The Random Oracle Paradigm

Mike Reiter

Based on “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols” by M. Bellare and P. Rogaway

Random Oracles

- “Random oracle” is a formalism to model such uses of hash functions that abound in “practical” cryptography

- Examples where f is a trapdoor permutation
  - Encryption:
    \[ E^h(x) = f(r) \| h(r) \oplus x \text{ for } r \leftarrow \text{domain}(f) \text{ and } h: \{0,1\}^* \rightarrow \{0,1\}^\infty \]
  - Signatures:
    \[ S^h(x) = f^{-1}(h(x)) \text{ for } h: \{0,1\}^* \rightarrow \text{range}(f) \]

- Such “heuristic” constructions cannot be proven secure under standard cryptographic assumptions
  - But can be proven secure in the “random oracle model”
Random Oracle Model

- If $h$ is a random oracle, then for each $x$, $h(x)$ is drawn uniformly at random from the range of $h$
  - If range of $h$ is $\{0,1\}^*$ then each bit of $h(x)$ is chosen randomly
  - $h(x)$ returns the same value if invoked twice

- Strategy for using random oracles to solve problem $\Pi$
  - Define $\Pi$ in the model of computation in which all parties (including the adversary) share a random oracle $h$
  - Develop an efficient protocol $P$ for $\Pi$ in this model
  - Prove that $P$ satisfies $\Pi$
  - Replace oracle accesses to $h$ by computation of some hash function that seems "good enough"

IND-CPA and IND-CCA in the RO Model

- Let $H$ denote the set of all functions $h:\{0,1\}^* \rightarrow \{0,1\}^l$
- We modify the definition of encryption so that encryption and decryption take a random $h$ as an oracle
  - We denote this by $\mathcal{AE} = (\mathcal{K}, E^h, D^h)$
- The adversary $A$ is also given $h$ as an oracle

\[
\begin{align*}
\text{Experiment } &\text{Expt}^{\text{ind-cca-}\beta}_{\mathcal{AE}}(A) \\
&h \leftarrow_R H \\
&(pk, sk) \leftarrow \mathcal{K}() \\
&b' \leftarrow A^{h, E^h_{(\cdot,\cdot), \beta}}(pk) \\
&\text{return } b'
\end{align*}
\]
IND-CPA Encryption in the RO Model

- Let $h: \{0,1\}^* \rightarrow \{0,1\}^l$
- Consider the following encryption scheme $\mathcal{AE} = \langle \mathcal{K}, \mathcal{E}^h, \mathcal{D}^h \rangle$
  - Let $\mathcal{AE} = \langle \mathcal{K}, \mathcal{E}, \mathcal{D} \rangle$ be a deterministic asymmetric encryption scheme
  
  Algorithm $\mathcal{E}^h_{pk}(M)$
  $r \leftarrow \text{Plaintexts}(pk)$
  return $\mathcal{E}^h_{pk}(r), h(r) \oplus M$

Definition:

Experiment $\text{Exp}_\text{OWF}^\text{ind-CPA}(A)$

$(pk, sk) \leftarrow \mathcal{K}()$

$M \leftarrow \text{Plaintexts}(pk)$

$C \leftarrow \mathcal{E}^h_{pk}(M)$

$M' \leftarrow \mathcal{A}(pk, C)$

if $M = M'$ return 1 else return 0

Proposition:

$\text{Adv}^\text{ind-CPA}_{\mathcal{AE}}(t, q_h, \mu_h, q_e, lq_e) \leq q_e \cdot \text{Adv}^\text{OWF}_{\mathcal{AE}}(t')$

where $t' = O(t)$.

Proof: Given an IND-CPA attacker $A$, we build a OWF attacker $B$. $B$, on input $(pk, C)$, implements $h$ and $\mathcal{E}^h_{pk}()$ for $A$.

For any $h(r)$ query by $A$ or $B$:

- If $\mathcal{E}^h_{pk}(r) = C$, then halt and return $r$.
- If $h(r)$ was previously queried, output same response.
- Otherwise $B$ generates $y \leftarrow \{0,1\}$ and saves $(r, y)$.

For any $\mathcal{E}^h_{pk}(LR(M_0, M_1, b))$ query, implement encryption faithfully.

- Except for query $i$, $1 \leq i \leq q_e$, chosen randomly, where $B$ responds with $(C, y)$ where $y \leftarrow \{0,1\}^l$. 


IND-CCA Encryption in the RO Model

- Let \( h_1 : \{0,1\}^* \to \{0,1\}^l_1 \) and \( h_2 : \{0,1\}^* \to \{0,1\}^l_2 \)
- Consider the following encryption scheme \( \tilde{\mathcal{AE}} = (\mathcal{K}, \tilde{E}^{h_1, h_2}, \tilde{D}^{h_1, h_2}) \)
  with plaintext space \( \{0,1\}^l_1 \)

Let \( \tilde{\mathcal{AE}} = (\mathcal{K}, \tilde{E}, \tilde{D}) \) be a deterministic asymmetric encryption scheme.

Algorithm \( \tilde{E}^{h_1, h_2}(M) \):
- \( r \leftarrow \text{Plaintexts}(pk) \)
- \( C_1 \leftarrow \tilde{E}_{pk}(r) \)
- \( C_2 \leftarrow h_1(r) \oplus M \)
- \( C_3 \leftarrow h_2(r, M) \)
- return \( C_1, C_2, C_3 \)

Algorithm \( \tilde{D}^{h_1, h_2}(C_1, C_2, C_3) \):
- \( r \leftarrow \tilde{D}_{sk}(C_1) \)
- \( M \leftarrow h_1(r) \oplus C_2 \)
- if \( h_2(r, M) = C_3 \) then return \( M \)
- else return \( \bot \)

Proposition:

\[
\text{Adv}_{\tilde{\mathcal{AE}}}^{\text{ind-cca}} (t, q_h, q_{h_1}, q_{h_2}, q_{e}, q_{e'}, q_{d}, q_{d'}, \tfrac{1}{t}) \leq q_e \cdot \text{Adv}_{\tilde{\mathcal{AE}}}^{\text{owf}} (t') + q_d \cdot 2^{-l_1}
\]

where \( t' = O(t) \).

Proof: Given an IND-CCA attacker \( A \), we build a OWF attacker \( B \).
\( B \), on input \((pk, C)\), implements \( h_1, h_2, \tilde{E}_{pk}^{h_1, h_2}(\cdot) \) and \( \tilde{D}_{sk}^{h_1, h_2}(\cdot) \) for \( A \).
For any \( h_1(r) \) query by \( A \) or \( B \):
- If \( \tilde{E}_{pk}(r) = C \), then halt and return \( r \).
- If \( h_1(r) \) was previously queried, output same response.
- Otherwise \( B \) generates \( y \leftarrow R \{0,1\}^l_1 \) and saves \((r, y)\).
For any \( h_2(r, M) \) query by \( A \) or \( B \):
- If \( \tilde{E}_{pk}(r) = C \), then halt and return \( r \).
- If \( h_2(r, M) \) was previously queried, output same response.
- Otherwise \( B \) generates \( y \leftarrow R \{0,1\}^l_2 \) and saves \((r, M, y)\).
IND-CCA in the RO Model

For any \( E^{h_2}_{\mathcal{P}}(LR(M_1, M_2, b)) \) query, implement it faithfully.

\( \nabla \) Except for query \( i \), \( 1 \leq i \leq q_e \), chosen randomly, where \( B \) responds with \( (C, y, z) \) where \( y \leftarrow_R \{0, 1\}^l \) and \( z \leftarrow_R \{0, 1\}^l \).

Whenever \( A \) queries \( D^{(h_1, h_2)}_{\mathcal{P}}(C_1, C_2, C_3) \)

\( \nabla \) If \( A \) previously queried \( h_1(r) \) and \( h_2(r, M) \) for \( r \) and \( M \) satisfying \( C_1 = E^h_{\mathcal{P}}(r), \ C_2 = M \oplus h_1(r), \) and \( C_3 = h_2(r, M) \), then return \( M \)

\( \nabla \) Otherwise abort

Unfortunately, this is no longer indistinguishable to \( A \).

So, we need to quantify the probability that \( A \) distinguishes the simulation as such.

--

IND-CCA in the RO Model

\( A \) distinguishes the simulation if it queries \( D^{(h_1, h_2)}_{\mathcal{P}}(C_1, C_2, C_3) \) where

\( C_3 = h_2(D_{sk}(C_1), C_2 \oplus h_1(D_{sk}(C_1))) \)

without first querying

\( h_2(D_{sk}(C_1), C_2 \oplus h_1(D_{sk}(C_1))) \)

\( \nabla \) Simulated \( D^{(h_1, h_2)}_{\mathcal{P}}(\cdot, \cdot, \cdot) \) aborts on such a query, but real \( D^{(h_1, h_2)}_{\mathcal{P}}(\cdot, \cdot, \cdot) \) does not

If \( q_d \) oracle queries are made in total, then the probability of this happening is at most \( q_d 2^{-l_2} \).

If \( A \) does not distinguish the simulation, then \( B \) wins with probability at least

\[
\frac{1}{q_e} \text{Adv}^{\text{ind-cca}}_{\mathcal{AT}}(A)
\]
RSA-OAEP

- RSA-OAEP is a standardized asymmetric encryption algorithm using the RSA permutation

- RSA-OAEP is IND-CCA secure, in the random oracle model, based on the one-wayness of the RSA permutation
  - The reduction is not “tight”, however
  - It was standardized under the belief that a tight reduction (in the RO model) to the one-wayness of RSA was shown
  - This turned out to be false, and so while a reduction has been shown, the best known reduction is not tight

RSA-OAEP

- Let $\text{RSA}_{N,e}$ denote the RSA permutation on $k$ bits with public key $(N, e)$, and $\text{RSA}_{N,d}$ its inverse
- Let $G: \{0,1\}^{k_0} \rightarrow \{0,1\}^{k-k_0}$ and $H: \{0,1\}^{k-k_0} \rightarrow \{0,1\}^{k_0}$ denote hash functions (random oracles)
- Let $[M]_{k_1}$ denote $k_1$ least significant bits of $M$, and $[M]^n$ denote $n$ most significant bits of $M$
- Plaintext space is $\{0,1\}^n$, where $k = n + k_0 + k_1$

Algorithm $E_{N,e}(m)$:
- $r \leftarrow_R \{0,1\}^{k_0}$
- $s \leftarrow G(r) \oplus (m, 0^{k_1})$
- $t \leftarrow r \oplus H(s)$
- $c \leftarrow \text{RSA}_{N,d}(s, t)$
- return $c$

Algorithm $D_{N,d}(c)$:
- $(s, t) \leftarrow_R \text{RSA}_{N,d}(c)$
- $r \leftarrow t \oplus H(s)$
- $M \leftarrow s \oplus G(r)$
- if $(M)_{j_{k_1}} = 0^{k_1}$
  - then return $[M]^n$ else return ⊥
Instantiating a Random Oracle

Once we’ve proved something in the random oracle model, we then have to instantiate the random oracle in practice.

There’s no “rigorous” way to do it, and many potential stumbling blocks:

- E.g., MD5: \{0,1\}^* \rightarrow \{0,1\}^{128} was thought to be one-way and collision resistant, and is a natural choice for a random oracle.
  - But \( \forall x \exists y \forall z : \text{MD5}(x||y||z) \) can be computed from \( |x|, \text{MD5}(x) \) and \( z \).

Some purely heuristic advice: choose one or more of these:

- A hash function with truncated output (e.g., first 64 bits of MD5).
- A hash with its input length restricted (e.g., MD5 on inputs < 400 bits).
- A hash function used in some nonstandard way (e.g., MD5(\(x|x)\)).

Example: the first 64 bits of \( h((x|x) \oplus C) \) for fixed random \( C \).