, \quad r_2 \in \mathbb{R} F \\
x_A &
\end{align*}
\]

Alice

\[
\begin{align*}
r_1 x_A + r_2 &
\end{align*}
\]

Bob

\[
\begin{align*}
r_1 x_B + r_2 &
\end{align*}
\]

Carol

\[ m_A = m_B ? \]
Example: ANY function

- $f(x_A, x_B) = x_A \land x_B$ ($x_A, x_B \in \{0, 1\}$)
  - Reduction to equality: $x_A \Rightarrow 0/1$, $x_B \Rightarrow 2/1$

- **General boolean $f$: write as *disjoint* 2-DNF

  - $f(x_A, x_B) = \bigvee_{(a,b):f(a,b)=1} (x_A=a \land x_B=b) = t_1 \lor t_2 \lor \ldots \lor t_m$

Exponential complexity
Notions of Simplicity - II

- Full decomposability:
  \[ g((x_1,\ldots,x_n),r) = (g_1(x_1,r),\ldots,g_n(x_n,r)) \]
  - Application: Basing SFE on OT [Kilian 88, ...]

\[ \text{Alice} \]

\[ g_A(x_B, r) \]

\[ \text{Bob} \]

\[ f(x_A, x_B) \]

\[ g_n(0, r) \quad g_n(1, r) \]

\[ x_n \quad g_n(x_n, r) \]
Example: iterated group product

• Abelian case
  – \( f(x_1, \ldots, x_n) = x_1 + x_2 + \ldots + x_n \)
  – \( g(x, (r_1, \ldots, r_{n-1})) = \)

\[
\begin{align*}
x_1 + r_1 & \quad x_2 + r_2 & \quad \ldots & \quad x_{n-1} + r_{n-1} & \quad x_n - r_1 - \ldots - r_{n-1}
\end{align*}
\]

• General case  [Kilian 88]
  – \( f(x_1, \ldots, x_n) = x_1 x_2 \ldots x_n \)
  – \( g(x, (r_1, \ldots, r_{n-1})) = \)

\[
\begin{align*}
x_1 r_1 & \quad r_1^{-1} x_2 r_2 & \quad r_2^{-1} x_2 r_3 & \quad \ldots & \quad r_{n-2}^{-1} x_{n-1} r_{n-1} & \quad r_{n-1}^{-1} x_n
\end{align*}
\]
Example: iterated group product

Thm [Barrington 86]
Every boolean \( f \in \text{NC}^1 \) can be computed by a poly-length, width-5 branching program.

\[
f(x_1, \ldots, x_n) \text{ reduces to } \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_m \text{ where:}
\]

- Each \( \pi_i \) depends on a single \( x_j \)
- \( \exists \) distinct \( \sigma_0, \sigma_1 \in S_5 \) s.t. \( \pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_m = \sigma_f(x) \)

Encoding iterated group product
- Every output bit of \( g \) depends on just a single bit of \( x \)
  - Efficient fully decomposable encoding for every \( f \in \text{NC}^1 \)
Notions of Simplicity - III

• **Low degree:** $g(x,r) = \text{vector of degree-d poly in } x,r \text{ over } F$
  – aka “Randomizing Polynomials” [IK00,…]
  – Application: round-efficient MPC

• **Motivating observation:**
  Low-degree functions are easy to distribute!
  – Round complexity of MPC protocols
  [BGW88,CCD88,CDM00,…]
    • Semi-honest model
      – $t<n/d \Rightarrow 2 \text{ rounds}$
      – $t<n/2 \Rightarrow \text{ multiplicative depth } + 1 = \lceil \log d \rceil + 1 \text{ rounds}$
    • Malicious model
      – Optimal $t \Rightarrow O(\log d) \text{ rounds}$
Examples

• What’s wrong with previous examples?
  – Great degree in $x$ (deg$_x$=1), bad degree in $r$

• Coming up:
  – Degree-3 encoding for every $f$
  – Efficient in size of branching program
Notions of Simplicity - IV

• Small locality:

- Application: parallel cryptography!
  \[\text{[AIK04,AIK05,AIK07…]}\]

• Coming up: encodings with locality 4
  - degree 3, fully decomposable
  - efficient in size of branching program
Parallel Cryptography

How low can we get?

<table>
<thead>
<tr>
<th>Complexity Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly-time</td>
</tr>
<tr>
<td>NC</td>
</tr>
<tr>
<td>log-space</td>
</tr>
<tr>
<td>NC^1</td>
</tr>
<tr>
<td>AC^0</td>
</tr>
<tr>
<td>NC^0</td>
</tr>
</tbody>
</table>
“Simplicity” of Cryptographic Primitives

• Can cryptographic primitives be computed by very simple functions?

\[ \text{Simple} = \text{each output bit depends on } O(1) \text{ input bits} \]
\[ = \text{const. depth circuits with bounded fan-in} \]
\[ = \text{NC}^0 \]

• Currently the smallest creature in the complexity zoo
Cryptography in NC$^0$?

- Longstanding open question
  - Håstad 87
  - Impagliazzo Naor 89
  - Goldreich 00
  - Cryan Miltersen 01
  - Krause Lucks 01
  - Mossel Shpilka Trevisan 03

- Real-life motivation: super-fast cryptographic hardware

- Tempting conjecture:
  - crypto hardness
  - [CM]: Yes
  - [G]: No
  - “complex” function
Basic Primitives:
One-way Function (OWF)
Basic Primitives:
Pseudorandom Generator (PRG)

Def. PRG is minimal if stretch=1
Previous Work

• Positive results
  – OWF in NC^0 [Goldreich 00, CryanMiltersen 01]
  – PRF in NC^1 from factoring [NaorReingold 97]
  – PRG (sub-lin stretch) in AC^0 from subset sum [ImpagliazzoNaor 89]
• Permutation in NC^0 which is P-complete to invert [Håstad 87]
• Function in NC^0 which is NP-complete to invert [AgrawalAllenderRudich 98]
• Heuristic construction of OWF/PRG in NC^0 [Goldreich 00, MST 03]

- No OWF in NC^0 [Goldreich 00, Cryan Miltersen 01]
- No PRG with large stretch in NC^0, NC^1, NC^2, NC^3, NC^4 [CM01, MosselShpilkaTrevisan 03]

Previous work

- factoring, discrete-log, lattices, …
- subset sum
- impossible

PRG / OWF

low stretch

open
Our Approach

Compile primitives in a “relatively high” complexity class into ones in NC⁰.
Sufficient Assumptions for Crypto in NC⁰

Caveats:
- We get PRG with sub-linear stretch.
- Decryption/verification not in NC⁰.
  - In fact, impossible to decrypt/verify in NC⁰.
  - But: can commit in NC⁰ with decommit in NC⁰.

### Assumptions

<table>
<thead>
<tr>
<th></th>
<th>OWF</th>
<th>PRG</th>
<th>Hash</th>
<th>Sym-Enc</th>
<th>PK-Enc</th>
<th>Signature</th>
<th>Commit</th>
<th>NIZK</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC¹</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC⁰</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC⁰₄</td>
<td>OWF</td>
<td>PRG</td>
<td>Hash</td>
<td>Sym-Enc</td>
<td>PK-Enc</td>
<td>NI-Com</td>
<td>Sign</td>
<td>NIZK</td>
</tr>
</tbody>
</table>

Assuming min-PRG in NC¹

*factoring, discrete-log/DDH, lattices, …*

*\[\text{AIK 04}] \ [\text{AIK 05}]\*
Cryptography with Constant Input Locality

Till now we considered only $\text{NC}^0$ functions…

$\text{NC}^0 = \text{const. depth circuits with bounded fan-in} = \text{each output bit depends on } O(1) \text{ input bits}$

**Q:** Can cryptographic primitives be realized by functions in which each input bit affects a constant number of output bits?
Outline

1. (Long) Introduction
2. Randomized Polynomials (w/applications to round-efficient MPC)
3. Randomized Encodings (w/applications to NC⁰ Cryptography)
4. Constant Input Locality
5. Computational Randomized Encodings (w/applications)
6. NC⁰ Linear Stretch PRG (w/applications)