Post-quantum security of cryptographic transformations in the random oracle model

Huang, Y.

Chapter 2

Preliminaries

Throughout this thesis, we assume readers' familiarity with general mathematics and a basic understanding of algorithms, computational complexity, and quantum information science. In this chapter, though, we set out our notation and terminology, and discuss several aspects that are particularly relevant.

2.1 Notations

Basic mathematics. Throughout the thesis, we take the convention that $\ln(x)$ and $\log(x)$ are the natural and binary logarithm respectively, and for every positive integer n we denote by $[n] := \{1, \ldots, n\}$.

The security parameter. In cryptography, it is a standard convention that all algorithms and adversaries (and likewise, functions) take as input the unary representation 1^λ of a security parameter λ. Throughout this thesis, we adopt this convention and keep λ implicit unless necessary. Whenever the considered algorithms or functions are associated with a mathematical object, such as their input/output space or an oracle they query, it is to be understood that the object is implicitly parameterized by λ as well.

The arrow notation. For a finite set $\mathcal{X}$, we will denote a random variable x being sampled uniformly from the set by $x \leftarrow \mathcal{X}$. For a distribution $\mathcal{D}$ that is not necessarily uniform, though, we will write $x \leftarrow \mathcal{D}$ to indicate that x is drawn from $\mathcal{D}$. Similarly, since an algorithm may in general have a randomized output, we denote by $x \leftarrow \mathcal{A}$ that x is an output generated by running $\mathcal{A}$. Sometimes we are interested in the probability that $x = x^\circ$ for a certain fixed value x°, in which case we often use the following short-hand notation

$$\Pr\left[\, x^\circ \leftarrow \mathcal{A} \,\right] := \Pr_{x \leftarrow \mathcal{A}}\left[\, x = x^\circ \,\right] .$$

Signature schemes. Following the standard definition, a *signature scheme* $\mathcal{S}$ consists of a key generation algorithms KGen, a signing algorithm Sign, and a verification algorithm Vrfy, all of which are PPT algorithms. By default, we denote by $\mathcal{PK}$ the space of public keys, by $\mathcal{SK}$ the space of secret keys, by $\mathcal{M}$ the message space and by $\mathcal{SGN}$ the space of signatures. For simplicity, we assume throughout that $\mathcal{S}$ is perfectly correct, i.e. $\mathsf{Vrfy}\big(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)\big) = 1$ with certainty, and that the public key can be efficiently derived from the secret key.

Statistical distance. For any two random variables x_1, x_2 over a finite set $\mathcal{X}$, we denote their *statistical distance* by

$$SD(x_1, x_2) := \frac{1}{2} \sum_{x^\circ \in \mathcal{X}} \big| \Pr[\, x_1 = x^\circ \,] - \Pr[\, x_2 = x^\circ \,] \big| \,,$$

which is known to equal $\max_V(\Pr[\, V(x_1) = 1 \,] - \Pr[\, V(x_2) = 1 \,])$ with the maximization over all predicates $V : \mathcal{X} \to \{0, 1\}$.

Guessing probability and min-entropy. For a random variable y over a finite set $\mathcal{Y}$, the *guessing probability* and the *min-entropy* are respectively defined as

$$\mathsf{guess}(y) := \max_{y^\circ} \Pr\big[\, y = y^\circ \big] \qquad \text{and} \qquad \mathsf{H}_\infty(y) := -\log\big(\mathsf{guess}(y)\big) \,.$$

For random variables x and y over respective finite sets $\mathcal{X}$ and $\mathcal{Y}$, the *conditional* guessing probability is defined as

$$\mathsf{guess}(y \mid x) := \sum_{x^\circ} \Pr[x = x^\circ] \cdot \max_{y^\circ} \Pr\big[\, y = y^\circ \;\big|\; x = x^\circ \big] \,.$$

It is well known that $\mathsf{guess}(y \mid x) = \max_f \Pr[y = f(x)]$, where the maximization is over all (deterministic or randomized) functions $f : \mathcal{X} \to \mathcal{Y}$. In line with the unconditional case above, the *conditional* min-entropy is then given by $\mathsf{H}_\infty(y \mid x) := -\log\big(\mathsf{guess}(y \mid x)\big)$.

2.2 Oracle Algorithms

Throughout this thesis, we frequently consider the random oracle model (ROM), where parties/algorithms have oracle access to a uniformly random function $H : \mathcal{X} \to \mathcal{Y}$, with suitably chosen finite domain $\mathcal{X}$ and range $\mathcal{Y}$. We also consider quantum algorithms, which can then query H "in superposition," i.e. in the form of the unitary $|x, y\rangle \mapsto |x, y + H(x)\rangle$. We write $\mathcal{A}^H$ to denote a (classical or quantum) algorithm with query access to H. Similarly, we write $\mathcal{A}^{H,\bullet}$

in case $\mathcal{A}$ may additionally make queries to another, possibly unspecified oracle, hinted at with a placeholder "$\bullet$" (or $\blacklozenge$, $\blacksquare$, $\blacktriangle$ etc.). Furthermore, in case of such an oracle algorithm $\mathcal{A}^{H,\bullet}$, we write $\mathcal{A}^{H,\mathcal{O}}$ to express that the unspecified oracle is instantiated with the particular algorithm $\mathcal{O}$. We note that such an instantiation may also have access to H, i.e., the oracle may be instantiated with an algorithm $\mathcal{O}^H$, in which case we then naturally write $\mathcal{A}^{H,\mathcal{O}^H}$.

We stress that we use the same notation for classical and quantum algorithms with respective classical or quantum access to the random oracle H, while the query access to any other oracle is always assumed to be classical in this work, and thus so is any oracle instantiation $\mathcal{O}$ we consider. Furthermore, by default we assume any oracle instantiation $\mathcal{O}$ to be at most *statically stateful*, meaning that the state (if any) is chosen at the beginning and then remains fixed.[1]

Regarding language and notation, since in the random oracle model by default all algorithms have access to H, we reserve the terminology *oracle* algorithm for those that have access to one (or more) additional oracle(s). Also, once we have specified that an oracle algorithm is of the form, say, $\mathcal{A}^{H,\bullet}$, i.e., with access to H and to another, unspecified oracle, we may then simply write $\mathcal{A}$ in later occurrences, taking the considered form as understood.

We note that even though an oracle algorithm $\mathcal{A}^{H,\bullet}$ typically expects a particular instantiation $\mathcal{O}$ of the oracle, we may consider a run of $\mathcal{A}^{H,\mathcal{O}'}$ for any instantiation $\mathcal{O}'$. We may also consider a run of $\mathcal{A}$ where different calls to the oracle are answered by different instantiations. We then write

$$\mathcal{A}^{H,[\mathcal{O}_1^{i_1},\mathcal{O}_2^{i_2},\ldots]}$$

to capture that the first i_1 oracle queries are answered by $\mathcal{O}_1$, the following i_2 queries by $\mathcal{O}_2$, etc. We note that in-between these oracle queries, there may be multiple queries to H.

For proof-technical reasons, we will also consider the case where an (oracle) algorithm may make *write* (a.k.a. *reprogramming*) *queries* to the random oracle H. On input (x, y), such a write query will redefine the function value of $H(x)$ to y; we will capture this by the command $H(x) := y$. We only consider *classical* write queries, and we will write $\mathcal{A}^{\bar{H}}, \mathcal{A}^{\bar{H},\bullet}, \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$ etc. to indicate that $\mathcal{A}$ (and $\mathcal{O}$) may also make classical write queries to H, next to the ordinary (classical or quantum) read queries.

For an oracle algorithm $\mathcal{A}^{\bar{H},\bullet}$ and a specific instantiation of the oracle, say, as $\mathcal{O}^{\bar{H}}$, we write

$$\mathcal{B}^{\bar{H}} := \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$$

[1] For comparison, the random oracle H with read queries only is statically stateful when instantiated/implemented (inefficiently) by choosing a random function at the beginning, while it is adaptively stateful when done using lazy sampling. The random oracle $\bar{H}$ with write queries (see below) is adaptively stateful no matter how it is implemented.

to specify that the algorithm $\mathcal{B}$ runs $\mathcal{A}$ and answers all oracle queries by means of running $\mathcal{O}^{\bar{H}}$ internally, while forwarding all random oracle queries of $\mathcal{A}$ and $\mathcal{O}$ to H. This indeed makes $\mathcal{B}$ an algorithm of the form $\mathcal{B}^{\bar{H}}$.

Pushing this a bit further, $\mathcal{B}^{\bar{H},\bullet} := \mathcal{A}^{\bar{H},[\mathcal{O}^{i-1},\bullet,\mathcal{P}^{\infty}]}$ then denotes the oracle algorithm that runs $\mathcal{A}$, answers the first $i-1$ oracle queries with $\mathcal{O}$, forwards the ith query to $\mathcal{B}$'s oracle, and answers all the remaining ones with $\mathcal{P}$. In case $\mathcal{A}$ makes at most q oracle queries, we could just as well write $\mathcal{P}^{q-i}$ instead of $\mathcal{P}^{\infty}$.

2.2.1 Equivalences of Oracle Algorithms

Different (oracle) algorithms may "behave the same way". We want to formally capture the possible meanings of the latter that will be important for us.

We say that two algorithms $\mathcal{B}^{\bar{H}}$ and $\mathcal{C}^{\bar{H}}$ are *semantically equal*, written as equality $\mathcal{B}^{\bar{H}} = \mathcal{C}^{\bar{H}}$, if the *joint distribution* of: (1) the output of the considered algorithm, and (2) the (possibly reprogrammed) random oracle H at the end of the execution of the algorithm, are the same, *for any initial choice* of H and any joint input. This is for instance satisfied when $\mathcal{C}$ is a purely syntactic rewriting of the defining code of $\mathcal{B}$.

Another natural, but weaker, equivalence relation, which we call *output equivalent*, is $\mathcal{B}^{\bar{H}} \simeq \mathcal{C}^{\bar{H}}$, which (by our definition) expresses that the two distributions of the respective *outputs only* are equal, *on average* over the random choice of H and for any joint input.

We stress that the assignment $\mathcal{B}^{\bar{H}} := \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$ implies semantical equivalence $\mathcal{B}^{\bar{H}} = \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$. Furthermore,

$$\mathcal{O}^{\bar{H}} = \mathcal{P}^{\bar{H}} \implies \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}} = \mathcal{A}^{\bar{H},\mathcal{P}^{\bar{H}}}$$

for any $\mathcal{A}^{\bar{H},\bullet}$, while $\mathcal{O}^{\bar{H}} \simeq \mathcal{P}^{\bar{H}}$ is in general not sufficient to conclude $\mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}} \simeq \mathcal{A}^{\bar{H},\mathcal{P}^{\bar{H}}}$.[2]

Output equivalence sometimes only holds approximately, informally denoted as $\mathcal{B}^{\bar{H}} \approx \mathcal{C}^{\bar{H}}$ then. This is commonly quantified via the *distinguishing advantage*

$$\left|\Pr\left[1 \leftarrow \mathcal{B}^{\bar{H}}\right] - \Pr\left[1 \leftarrow \mathcal{C}^{\bar{H}}\right]\right|,$$

where $\mathcal{B}$ and $\mathcal{C}$ are then typically assumed to have a binary output, and where the probability is over the randomness of the algorithms ($\mathcal{B}$ and $\mathcal{C}$) and the random choice of the random oracle H. We note that for simplicity, we consider here algorithms with no input, but the same quantities can also be considered for any choice of input, of course. In case of binary outputs, the above *distinguishing advantage* coincides with the statistical distance $SD\left(\mathcal{B}^{\bar{H}}, \mathcal{C}^{\bar{H}}\right)$ between the two output distributions, and so we then use the two expressions

[2] Even if $\mathcal{O}$ and $\mathcal{P}$ are restricted to read access only, still $\mathcal{O}^{H} \simeq \mathcal{P}^{H} \not\Rightarrow \mathcal{A}^{\bar{H},\mathcal{O}^{H}} \simeq \mathcal{A}^{\bar{H},\mathcal{P}^{H}}$.

interchangeably.[3] In case of non-binary outputs, the statistical distance upper bounds the distinguishing advantage.

2.2.2 Conditioning of Oracle Algorithms

The run of an algorithm $\mathcal{A}^{\bar{H}}$ defines the (distribution of the) output of the algorithm, but also various internal variables, like the local randomness of $\mathcal{A}$ in case of a classical algorithm or the measurement outcomes of measurements performed by $\mathcal{A}$ in case of a quantum algorithm, the (classical) write queries to H, etc. For any event Λ defined by these random variables, we can then consider a run of $\mathcal{A}^{\bar{H}}$ conditioned on Λ, which we will denote by $\mathcal{A}^{\bar{H}}[\Lambda]$. We may use the variation $\mathcal{A}[\Lambda]^{\bar{H}}$ on the notation to indicate that Λ does not depend on H. We mainly use the latter in case of a classical algorithm $\mathcal{A}$, where it then means that Λ is determined by $\mathcal{A}$'s local randomness (and its input). In this case, the assignment $\mathcal{B}^{\bar{H}} := \mathcal{A}[\Lambda]^{\bar{H}}$ is well-defined as the oracle algorithm that runs $\mathcal{A}$ but samples the local randomness conditioned on Λ.

This notation extends naturally to $\mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}[\Lambda]$ for any oracle algorithm $\mathcal{A}^{\bar{H},\bullet}$ and instantiation $\mathcal{O}^{\bar{H}}$, as well as to the variations $\mathcal{A}[\Lambda]^{\bar{H},\bullet}$ and $\mathcal{A}^{\bar{H}}[\Lambda]^{\bullet}$, when Λ does not depend on (H and) the oracle.

The above notation will mainly be useful in the following context. Consider two algorithms $\mathcal{B}^{\bar{H}}$ and $\mathcal{C}^{\bar{H}}$, as well as an event Λ that is well-defined for either of the two executions. Then the equivalences $\mathcal{B}^{\bar{H}}[\Lambda] = \mathcal{C}^{\bar{H}}[\Lambda]$ and $\mathcal{B}^{\bar{H}}[\Lambda] \simeq \mathcal{C}^{\bar{H}}[\Lambda]$ are naturally defined by means of the equality of the respective *conditional* distributions. Furthermore, if Λ has the same probability $\Pr[\Lambda]$ in a run of $\mathcal{B}^{\bar{H}}$ and in a run of $\mathcal{C}^{\bar{H}}$, then

$$SD\big(\mathcal{B}^{\bar{H}}, \mathcal{C}^{\bar{H}}\big) \leq \Pr[\Lambda] \cdot SD\big(\mathcal{B}^{\bar{H}}[\Lambda], \mathcal{C}^{\bar{H}}[\Lambda]\big) + \Pr[\neg\Lambda] \cdot SD\big(\mathcal{B}^{\bar{H}}[\neg\Lambda], \mathcal{C}^{\bar{H}}[\neg\Lambda]\big)\,.$$

In particular, if $\mathcal{B}^{\bar{H}}[\Lambda] \simeq \mathcal{C}^{\bar{H}}[\Lambda]$ then $SD\big(\mathcal{B}^{\bar{H}}, \mathcal{C}^{\bar{H}}\big) \leq \Pr[\neg\Lambda]$.

We conclude by noting that in case of an oracle algorithm $\mathcal{A}^{\bar{H},\bullet}$, an instantiation $\mathcal{O}^{\bar{H}}$ of the oracle, and an event Λ defined by the local randomness of $\mathcal{O}$ (for any fixed input to $\mathcal{O}$), if $\mathcal{A}^{\bar{H},\bullet}$ is promised to make precisely *one* query to the oracle then the event Λ is also well-defined by a run of $\mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$, and $\mathcal{A}^{\bar{H},\mathcal{O}[\Lambda]^{\bar{H}}} = \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}[\Lambda]$.

2.2.3 Simulating the Write Access

It is not too hard to see that write queries to H can always be simulated internally, in the sense that there exists an (adaptively stateful) algorithm $\mathcal{S}^{H}$, which makes one H-query whenever it is invoked with a read request (and no query upon any write request), and such that $\mathcal{A}^{\bar{H}} \simeq \mathcal{A}^{\mathcal{S}^{H}}$ for every $\mathcal{A}^{\bar{H}}$.

[3] The former is more common in the cryptography literature, while the advantage of the latter is its succinctness.

$\mathcal{S}$ simply simulates any write query to H by bookkeeping the reprogramming requests and properly adjusting the future read queries to H, which it relays. It is obvious how this works in case $\mathcal{A}$ makes *classical* read queries to H, but it can also be done in case of quantum read queries, where $\mathcal{S}$ then is a suitable quantum algorithm.

Lemma 2.1. *There exists a stateful oracle algorithm $\mathcal{S}^H$, which makes 1 read query per each read-query invocation, and such that $\mathcal{A}^{\bar{H}} = \mathcal{A}^{\mathcal{S}^H}$ for every $\mathcal{A}^{\bar{H}}$.*

Proof. The oracle algorithm $\mathcal{S}^H$ operates by bookkeeping a list of classical write queries of $\mathcal{A}$. Upon receiving a write query (x, y), it first removes any occurrences of the form $(x, \cdot)$ from the list (if there exists any) and then inserts (x, y) into the list. Answering a read query then can be performed by $\mathcal{S}^H$, by invoking one single query to H, as follows. Given a list $\{(x_1, y_1), \ldots, (x_L, y_L)\} \subset \mathcal{X} \times \mathcal{Y}$ of classical write queries with pairwise distinct x_i's, $\mathcal{S}^H$ (efficiently) computes the unitary $|x, y\rangle \mapsto |x, y + H'(x)\rangle$ by means of a quantum algorithm that makes a single quantum query to H, i.e. to the unitary $|x, y\rangle \mapsto |x, y + H(x)\rangle$, for H' defined as

$$H'(x) = \begin{cases} y_i & \text{if } \exists\, i : x = x_i\,, \\ H(x) & \text{otherwise}\,. \end{cases}$$

This is trivial if $\mathcal{S}$ is given *control* access to H, i.e., access to the unitary $|b, x, y\rangle \mapsto |b, x, y + bH(x)\rangle$. Furthermore, control access to H can be simulated using a single ordinary quantum access to H as follows: $\mathcal{S}$ first prepares a uniform superposition of elements in $\mathcal{Y}$ in an auxiliary register $|z\rangle$, applies a control swap to $|y, z\rangle$ with $|b\rangle$ being the control bit, makes the quantum query to H, and finally applies a control swap again. One can conclude by noting that $\mathcal{S}^H$ perfectly simulates the view of the attacker $\mathcal{A}$. □

By default, we then write $\bar{\mathcal{A}}^H := \mathcal{A}^{\mathcal{S}^H}$ for the algorithm that simulates $\mathcal{A}$'s write queries internally, and thus is such that $\bar{\mathcal{A}}^H \simeq \mathcal{A}^{\bar{H}}$.[4] We note that in case of an oracle algorithm $\mathcal{A}^{\bar{H},\bullet}$ and setting $\bar{\mathcal{A}}^{H,\bullet} := \mathcal{A}^{\mathcal{S}^H,\bullet}$, the equality $\bar{\mathcal{A}}^{H,\mathcal{O}^{\bar{H}}} \simeq \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}$ of the output distributions holds (only) *conditioned* on the event that $\mathcal{O}$ makes no (read or write) query to H on a point that has previously been reprogrammed by $\mathcal{A}$. Thus, $\bar{\mathcal{A}}^{H,\mathcal{O}^{\bar{H}}}[\Omega] \simeq \mathcal{A}^{\bar{H},\mathcal{O}^{\bar{H}}}[\Omega]$ with Ω being the said event.

[4] Here the respective output distributions of the two algorithms are actually equal *for any choice of H*; thus, we have an equivalence that lies in-between $=$ and $\simeq$, but this is not important to us.