Fixing Cracks in the Concrete: Random Oracles with Auxiliary Input, Revisited

Yevgeniy Dodis
New York University

Joint work with Siyao Guo (Simons Institute, UC Berkeley)
Jonathan Katz (University of Maryland)
Hash Functions Are Ubiquitous

- OWFs
- PRG/PRFs
- MACs
- CRHFs
- KDFs
- ...

How to assess the best possible concrete security for each application?
Random Oracle Model Methodology

[BR93]

Random Function

\[ O : [N] \rightarrow [M] \]

A: \( T \) queries

Clean proofs, Precise bounds: e.g., OWFs/MACs

\( T/min(N,M) \), PRFs/PRGs \( T/N \), CRHFs \( T^2/M \), etc.

Simple Proof Techniques: programmability, lazy

sampling, distinguishing-to-extraction, etc.

**Practical heuristics:** for “natural” applications,

Security in ROM = Security in “standard model”

(for the best instantiation of \( O \))

- Theoretical counter-examples [CGH04, etc.]

“artificial” and don’t affect widely used applications
Random Oracle Model Methodology [BR93]

Practical heuristics: for “natural” applications,
Security in ROM = Security in “standard model”
(for the best instantiation of O)

• Theoretical counter-examples [CGH04, etc.]
  “artificial” and don’t affect widely used applications
Random Oracle Model Methodology [BR93]

Practical heuristics: for “natural” applications,
Security in ROM = Security in “standard model”
(for the best instantiation of O)

- Theoretical counter-examples [CGH04, etc.]
  “artificial” and don’t affect widely used applications
**Cracks in the Concrete**

**Random Oracle**

**Standard Model**

**CRHF:** \( T^2 / M \)  
1

**OWF:**  
\( \frac{T}{N} \)  
1

\( [N] \rightarrow [N] \)  
(rainbow tables; in time \( N^{2/3} \) [Hel80])

**PRG:**  
\( \frac{T}{N} \)  
\( \frac{1}{N^{1/2}} \)  
(constant time [AGHP92])
Non-uniform cracks in the Concrete

Random Oracle       Standard Model

CRHF: T^2/N   1

OWF: T/N   1
[N] => [N]
(rainbow tables; in time N^{2/3} [Hel80])

PRG: T/N   1/N^{1/2}
(constant time [AGHP92])
Non-Uniform Adversaries

• Modeled as families of circuits
  – Can “hardwire” arbitrary (bounded) “advice” before attacking the system
  – **Preprocessing**: special case of “computable” advice (corresponds to potentially implementable attack)

• Why/how did this become “standard” model?
  – Uniform model *too weak* (e.g., attacker can focus on a given security parameter in advance)
  – Sometimes **preprocessing realistic** (rainbow tables!)
  – Seems critical for protocol composition (i.e., ZK)
  – Wlog, **deterministic** attacker (P/poly=BPP/poly)
  – (Non-uniform) **hardness vs randomness**: non-uniform lower bounds => derandomization
Non-uniform Cracks in the Concrete

Can we “salvage” ROM methodology and be consistent with non-uniform attackers?
[Unr07] YES: ROM with Auxiliary-Input (ROM-AI)

The ROM methodology is blatantly false for most natural and widely deployed applications when:
• Preprocessing is allowed
• The standard model adversary is non-uniform


• Theoretical counter-examples [CGH04, etc.] “artificial” and don’t affect widely used applications
Fixing **Cracks** in the Concrete

**ROM-AI**

<table>
<thead>
<tr>
<th>Random function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O: [N] \rightarrow [M]$</td>
</tr>
</tbody>
</table>

$A = (A_0, A_1)$

- **$A_0$**: *computationally unbounded*, gets entire RO, and passes $S$ bits of $O$-dependent advice to $A_1$
  - Becomes *non-uniform* advice when $O$ instantiated
  - Separating $S$ and $T$ for more accurate *time-space tradeoffs* (e.g., for RAM attackers vs. circuits)

- **$A_1$**: $T$ queries

**ROM vs. standard model “separations” disappear!**

Concrete bounds in **ROM-AI** are meaningful against standard model **non-uniform** attackers!
Fixing **Cracks in the Concrete**

**ROM-AI**

Random function

\[ O: [N] \rightarrow [M] \]

\( A = (A_0, A_1) \)

\( A_0: S \text{ bits} \)

\( A_1: T \text{ queries} \)

**ROM-AI methodology:** for "natural" applications,

Security in **ROM-AI** = Security in "standard model"

against **non-uniform** attackers

(for the best instantiation of \( O \))

Security against any **preprocessing** attacks
Handling Auxiliary Input?

**Problem:** conditioned on S-bit “leakage”, values of random oracle are not random and independent

<table>
<thead>
<tr>
<th></th>
<th>Traditional ROM</th>
<th>ROM-AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy Sampling</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Programmability</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Distinguishing-to-Extraction</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Handling Auxiliary Input: Pre-sampling [Unruh07]

- **Intuition**: conditioned on $S$-bit leakage, only “few” values of $O(x)$ are “heavily biased”
  - $A_0$ can pass these “few” values as advice to $A_1$

- The rest can be re-sampled **fresh and independent of the leakage**!
  - Lazy sampling, programmability, etc. all come back as long as avoid the “few” pre-sampled points
Handling Auxiliary Input: Pre-sampling [Unruh07]

Random Function
\( O : [N] \rightarrow [M] \) \( \approx \epsilon \)

Random Function
\( R : [N] \rightarrow [M] \mid Pre^O \)

\( Pre^O = \{(x_1,y_1),..., (x_p,y_p)\} \)

\( S \) bits about \( O \), then \( T \) queries

\( y = O(x) \)

\( y = \begin{cases} 
  y_i, & \text{if } x = x_i \\
  R(x), & \text{o.w.} 
\end{cases} \)
Handling Auxiliary Input: Pre-sampling [Unruh07]

\[ \text{Random Function } O : [N] \rightarrow [M] \]

\[ \approx \varepsilon \]

\[ \text{Random Function } R : [N] \rightarrow [M] \mid Pre^O \]

- \( Pre^O \) can depend on \( S \)-bit “leakage” \( z \) about \( O \)
  - \( P \) is a free parameter optimized later (see below)
- But \( R \) is random and independent on \( z \)
- How big is \( \varepsilon \)?

\[ [\text{Unr}07] : \varepsilon < (ST/P)^{1/2} \]

\( S \) bits about \( O \), then \( T \) queries
Security of OWFs in ROM-AI

$S$ bits, $T$ queries

Random Function $O:[N] \rightarrow [N]$

$\approx \epsilon$

Random Function $R:[N] \rightarrow [N]$ | $\text{Pre}^O$

Advantage < $(ST/P)^{1/2} + P/N + T/(N-P)$

$P = (STN^2)^{1/3} \Rightarrow$ Advantage < $(ST/N)^{1/3} + T/N$

Does not match best generic attacks 😞 :

Advantage > $ST/N + T/N$ (if $ST^2 < N$)
Our Motivating Question

**Exact** security for **basic primitives**?
(critical for **ROM-AI methodology**)
Our Results

• Unruh’s "pre-sampling conjecture" false
  – For many apps (OWFs, MACs, etc.), pre-sampling (as defined above) cannot give tight bounds

• New technique: Compression
  – Apply to get nearly tight ROM-AI bounds for OWFs, MACs, PRGs, PRFs, (salted) CRHFs 😊
  – Bounds much weaker than traditional ROM 😞 (because there are non-uniform attacks!)

• Salting provably defeats preprocessing!
  – Long-enough salt ⇒ ROM-AI-sec. ≈ ROM-sec.
  – Possible way to reconcile theory and practice!
Our Results

- Salting provably defeats preprocessing!
  - Long-enough salt $\Rightarrow$ ROM-AI-sec. $\approx$ ROM-sec.
  - Possible way to reconcile theory and practice!
Improve Pre-sampling?

$S$ bits, $T$ queries

Random Function $O:[N] \rightarrow [N] \approx \varepsilon$

Random Function $R:[N] \rightarrow [N] \mid \text{Pre}^O$

- [Unr07]: $\varepsilon < (ST/P)^{1/2}$
  - Can’t get $\text{negl}(n)$ security with $P = \text{poly}(n)$ 😞
  - Conj: can get $\varepsilon = \text{negl}(n)$ for $P = \text{poly}(n)$ 😊

- Our result: $\varepsilon > \Omega(ST/P)$
  - Unruh’s conjecture false (in this generality)

- Is it enough to prove tight security?

$tight! [\text{CDGS17}]$
Security of OWF}s in ROM-AI

Random Function $O:[N] \rightarrow [N]$ with $S$ bits, $T$ queries

Random Function $R:[N] \rightarrow [N] \mid \text{Pre}^0$

$|\text{Pre}^0| = P$

$P = (STN^2)^{1/3}$

Advantage $< (ST/P)^{1/2}$ + $P/N + T/(N-P)$

Advantage $< (ST/N)^{1/3}$ + $T/N$

Does not match best generic attacks

Advantage $> ST/N + T/N$ (if $ST^2 < N$)

via "dream pre-sampling"
Our New Technique

Compression Paradigm \([GT00, DTT10]\)!

High advantage \(\Rightarrow\) Compressing RO

RO is impossible to compress

\(\Rightarrow\) **Exact** security bounds

**Challenge:** need to compress by more than \(S\) bits!
(salted) CRHFs

\[ \Pr[A_1^O(A_0(O), a) = (x, x') \text{ s.t. } x \neq x', O(a,x) = O(a,x')] = O(S/K + T^2/M) \]

**Optimal:** can hardwire \( S \) collisions inside advice \( A_0(O) \)!

**Idea:** compress \( O(a,x') \) using indices \( i,j \in [T] \) and \( O(a,x) \)

**# of saved bits:** \( |\# \text{ of } a \text{ s.t. } A \text{ succeeds}| \times (\log M - 2\log T) \)

\[ = \varepsilon K \times \log(M/T^2) \]

**# of spent bits:** \( S + \text{ description of set } \{a \mid A \text{ succeeds} \} = S + \log(K \text{ choose } \varepsilon K) \)

\[ \Rightarrow S > \varepsilon K \log(\varepsilon M/eT^2) \]

\[ \Rightarrow \varepsilon = O(S/K + T^2/M) \]
The Order Issue

Consider 2 salts: $O(a_1, x_1) = O(a_1, x_1')$; $O(a_2, x_2) = O(a_2, x_2')$

Ideally, compress both $O(a_1, x_1')$ and $O(a_2, x_2')$

**Problem:** what if $A(z, a_1)$ would query $O(a_2, x_2')$??

(not so crazy because of advice $z$...)

**Solution:** run $A$ on all salts $a$ where he succeeds, and keep track of “salt-specific” indices $i_a, j_a$ for the first collision (which exists!) on all such $a$’s.
# ROM-AI Bounds for Basic Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Our ROM-AI Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWF*</td>
<td>ST/N</td>
</tr>
<tr>
<td>PRG</td>
<td>$(ST/N)^{1/2}$</td>
</tr>
<tr>
<td>PRF</td>
<td>$(ST/N)^{1/2}$</td>
</tr>
<tr>
<td>MAC*</td>
<td>ST/N</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
Always Better than Pre-sampling 😊

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Our ROM-AI Bound</th>
<th>Pre-Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWF*</td>
<td>ST/N</td>
<td>(ST/N)^{1/3}</td>
</tr>
<tr>
<td>PRG</td>
<td>(ST/N)^{1/2}</td>
<td>(ST/N)^{1/3}</td>
</tr>
<tr>
<td>PRF</td>
<td>(ST/N)^{1/2}</td>
<td>(ST/N)^{1/3}</td>
</tr>
<tr>
<td>MAC*</td>
<td>ST/N</td>
<td>(ST/N)^{1/3}</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
### Nearly Tight 😊

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Our ROM-AI Bound</th>
<th>Best Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OWF</strong></td>
<td>ST/N</td>
<td>Min(SST/N, (S^2T/N^2)^{1/3})</td>
</tr>
<tr>
<td><strong>PRG</strong></td>
<td>(ST/N)^{1/2}</td>
<td>(ST/N)^{1/2}</td>
</tr>
<tr>
<td><strong>PRF</strong></td>
<td>(ST/N)^{1/2}</td>
<td>(ST/N)^{1/2}</td>
</tr>
<tr>
<td><strong>MAC</strong></td>
<td>ST/N</td>
<td>Min(SST/N, (S^2T/N^2)^{1/3})</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
But Much Weaker than ROM 😞

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Our ROM-AI Bound</th>
<th>Traditional ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWF*</td>
<td>ST/N</td>
<td>T/N</td>
</tr>
<tr>
<td>PRG</td>
<td>$(ST/N)^{1/2}$</td>
<td>T/N</td>
</tr>
<tr>
<td>PRF</td>
<td>$(ST/N)^{1/2}$</td>
<td>T/N</td>
</tr>
<tr>
<td>MAC*</td>
<td>ST/N</td>
<td>T/N</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
But Much Weaker than ROM 😞

Non-uniform attackers too strong!!!

Maybe we can all live in peace?
How to Defeat Preprocessing?

Extensively used in theory and practice:
Saw the magic for CRHF already!

Chose random public salt after preprocessing;
Prepend as input to $O$
## Security Bounds for Salting

\( O: [K] \times [N] \rightarrow [M] \)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Salted ROM-AI Bound</th>
<th>Traditional ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWF*</td>
<td>( T/N + ST/KN )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>PRG</td>
<td>( T/N + (ST/KN)^{1/2} )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>PRF</td>
<td>( T/N + (ST/KN)^{1/2} )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>MAC*</td>
<td>( T/N + ST/KN )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>CRHF</td>
<td>( T^2/M + S/K )</td>
<td>( T^2/M )</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
## Security Bounds for Salting

\[ O: [K] \times [N] \rightarrow [M] \]

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Salted ROM-AI Bound</th>
<th>Traditional ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWF*</td>
<td>( T/N + (STKN)^{1/2} )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>PRG</td>
<td>( T/N + (STKN)^{1/2} )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>PRF</td>
<td>( T/N + (STKN)^{1/2} )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>MAC*</td>
<td>( T/N + STKN )</td>
<td>( T/N )</td>
</tr>
<tr>
<td>CRHF</td>
<td>( T^2/M + SK )</td>
<td>( T^2/M )</td>
</tr>
</tbody>
</table>

* Length preserving for simplicity
Salting Provably helps!

At most $n$ bits of salt yield $\approx$ same security in ROM with auxiliary input as without auxiliary input.

n-bit salt provably defeats preprocessing
Summary

T → S,T

Concrete Crack Repair

Before

After
Summary

• **ROM-AI is the new “cool kid” in town!**
  – very clean: just $S$ and $T$!
  – addresses both *theory* (non-uniformity) and *practice* (preprocessing)
  – non-trivial, yet *elegant* proofs
  – 1000’s of ROM papers need re-evaluation!

• **Obfuscation without the mess!**
Thanks!

Your proposal is written with clarity and conviction. Send it up to legal for obfuscation.
Follow-Up Work [CDGS17]

- **Optimal Pre-Sampling Error** $ST/P$
  - Improves $(ST/P)^{1/2}$ [Unruh07]
  - Gives tight bounds for indistinguishability apps
- **New pre-sampling** for unpredictability apps
  - Matches compression for all current apps
- **Salting generically defeats preprocessing**
- **Random Permutation and Ideal Cipher** with Auxiliary Input
Limitation of Pre-sampling

\begin{align*}
\text{Random Function} & \quad \approx \varepsilon \\
O: [N] \rightarrow \{0, 1\} & \quad \text{Random Function} \\
A = (A_0, A_1) & \quad R: [N] \rightarrow \{0, 1\} \mid \text{Pre}^0 \\
\Pr[A_1^O(A_0(O)) = 1] & \quad |\text{Pre}| = P \\
\Pr[A_1^R|\text{Pre}(A_0(O)) = 1] & > \frac{1}{24}P \\
> \frac{1}{2} + \frac{1}{3}L^{1/2} & \leq \frac{1}{2} + \frac{P}{2L} \\
A_0(O) = \text{Marj}(O_1, \ldots, O_L) & \quad A_1 = 1 \text{ if } A_0(O) = O_i \text{ where } i \sim [L]
\end{align*}
Extending to large $S, T$

Extending to large $T$: $\text{xor}$ first

Extending to large $S$: $\text{repeat}$ on disjoint inputs