Summary on paper: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols [BR93]

Joe Zhu, Haoran Zhang, Yaw Mireku

May 27, 2024

1 Introduction and Background

This paper presents a bridge between cryptographic theory and practice by introducing a paradigm based on the use of random oracles.

1.1 Disparity between Theoretical Practical views:

Theorists take certain primitives for granted and build schemes on top of these primitives. For example, a one-way function is considered a basic building block, and more complex primitives like pseudorandom functions are built on top of it. However, in practice, powerful primitives like the Data Encryption Standard (DES) are already available and used directly as pseudorandom functions. If one needs a one-way function, it is often constructed from DES, thereby transforming a theoretically "simple" primitive into a "complex" one. This disparity results in inefficiencies and a mismatch between theoretical constructs and practical applications.

Practical primitives not only possess desirable properties, but also have strengths which have not been defined or formalized. For instance, the MD5 hash function, denoted as h_2, with restricted input length ≤ 400, exhibits the properties that it is hard to find an x such that $h_2(x) = x$, that it is hard to find an x such that $h_2(x)$ has Hamming weight exceeding 120, and that $f_a(x) = h_2(xa)$ is a pseudorandom function family; etc [NS98]. These properties, although not fully formalized in theoretical terms, are valuable in practice.

1.2 This Paper's Contribution

1. **The paper raises the implicit philosophy behind the use of a random oracle in an explicitly articulated paradigm**, demonstrating its significant benefits to practical cryptography.
2. **It systematically applies this paradigm** to various cryptographic problems, resulting in efficient solutions.
3. **The paper provides definitions** and proofs to justify the previously "unjustified" uses of hash functions within the random oracle model.
4. **It suggests constructions of hash functions** deemed appropriate to instantiate the random oracle, further discussed in the instantiation section.

1.3 The Random Oracle Paradigm

The proposed paradigm for designing efficient cryptographic protocols involves the following steps:

1. Find a formal definition for a protocol problem Π within a model where all parties have access to a public random oracle R.
2. Design a protocol P for Π in the random oracle model.
3. Prove that P satisfies the definition for Π.
4. Replace accesses to the random oracle R with the computation of an appropriately chosen function h.

1.4 Thesis

When the above method is properly carried out, the resulting protocol will be both secure and efficient. This approach provides a practical pathway to leverage the theoretical strengths of the random oracle model while addressing the efficiency needs of real-world applications. By carefully choosing and using practical primitives, cryptographic protocols can achieve the desired security properties without sacrificing performance.

2 Preliminaries

To ensure that we speak the same language or protocol.

2.1 Some Definitions Outside of Class

Given that we have been taking this cryptography class, we have a solid understanding of the basic notions. However, to fully comprehend the concepts presented in this paper, we need to define some additional terms.

PPT stands for Probabilistic Polynomial Time. **Negligible Function:** A function $\epsilon(k)$ is considered negligible if, for every positive integer c, there exists an integer k_c such that for all $k > k_c$, $\epsilon(k) \leq \frac{1}{k^c}$. In other words, as k grows, $\epsilon(k)$ becomes exceedingly small. **Non-negligible Function:** A function is non-negligible if it is not negligible

2.2 Most Things Follow the Convention from Class

The following definitions have been discussed in class, albeit with minor variations. For the sake of clarity, they will be explicitly redefined here.

$A \rightarrow a$: This A is a probabilistic algorithm and a is its output. And $[A(x, y, z, \dots)]$ denotes the set of all possible outputs of A when run with inputs $x, y, z, \dots$.

$\{0,1\}^*$ denotes the set of all finite binary strings. $\{0,1\}^\infty$ represents the set of all infinite binary strings. Concatenation of strings a and b is denoted by $a||b$ or simply ab. The empty string is denoted by Λ.

Random Oracles: A random oracle R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$, chosen uniformly at random from the set of all such maps. The notation 2^∞ denotes the set of all random oracles. R represents a "generic" random oracle.

$G : \{0,1\}^* \rightarrow \{0,1\}^\infty$ denotes a random generator, which produces an infinite sequence of random bits.

$H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a random hash function that maps a binary string of any length to a fixed-size output of k bits.

When multiple random oracles are used, they are assumed to be independently chosen to ensure their outputs are not correlated.

Trapdoor Permutations: $\mathcal{G}$ denotes a PPT algorithm that generates trapdoor permutations. On input 1^k, $\mathcal{G}$ outputs a triple of algorithms (f, f^{-1}, d), where:

- f is a permutation function,
- f^{-1} is its inverse, denoted by g in class,
- d is a probabilistic algorithm whose output is a subset of $\{0,1\}^k$.

This d corresponds to the modulus N in the RSA encryption scheme seen in class. The functions f and f^{-1} must be computable in polynomial time.

For an adversary A, the probability $\epsilon(k) = Pr[(f, f^{-1}, d) \leftarrow \mathcal{G}(1^k); y \leftarrow f(x) : A(f, y, d) \Rightarrow 1]$ is negligible. This indicates that the adversary's chance of inverting f without the trapdoor information f^{-1} is negligible, ensuring the security of the permutation.

3 Encryption

Adversaries are considered non-uniform algorithms in the time complexity domain. By extending the public key encryption notion to the Random Oracle Model, we specify the scheme by a PPT generator $\mathcal{G}$. The generator then takes in security parameter 1^k and outputs a pair of probability encryption and decryption algorithms, also known as, (E, D) which run in $\mathcal{G}$'s time complexity. In other words, if one wants to encrypt a message x to user U, they can compute $y \leftarrow E^R(x)$ and send it to the U. U can decrypt a ciphertext y by computing $x \leftarrow D^R(y)$. $D^R(E^R(x)) = x$ is required for all x. If y is not encrypted under E^R of any x, then $D^R(y) = 0$.

3.1 Polynomial Security

The "basic" security goal of public key encryption was shown by Goldwasser and Micali's polynomial and semantic security. [SS84] If B_f denotes a predict for f, then according to Goldwasser and Micali's probabilistic encryption, the security can be shown by $E(x) = f(r_1)||...||f(r_|x|)$ where every r_1 is randomly selected from f and limited to $B_f(r_i) = x_i$, which has a complexity of $O(k \cdot |x|)$, and it takes evaluations of $O(|x|)$ of f to encrypt and f^-1 to decrypt. Goldwasser and Micali's polynomial security is adapted to the random oracle model. A CP-adversary is introduced, in which A is (F, A_1), namely two non-uniform polynomial time algorithms with access to an oracle. For $\mathcal{G}$ to be a secure encryption scheme in the random oracle model, we suppose for any CP-adversary $A = (F, A_1)$, $\Pr[R^2; (m_0, m_1) \leftarrow F^R(E);\ b \leftarrow \{0,1\};\ \alpha \leftarrow E^R(m_b) : A_1^R(E, m_0, m_1, \alpha) = b] \leq \frac{1}{2} + k^{-w(1)})$. By $E(x) = f(r)||G(r) \oplus x$, let $\mathcal{G}_*$be a trapdoor permutation generator and let $G : \{0,1\}^* \to \{0,1\}^\infty$ be a random generator showing that this scheme is polynomially secure in the random oracle model and achieving encryption size $O(|x| + k)$.

3.2 Chosen Ciphertext Security

Naor and Yung defined the chosen ciphertext security [MM90] and created the first scheme to possibly achieve it. [RS92] Rackoff and Simon later provided a stronger scheme with a solution to achieve, which is adapted to the random oracle setting. In an RS-adversary, also known as "Rackoff-Simon Adversary," $A = (F, A_1)$ with each having access to an oracle R and D^R, a black box implementation. Encryption scheme $\mathcal{G}$ securely fights against RS-attack if for each RS-adversary:

$A = (F, A1), \Pr[R \leftarrow 2^\infty; (E, D) \leftarrow \mathcal{G}(1^k); (m_0, m_1) \leftarrow F^{R,D^R}(E); b \leftarrow \{0,1\};\ \alpha \leftarrow E^R(m_b) : A_1^{R,D^R}(E, m_0, m_1, \alpha) = b] \leq \frac{1}{2} + k^{-w(1)}$.

There appears to be a secure scheme against RS-attack efficiently. By $E(x) = f(r)||G(r) \oplus x||H(rx)$, let $\mathcal{G}_*$ be a trapdoor permutation generator. Let $G : \{0,1\}^* \to \{0,1\}^\infty$ be a random generator, and let $H : \{0,1\}^* \to \{0,1\}^k$ be an independently derived hash function from the random oracle. The theorem is to show this scheme is secure against chosen-ciphertext attack. In comparison to Zheng and Seberry's scheme, $E^*(x) = f(r)||G(r) \oplus x||H(x)))$, this scheme is as efficient and carries the same security properties.

3.3 Non-Malleability

Dolev, Dwork and Naor introduced the non-malleability notion [DM91], which means that one cannot formulate an encryption scheme of string $x\prime$ by seeing an encryption of a string x. Their work of non-malleability is adapted to the random oracle setting, resulting in a relation that $p^R_{E,\pi} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ requires to satisfy $p^R_{E,\pi}(x, x) = p^R_{E,\pi}(x, 0^i) = 0$ for every $x \in \{0,1\}^*$, $i \in N, R \in 2^\infty$, and $E, \pi \in \{0,1\}^*$; p must be computable by $M^R(x, y, E, \pi)$, namely a polynomial time Turing machine. In order for encryption scheme $\mathcal{G}$ to be non-malleable, for every interesting relation p and M-adversary (F, A) there is a A_* such $|\epsilon(k) - \epsilon_*(k)|$ is negligible,

$\epsilon(k) = \Pr[R \leftarrow 2^\infty; (E, D) \leftarrow \mathcal{G}(1^k); \pi \leftarrow F^R(E); x \leftarrow \pi^R(1^k); \alpha \leftarrow E^R(x); \alpha\prime \leftarrow A^R(E, \pi, \alpha) : p^R_{E,\pi}(x, D^R(\alpha\prime)) = 1]$

$\epsilon(k) = \Pr\ [R \leftarrow 2^\infty; (E, D) \leftarrow \mathcal{G}(1^k); \pi \leftarrow F^R(E); x \leftarrow \pi^R(1^k);\ \alpha\prime_* \leftarrow A^R_*(E, \pi) : p^R_{E,\pi}(x, D^R(\alpha\prime_*)) = 1]$.

The encryption scheme is the same as the chosen ciphertext security of the previous paragraph, which is shown to be non-malleable.

4 Signatures

A digital signature scheme consists of three polynomial-time algorithms:

1. The generator
2. The signing algorithm
3. The verifying algorithm

The generator's purpose in this scheme is to produce a public and secret key pair. The signing algorithm creates a signature for a message using the secret key, and it is the job of the verifying algorithm to check the signature validity with the public key. This verification has to work for any signature generated by the signing algorithm.

To ensure security, the scheme should be resilient to forgeries by any polynomial-time adversary that has access to both the random and signing oracles. Specifically, an adversary shouldn't be able to produce a valid signature for any message it hasn't yet queried from the signing oracle. This notion is formalized with the use of a probabilistic polynomial-time definition, where the adversary must have a negligible success probability. A trapdoor function is a type of function that is easy to compute but hard to invert without some kind of secret. A signature scheme based on a trapdoor permutation that is uniform could have a signature generation that involves hashing the message and applying the inverse of the permutation to the hash. Verification would check if applying the permutation to the signature yields the hash of the message.

Due to non-uniformity in standard trapdoor permutations like RSA, any such scheme would need modifications to ensure uniformity. One way to accomplish this is repeatedly hashing the message with increasing prefixes until a valid domain member is found before signing. Another method involves transforming the domain to be uniform using techniques from other cryptographic constructions.

The security analysis demonstrates that if any adversary can forge a signature with non-negligible probability, it implies the ability to invert the permutation function. This reduction is a strong security guarantee since a trapdoor function is difficult to invert directly. This connects back to the main idea of demonstrating that a random oracle, though an idealized model, can enhance the security and efficiency of cryptographic protocols.

5 Zero Knowledge

Random oracles can also be used to enhance cryptographic protocol in the context of zero-knowledge proofs. Interactive proofs involve a back-and-forth communication between a prover and a verifier, while in non-interactive proofs, the verifier can independently check a proof's validity without further interaction with the prover. The random oracle model can be used to convert interactive proofs into non-interactive ones while still maintaining both efficiency and security.

5.1 Definitions

Zero-knowledge proofs allow a prover to convince a verifier that a statement is true without revealing any additional information. In the random oracle model, these proofs are extended from traditional interactive settings, and both the prover and verifier have access to a random oracle. By the completeness condition, an honest prover should be able to convince an honest verifier if a statement is true. On the other hand, soundness ensures that a dishonest prover can't convince a verifier of a false statement except with negligible probability. The verifier in these cases is able to see the transcript of the interaction and the random oracle's responses.

Ensuring the zero-knowledge property involves demonstrating that a simulator can create indistinguishable transcripts from actual prover-verifier interactions. The simulator can prescribe part of the oracle responses before the rest is filled randomly, and the verifier would not be able to distinguish between real and simulated interactions.

5.2 Protocol

An interactive zero-knowledge proof can be transformed into a non-interactive one by using a random hash function to replace the verifier's random challenges. The prover computes messages and challenge bits based on the hash of these messages and then sends them back to the verifier. After this step, the verifier is able to check if all challenges are met correctly without having to send any more challenges or information.

The new non-interactive proof retains completeness, soundness, and zero-knowledge properties. The transformed protocol is also efficient, with the amount of computation done by the prover and verifier increased only by a polynomial factor.

6 Instantiation

The authors provide the following guidance for how one can instantiate a random oracle: First, Note that the specific details of the target protocol, whose random oracles are being instantiated, are not of primary importance. Greater emphasis is placed on the number of oracles used and their input/output length requirements. Second, choosing a concrete function h to instantiate an oracle requires significant care. Here are some pitfalls that one should be aware of:

MD5 as an Oracle: MD5 is not suitable because it has structural properties [Tsu92] that allow for easy computation of certain values given an input and its MD5 hash. for any x, there exists a y such that for any z, MD5$(x||y||z)$ can be computed easily given the length of x, MD5(x), and z. This structure makes MD5 unsuitable as a random oracle.

MD5 Compression Function: MD5 Compression Function as a replacement to avoid the structural problem mentioned above also fails due to the possibility of efficiently finding collisions, as demonstrated by prior research [DBB93].

6.1 Suitable Candidates for Instantiation

Although standard hash functions (as above) are too structured to make good random oracles, the authors suggest several candidate construction methods for instantiating random oracles:

1. **Truncated Hash Functions:** Using a hash function with its output truncated or folded in some manner. For example, $h_1(x)$ could be the first 64 bits of MD5(x).
2. **Restricted Input Length:** Using a hash function with restricted input lengths. For example, $h_2(x)$ could be MD5(x), where $|x| < 400$.
3. **Nonstandard Usage:** Using a hash function in a nonstandard way. For example, $h_3(x)$ could be MD5$(x||x)$, where x is a fixed string.
4. **First Block Compression Function:** Using the "first block compression function" of a cryptographic hash function. For example, h_4 could be the compression of the first 512-bit block when MD5(x) is computed.

6.2 An Example

The authors give an example of constructing a good replacement for a random oracle using a heuristic map h' from $\{0,1\}^{256} \rightarrow \{0,1\}^{64}$ defined as the first 64 bits of $h_4((x) \oplus C)$, where h_4 is a function derived from a cryptographic hash function's first block compression function, and C is a randomly chosen 512-bit constant.

Extending Domain and Range to match one's needs: for example, constructing a map $h : \{0,1\}^* \rightarrow \{0,1\}^\infty$ using the following steps.

1. You can define
$$h''(x) = h'(x_0\langle 0\rangle)||h'(x_1\langle 1\rangle)||h'(x_2\langle 2\rangle)||\ldots$$
where $|x| = 224$ and $\langle i\rangle$ is the 64-bit encoding of i.
2. Extends h'' by encoding each input x by x', consisting of x, the bit "1", and enough "0"s to make $|x'|$ a multiple of 128 bits.
3. Finally, define $h(x)$ by combining the outputs of h'' using XOR:
$$h(x) = h''(x'_0\langle 0\rangle) \oplus h''(x'_1\langle 1\rangle) \oplus \ldots \oplus h''(x'_n\langle n\rangle)$$

Note that this is one way to instantiate a random oracle; there are lots of other equally valid ways to do so.

7 Conclusion

The protocols applied in this paper are constructed via procedures of selecting a concrete protocol, searching for a successful attack, creating a solution and pursuing an end. However this method is problematic when it comes to declaring goals and finding solutions to them. Since modern cryptography contains more than a specific set of solutions to a problem, this approach is not very effective in terms of solving specific problems. Although improvements can be made, this methodology still manages to be more than just declaring a protocol.

References

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, page 62–73, New York, NY, USA, 1993. Association for Computing Machinery.

[DBB93] Bert Den Boer and Antoon Bosselaers. Collisions for the compression function of md5. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 293–304. Springer, 1993.

[DM91] Dwork Danny, Dolev Cynthia and Naor Moni. Non-malleable cryptography. page 542–552, New York, NY, USA, January 1991. Association for Computing Machinery.

[MM90] Naor Moni and Yung Moti. Public-key cryptosystems provably secure against chosen ciphertext attacks. page 427–437, New York, NY, USA, April 1990. Association for Computing Machinery.

[NS98] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, page 59–66, New York, NY, USA, 1998. Association for Computing Machinery.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 433–444, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[SS84] Goldwasser Shafi and Micali Silvio. Journal of computer and system sciences. pages 270–299, April 1984.

[Tsu92] Gene Tsudik. Message authentication with one-way hash functions. *SIGCOMM Comput. Commun. Rev.*, 22(5):29–38, oct 1992.