Appears in the proceedings of the *First ACM Conference on Computer and Communications Security*, ACM, November 1993.

Random Oracles are Practical: A Paradigm for Designing Efficient Protocols

Mihir Bellare* Phillip Rogaway†

August 2, 2021

Abstract

We argue that the random oracle model —where all parties have access to a public random oracle— provides a bridge between cryptographic theory and cryptographic practice. In the paradigm we suggest, a practical protocol P is produced by first devising and proving correct a protocol P^R for the random oracle model, and then replacing oracle accesses by the computation of an "appropriately chosen" function h. This paradigm yields protocols much more efficient than standard ones while retaining many of the advantages of provable security. We illustrate these gains for problems including encryption, signatures, and zero-knowledge proofs.

*Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: mihir@cs.ucsd.edu

† Department of Computer Science, University of California at Davis, Davis, CA 95616, USA. E-mail: rogaway@cs.davis.edu

1 Introduction

Cryptographic theory has provided a potentially invaluable notion for cryptographic practice: the idea of provable security. Unfortunately, theoretical work often seems to gain provable security only at the cost of efficiency. This is due in part to the following. Theorists view certain primitives (e.g., one-way functions) as "basic" and build more powerful primitives (e.g., pseudorandom functions) out of them in inefficient ways; but in practice, powerful primitives are readily available and the so-called basic ones seem to be no easier to implement. In fact theorists deny themselves the capabilities of practical primitives which satisfy not only the strongest kinds of assumptions they like to make, but even have strengths which have not been defined or formalized.

In order to bring to practice some of the benefits of provable security, it makes sense to incorporate into our models objects which capture the properties that practical primitives really seem to possess, and view these objects as basic even if the assumptions about them are, from a theoretical point of view, very strong. This paper highlights the efficacy and potential of one such approach. The idea is a simple one: namely, provide all parties —good and bad alike— with access to a (public) random oracle; prove correct a protocol in this model; then replace the random oracle by an object like a hash function. We stress that the proof is in the random oracle model and the last step is heuristic in nature. It is a thesis of this paper that significant assurance benefits nonetheless remain.

The idea of such a paradigm builds on work of Goldreich, Goldwasser and Micali [20, 21] and Fiat-Shamir [14]. It is guided by many previous "unjustified" uses of hash functions. Finally, it incorporates viewpoints which, shared and verbally articulated by many members of our community, should be regarded as folklore. In this light, we view our contribution as follows. First, we raise the implicit philosophy behind the use of a random oracle to an explicitly articulated paradigm which we maintain brings significant benefits to practice. Second, we systematically apply the paradigm to diverse cryptographic problems to obtain efficient solutions. Third, we provide definitions and proofs to show that some of the previously "unjustified" uses of hash functions can find justification in the random oracle model. Finally, we suggest constructions of hash functions which we believe are appropriate to instantiate the random oracle. We proceed by describing the paradigm in further detail. For details on background and related work see Section 1.3.

1.1 The Random Oracle Paradigm

The aforementioned disparity between the theoreticians' and practioners' views on primitives is illustrated by the following example. Theorists view a one-way function as a basic object and build pseudo-random functions from them. But in practice, as indicated by Luby and Rackoff [30, 31], the DES provides a pseudorandom function of 64 bits to 64 bits. Ironically, if one needs a practical protocol for a one-way function, likely one would construct it *from* DES—thereby reducing the "simple" primitive to the "complex" one.

If one is trying to design efficient protocols, it makes more sense to start off making strong, realistic assumptions about the primitives that will be used. Based on the paragraph above, a pseudorandom function on 64-bit strings is an excellent starting point. As we describe below, it seems reasonable to adopt even more generous assumptions.

Powerful Primitives. Let us look at a second efficiently-computable primitive: the map h_2 defined by the MD5 algorithm [37] restricted to inputs of length ≤ 400, say.[1] One has expectations like these of this function: that it is hard to find an x such that $h_2(x) = x$; that it is hard to

[1] See Section 6 for why we prefer not to use MD5 itself.

find an x such that $h_2(x)$ has Hamming weight exceeding 120; that $f_a(x) = h_2(xa)$ is (in practice) a pseudorandom function family; etc. What really *is* this object? To date, there has been no satisfactory answer. That is, there is no formal definition which captures a large fraction of the nice properties this function seems to possess—and it is not clear that one can be found.

THE PARADIGM. Our answer to "what might a function like h_2 accomplish?" is to say that it can be thought of as a random function in the sense that it can be used in the following design methodology in the role of h. Suppose one has a protocol problem Π (the problem being "independent" of the primitive h.) In order to devise a good protocol P for Π:

(1) Find a formal definition for Π in the model of computation in which all parties (including the adversary) share a random oracle R.

(2) Devise an efficient protocol P for Π in this random oracle model.

(3) Prove that P satisfies the definition for Π.

(4) Replace oracle accesses to R by computation of h.

It is our *thesis* that this method, when properly carried out, leads to secure and efficient protocols. Indeed, protocols constructed under this paradigm have so far proven "secure" in practice. But we stress that all claims of provable security are claims made within the random oracle model, and instantiating the oracle with h is only a heuristic whose success we trust from experience.

Note that h cannot really be like a random function because it has a short description. In many ways, h is very different from a random oracle. This has not altered the success of the method.

We stress that the protocol problem Π and protocol P must be "independent" of the hash function we are to use. It is easy to construct unnatural problems or protocols whose description and goals depend explicitly on h so that the protocol is secure in the random oracle model but fails when the random oracle is instantiated with the hash function. The notion of "independence" will not be formalized in this paper.

INSTANTIATION. For the body of this paper, we assume a random oracle R from $\{0,1\}^*$ to $\{0,1\}^\infty$. We use such an oracle without further explanation to provide whatever random maps are convenient for describing a given protocol.

When instantiating a random oracle by a concrete function h, care must be taken first to ensure that h is adequately conservative in its design so as not to succumb to cryptanalytic attack, and second to ensure that h exposes no relevant "structure" attributable to its being defined from some lower-level primitive. Examples of both types of pitfalls are given in Section 6. As explained in that section, standard hash functions like MD5 and SHA don't by themselves make good replacements for a random oracles; but one doesn't have to look much further. Candidate instantiations include hash functions with their outputs truncated; hash functions with their input lengths restricted; and hash functions used in some nonstandard way, such as $h_3(x) = \mathrm{MD5}(xx)$. See Section 6.

1.2 Results

The results of this paper can can be divided into three kinds. First are new and efficient solutions for various cryptographic problems. Second are justifications of known heuristics. Third are some "theoretical" results in the random oracle model which our investigations have lead us to prove. In each case we provide protocols, theorems, and the new definitions appropriate to the random oracle setting.

EFFICIENT ENCRYPTION. Goals which are possible but impractical in the standard setting become practical in the random oracle setting. We illustrate with one example: public key encryption. In

what follows G: $\{0,1\}^* \to \{0,1\}^\infty$ is a random generator; k is the security parameter; H: $\{0,1\}^* \to \{0,1\}^k$ is a random hash function; f is a trapdoor permutation with inverse f^{-1}; $G(r) \oplus x$ denotes the bitwise XOR of x with the first $|x|$ bits of the output of $G(r)$; and "$\|$" denotes concatenation. For a concrete implantation, f might be squaring [42, 3] or RSA [38]. We suggest two schemes to encrypt efficiently in the random oracle model:

(1) Set $E^G(x) = f(r) \ \| \ G(r) \oplus x$ for a random value r from the domain of f.

(2) Set $E^{G,H}(x) = f(r) \ \| \ G(r) \oplus x \ \| \ H(rx)$ for a random value r from the domain of f.

Here x is the message to be encrypted, f is the recipient's public key, and f^{-1} is his secret key. For background, definitions, precise statements of results, and efficiency comparisons with known schemes see Section 3, but, briefly, what is argued there is the following: the first scheme achieves polynomial/semantic security as defined by [24]; the second is secure against chosen-ciphertext attack in the sense of [36] as well as non-malleable in the sense of [13]; and both are significantly more efficient than previous provably-secure schemes [24, 4, 34, 36, 11, 13] for the same goals.

Justification Of Known Heuristics. A variety of well-known "tricks" find formal justification by moving to the random oracle setting. (This does not mean that existing protocols can usually be justified by adopting a random oracle model; to the contrary, it appears to be more the exception than the rule.) We illustrate with the following pair of examples.

Popular signature schemes such as RSA are an instance of the following: for a trapdoor permutation f and hash function H the signature of message x is $f^{-1}(H(x))$. It is widely recognized that "usual" properties of a hash function such as collision-freeness don't suffice to make such a method a secure signature scheme. However for H a random hash function we show the scheme is secure against adaptive chosen message attack. See Section 4.

A heuristic to eliminate interaction in a zero-knowledge interactive proof, attributed to M. Blum,[2] is to have the prover essentially ask of himself the queries that a verifier would ask by computing these queries as the hash of the messages already exchanged between the parties. We show that this construction is provably secure in the random oracle model. Providing this proof has necessitated giving formal definitions for zero-knowledge in the random oracle model. See Section 5.

Theoretical Results. Generalizing the result just described, we show that *any* language that has an interactive proof can have its proof efficiently transformed into a non-interactive zero-knowledge one. The model of computation is that all parties —including cheating provers— are afforded only polynomially many queries to the random oracle. We also show that in the random oracle model, constant round, information theoretically secure function evaluation is possible.[3] Definitions and proofs of these results are omitted for lack of space.

1.3 Background and Related Work

The basic idea of proving correct a protocol in a model where the parties have a random oracle and then instantiating that oracle with an appropriate cryptographic primitive originates in [20, 21]. The cryptographic primitive suggested and constructed for this purpose by [20] is the pseudo-random function (PRF). For a PRF to retain its properties, however, the seed via which it is specified (and which enables its computation) must remain unknown to the adversary. Thus the applicability of the paradigm is restricted to protocols in which the adversary is denied access to the random

[2] Personal communication, via S. Micali and S. Rudich.

[3] In this application it does *not* suffice to replace the pseudorandom generator used in [1] by a random generator.

oracle.[4] Thus in many applications (and the ones of this paper in particular) PRFs don't suffice. Note, however, that when the setting permits instantiation of the oracle via PRFs, the resulting protocol can usually be proven correct in the standard model of computation under a standard complexity-theoretic assumption, something instantiation via hash functions as we suggest does not achieve.

The first work which explicitly adopts a *public* random oracle model —all parties, adversary included, can access the oracle— is that of Fiat and Shamir [14]. The authors use this model to turn an identification scheme into a digital signature scheme (without "totally" sacrificing rigor in the course of this transformation).

M. Blum's aforementioned idea of making interactive proofs non-interactive can be thought of as an extension of the Fiat-Shamir idea. An exciting recent result on computationally bounded checking, due to Micali [32], exploits in part this same technique.

Impagliazzo and Rudich [27] model one-way functions as random oracles. They do this in order to show that proving the existence of a secret key exchange protocol given a black box one-way function is as hard as separating P from NP. They also use random oracles for positive results; among these, they formalize and prove the existence of a private key cryptosystem in the random oracle model.

Concurrent and independent of our work, Leighton and Micali [28] view hash functions as public random oracles to justify the security of a new, efficient signature scheme. They use the random oracle model to define and prove exact, non-asymptotic security. In another paper [29] the same authors use hash functions viewed as random oracles to give new secret key exchange schemes.

Because of the breadth of topics in this paper, history specific to to a particular goal is summarized in the section that describes that goal.

1.4 Future Directions

Brought out in only a limited way in the current work, and fully in [28], is the fact that the random oracle model facilitates giving definitions and results precise in the sense of avoiding complexity theory and asymptotics. It is feasible and desirable to make our results precise in this sense. A typical theorem would express the advantage an adversary gains in terms of the number of oracle queries which she makes.

We know no complexity-theoretic assumption which does a good job of capturing all the nice properties of a public random oracle. Is there a way to extend the [20] notion of a pseudorandom function family to an equally useful and compelling notion which involves no hidden randomness?

2 Preliminaries

NOTATION. $\{0,1\}^*$ denotes the space of finite binary strings and $\{0,1\}^\infty$ denotes the space of infinite ones. Strings are finite unless we say otherwise. We denote by $a\|b$, or just ab, the string which is the concatenation of strings a and b. The empty string is denoted Λ. A polynomial time algorithm is one which runs in time polynomial in its first argument. "PPT" stands for "probabilistic, polynomial time." A function $\epsilon(k)$ is *negligible* if for every c there exists a k_c such that $\epsilon(k) \leq k^{-c}$ for every $k \geq k_c$. A function is said to be *non-negligible* if it is not negligible. We'll use the notation "$k^{-\omega(1)}$" to mean the class negligible functions or a particular anonymous function in this class.

[4] That is, the adversary is denied *direct* access to the oracle. A particular problem might permit the adversary indirect access to the oracle via her interaction with the good parties.

Notation for probabilistic algorithms, spaces and experiments follows [26]. If A is a probabilistic algorithm then, for any inputs $x, y, \ldots$ the notation $A(x, y, \ldots)$ refers to the probability space which to the string σ assigns the probability that A, on input $x, y, \ldots$, outputs σ. If S is a probability space we denote its support (the set of elements of positive probability) by $[S]$. If S is a probability space then $x \leftarrow S$ denotes the algorithm which assigns to x an element randomly selected according to S. In the case that $[S]$ consists of only one element e we might also write $x \leftarrow e$. For probability spaces $S, T, \ldots$, the notation $\Pr[\, x \leftarrow S;\ y \leftarrow T;\ \cdots : p(x, y, \cdots)\,]$ denotes the probability that the predicate $p(x, y, \cdots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc. Let f be a function. We extend this notation of [26] to define also probability spaces and algorithms via experiments. For example $\{\, x \leftarrow S;\ y \leftarrow T;\ \cdots :\ f(x, y, \cdots)\,\}$ denotes the probability space which to the string σ assigns the probability $\Pr[\, x \leftarrow S;\ y \leftarrow T;\ \cdots :\ \sigma = f(x, y, \cdots)\,]$. And

$$\langle\, a, b, \ldots :\ x \leftarrow S;\ y \leftarrow T;\ \cdots :\ f(a, b, \cdots, x, y, \cdots)\,\rangle$$

denotes the algorithm which on inputs $a, b, \ldots$ runs the sequence of experiments $x \leftarrow S$, $y \leftarrow T$,..., and outputs $f(a, b, \cdots, x, y, \cdots)$.

Oracles. For convenience, a random oracle R is a map from $\{0,1\}^*$ to $\{0,1\}^\infty$ chosen by selecting each bit of $R(x)$ uniformly and independently, for every x. Of course no actual protocol uses an infinitely long output, this just saves us from having to say how long "sufficiently long" is. We denote by 2^∞ the set of all random oracles.

The letter "R" will denote the "generic" random oracle, while $G\colon\ \{0,1\}^* \to \{0,1\}^\infty$ will denote a random generator and $H\colon\ \{0,1\}^* \to \{0,1\}^k$ a random hash function. Whenever there are multiple oracles mentioned, all of these are independently selected. Via all sorts of natural encodings, a single random oracle R can be used to provide as many independent random oracles as one wants.

As usual the oracles provided to an algorithm are indicated by superscripts. Sometimes the oracle is understood and omitted from the notation.

Trapdoor Permutations. Following [26], a *trapdoor permutation generator* is a PPT algorithm $\mathcal{G}_*$ which on input 1^k outputs (the encoding of) a triple of algorithms (f, f^{-1}, d). The first two are deterministic and the last is probabilistic. We require that $[d(1^k)]$ be a subset of $\{0,1\}^k$ and that f, f^{-1} be permutations on $[d(1^k)]$ which are inverses of one another. We require that there exist a polynomial p such that f, f^{-1} and d are computable in time $p(k)$, and that for all nonuniform polynomial time adversaries M,

$$\varepsilon(k) = \Pr[\, (f, f^{-1}, d) \leftarrow \mathcal{G}_*(1^k);\ x \leftarrow d(1^k);\ y \leftarrow f(x) :\ M(f, d, y) = x\,]$$

is negligible. As mentioned before, squaring modulo an appropriate composite number [42, 3], variations of it [26], or RSA [38] are good examples of trapdoor permutations. Call a trapdoor permutation generator $\mathcal{G}_*$ *uniform* if for all k and all $(f, f^{-1}, d) \in [\mathcal{G}(1^k)]$ it is the case that d is the uniform distribution on $\{0,1\}^k$.

3 Encryption

We have relied on definitional work in [24, 33, 19, 18, 34, 13]. For simplicity we consider adversaries who are nonuniform (polynomial time) algorithms, possibly probabilistic; extensions to the uniform case can be made following [18].

Encryption. We extend the notion of public key encryption [12] to the random oracle model. The scheme is specified by a PPT *generator* $\mathcal{G}$ which takes a security parameter 1^k and outputs a

pair of probabilistic algorithms (E, D) which are called the encryption and decryption algorithms respectively and which run in time bounded by $\mathcal{G}$'s time complexity. A user U runs $\mathcal{G}$ to get (E, D) and makes the former public while keeping the latter secret. To encrypt message x anyone can compute $y \leftarrow E^R(x)$ and send it to U; to decrypt ciphertext y user U computes $x \leftarrow D^R(y)$. We require $D^R(E^R(x)) = x$ for all x and assume for simplicity that $D^R(y) = 0$ if y is not the encryption under E^R of any string x.

3.1 Polynomial Security

BACKGROUND. The "basic" security goal of public key encryption finds its formalization in Goldwasser and Micali's (equivalent) notions of polynomial and semantic security [24]. If B_f denotes a *hard core predicate* for f (cf. [5, 43, 23]) then security in the sense of [24] can be achieved by setting $E(x) = f(r_1) \parallel \ldots \parallel f(r_{|x|})$ where each r_i is randomly chosen from the domain of f with the restriction that $B_f(r_i) = x_i$. This yields an encryption of length $O(k \cdot |x|)$, which requires $O(|x|)$ evaluations of f to encrypt and $O(|x|)$ evaluations of f^{-1} to decrypt, which is not practical. A more efficient construction of Blum and Goldwasser [4] yields encryptions of size $O(|x| + k)$ requiring $O(|x|)$ modular squarings operations to encrypt and $O(1)$ modular exponentiations plus $O(|x|)$ modular squaring to decrypt, which is still expensive. Practioners often embed the message x into an otherwise random value r_x and then set $E(x) = f(r_x)$. For example, this is exactly what [39] specifies. The embeddings used in practice usually do not guarantee that x is as hard to find as r_x, let alone that all properties of x are hidden.

DEFINITION. We adapt the notion of polynomial security [24] to the random oracle model. (A similarly-extended notion for semantic security remains equivalent.) A CP-adversary (chosen-plaintext adversary) A is a pair of nonuniform polynomial time algorithms (F, A_1), each with access to an oracle. For an encryption scheme $\mathcal{G}$ to be secure in the random oracle model we require that for any CP-adversary $A = (F, A_1)$,

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

Note that the oracle used to encrypt and decrypt is given to the adversary who tries to distinguish the encryption of strings m_0 and m_1, so, for example, a hash $H(x)$ with H derived from R could most certainly *not* appear in the secure encryption of a string x.

ENCRYPTION BY $E(x) = f(r) \parallel G(r) \oplus x$. To specify our encryption scheme, let $\mathcal{G}_*$ be a trapdoor permutation generator and let G: $\{0,1\}^* \to \{0,1\}^\infty$ be a random generator. On input 1^k our generator $\mathcal{G}$ runs $\mathcal{G}_*$ to get (f, f^{-1}, d). It sets E^G to the following algorithm:

$$E^G \leftarrow \langle\, x :\ r \leftarrow d(1^k) :\ f(r) \ \parallel\ G(r) \oplus x \,\rangle$$

where $G(r) \oplus x$ denotes the XOR of the first $|x|$ bits of $G(r)$ with x. Of course the decryption function is then $D^G(ys) = s \oplus G(f^{-1}(y))$.

THEOREM. In Appendix A we show that the above scheme is polynomially secure in the random oracle model.

EFFICIENCY. We achieve encryption size $|x| + k$. Besides hashing of negligible cost, encryption needs one application of f and decryption needs one application of f^{-1}. Setting f to squaring this means one modular squaring to encrypt and one modular exponentiation to decrypt. This is much more efficient than the scheme of [4] discussed above.

3.2 Chosen Ciphertext Security

BACKGROUND. Naor and Yung [34] provided a definition of chosen ciphertext security and the first scheme to provably achieve it. Rackoff and Simon [36] suggested a stronger notion and a corresponding solution; another solution was given by De Santis and Persiano [11]. The last two exploit proofs of knowledge, as suggested earlier by [17, 6]. All known schemes provably secure under standard assumptions rely on non-interactive zero-knowledge proofs [7, 16] and are prohibitively inefficient. Damgård [10] suggests an efficient scheme to achieve the definition of [34], but this scheme is not proven to achieve the definition of [34] and it does not achieve the one of [36] which we are interested in. A scheme of Zheng and Seberry [44] closely related to ours will be discussed later.

DEFINITION. We adapt the definition of [36] to the random oracle setting. An RS-adversary ("Rackoff–Simon adversary") A is a pair of nonuniform polynomial time algorithms $A = (F, A_1)$, each with access to an oracle R and a black box implementation of D^R. F's job is to come up with a pair of (equal length) messages m_0 and m_1 such that if A_1 is given the encryption α of a random one of these, A_1 won't be able to guess well which one as long as A_1 is not allowed to ask α of the decryption oracle. Formally, A_1 is forbidden from asking an oracle query equal to its final argument. Encryption scheme $\mathcal{G}$ is secure against RS-attack if for each RS-adversary $A = (F, A_1)$,

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

ENCRYPTION BY $E(x) = f(r) \parallel G(r) \oplus x \parallel H(rx)$. It is easy to see that the scheme of the previous section is not secure against RS-attack. We now specify an efficient scheme which is. 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 a random hash function, independently derived from the random oracle. The generator $\mathcal{G}$ of our scheme runs $\mathcal{G}_*$ to get (f, f^{-1}, d). It sets $E^{G,H}$ to the following algorithm:

$$E^{G,H} \leftarrow \langle\, x :\ r \leftarrow d(1^k) :\ f(r) \ \parallel\ x \oplus G(r) \ \parallel\ H(rx) \,\rangle\ .$$

To decrypt a string y, parse it to $a \parallel w \parallel b$ for $|a| = |b| = k$ and define $D^{G,H}(y)$ as $w \oplus G(f^{-1}(a))$ if $H(f^{-1}(a) \parallel w \oplus G(f^{-1}(a))) = b$, and 0 otherwise.

THEOREM. In Appendix A we show that the above scheme is secure against chosen-ciphertext attack.

EFFICIENCY. We achieve encryption size $O|x| + 2k$. Besides hashing of negligible cost, encryption needs one application of f and decryption needs one application of f^{-1}. Setting f to squaring this means one modular squaring to encrypt and one modular exponentiation to decrypt. This is more efficient than any of the schemes discussed above.

Translated into the random oracle model and our notation, the scheme of Zheng and Seberry [44] is $E^*(x) = f(r) \parallel (G(r) \oplus (xH(x)))$. This scheme is as efficient as ours, and we believe it has the same security properties. Thus, the random oracle model serves to justify the construction of [44].

3.3 Non-Malleability

BACKGROUND. The notion of non-malleability was introduced by Dolev, Dwork and Naor [13]. Informally, an encryption scheme is non-malleable if you cannot, by witnessing an encryption of a

string x, produce the encryption of a related string x'. For example, given the encryption of x you shouldn't be able to produce the encryption of $\overline{x}$. The notion extends polynomial security, and in particular the latter is implied by the former. A construction of non-malleable schemes is given in [13]. However, this construction is completely impractical, involving huge public keys, computation of multiple signatures, and many non-interactive zero knowledge proofs.

DEFINITION. We adapt to the random oracle setting the definition of [13]. An *interesting* relation $\rho^R_{E,\pi} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ must satisfy $\rho^R_{E,\pi}(x,x) = \rho^R_{E,\pi}(x,0^i) = 0$ for every $x \in \{0,1\}^*$, $i \in \mathsf{N}$, $R \in 2^\infty$, and $E, \pi \in \{0,1\}^*$; furthermore, ρ must be computable by a polynomial time Turing machine $M^R(x,y,E,\pi)$. An M-adversary ("malleability adversary") $\mathcal{A}$ is a pair (F,A) of non-uniform probabilistic polynomial time algorithms, each with access to an oracle R. When F runs it outputs the description of an algorithm π which also takes an oracle and which runs with time complexity no greater than that of F. For an encryption scheme $\mathcal{G}$ to be *non-malleable* we require that for every interesting relation ρ and every M-adversary (F,A) there exists a (nonuniform) polynomial time A_* such that $|\varepsilon(k) - \varepsilon_*(k)|$ is negligible, where

$$\begin{aligned}
\varepsilon(k) &= \Pr[\, R \leftarrow 2^\infty;\ (E,D) \leftarrow \mathcal{G}(1^k);\ \pi \leftarrow F^R(E);\ x \leftarrow \pi^R(1^k); \\
&\qquad \alpha \leftarrow E^R(x);\ \alpha' \leftarrow A^R(E,\pi,\alpha) :\ \rho^R_{E,\pi}(x, D^R(\alpha')) = 1\,] \\
\varepsilon_*(k) &= \Pr[\, R \leftarrow 2^\infty;\ (E,D) \leftarrow \mathcal{G}(1^k);\ \pi \leftarrow F^R(E);\ x \leftarrow \pi^R(1^k); \\
&\qquad \alpha'_* \leftarrow A^R_*(E,\pi) :\ \rho^R_{E,\pi}(x, D^R(\alpha'_*)) = 1\,]
\end{aligned}$$

See [13] for explanations on the intuition underlying this definition, including the restriction on the relation ρ.

ENCRYPTION BY $E(x) = f(r) \parallel G(r) \oplus x \parallel H(rx)$. The encryption scheme is the same as that of the previous section.

THEOREM. In Appendix A we show that the above scheme is non-malleable.

4 Signatures

DEFINITION. We extend the definitions of [26] to the random oracle setting. A digital signature scheme is a triple $(\mathcal{G}, \mathsf{Sign}, \mathsf{Verify})$ of polynomial time algorithms, called the *generator, signing algorithm,* and *verifying algorithm,* respectively. The first two are probabilistic and the last two have access to the random oracle. On input 1^k, the generator produces a pair $(\mathrm{PK}, \mathrm{SK})$ of matching public and secret keys. To sign message m compute $\sigma \leftarrow \mathsf{Sign}^R(\mathrm{SK}, m)$; to verify (m,σ) compute $\mathsf{Verify}^R(\mathrm{PK}, m, \sigma) \in \{0,1\}$. It must be the case that $\mathsf{Verify}^R(\mathrm{PK}, m, \sigma) = 1$ for all $\sigma \in [\mathsf{Sign}^R(\mathrm{SK}, m)]$. An S-adversary ("signing adversary") is a (nonuniform) polynomial-time algorithm F with access to R and a signing oracle. The output of F is a pair (m,σ) such that m was not queried of the signing oracle. The signature scheme is secure if for every S-adversary F the function $\varepsilon(k)$ defined by

$$\Pr[\, R \leftarrow 2^\infty;\ (\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(1^k);\ (m,\sigma) \leftarrow F^{R, \mathsf{Sign}^R(\mathrm{SK},\cdot)}(\mathrm{PK}) :\ \mathsf{Verify}^R(\mathrm{PK}, m, \sigma) = 1\,]$$

is negligible. We say that F is *successful* if its output (m,σ) satisfies $\mathsf{Verify}^R(\mathrm{PK}, m, \sigma) = 1$.

PROTOCOL. Fix a trapdoor permutation generator $\mathcal{G}_*$. For simplicity assume it is uniform; see below for how to patch things for standard ones. Let $H\colon\ \{0,1\}^* \to \{0,1\}^k$ denote as usual a random hash function. Let $\mathcal{G}$ on input 1^k compute $(f, f^{-1}, d) \leftarrow \mathcal{G}_*(1^k)$, set $\mathrm{PK} = f$ and $\mathrm{SK} = f^{-1}$, and

output $(\mathrm{PK}, \mathrm{SK})$. The signature scheme is $(\mathcal{G}, \mathsf{Sign}^H, \mathsf{Verify}^H)$ where $\mathsf{Sign}^H(f^{-1}, m) = f^{-1}(H(m))$ and $\mathsf{Verify}^H(f, m, \sigma)$ is 1 if and only if $f(\sigma) = H(m)$. In other words just the "classical" method of signing with the aid of a hash function.

Uniformity: A Technicality. Standard trapdoor permutations (squaring based or RSA) are not uniform and the scheme must be patched to handle them. There are many ways of patching. RSA, and squaring as defined in [26], have dense domains in which membership can be efficiently tested. So to sign m we could modify the scheme to compute $H(1 \parallel m), H(2 \parallel m), \ldots$ until a member $y = H(i \parallel m)$ of the domain is found and then return $(i, f^{-1}(y))$. Verification is defined in the obvious way. Another alternative for these functions is apply the construction of [2, Section 4.2] to make them uniform. The squaring functions defined in [42, 3] don't have efficiently testable domains but various patches can nonetheless be made. In fact it isn't even necessary for the function to be a permutation; Rabin's squaring function [35] can be patched to work too.

Security. The proof that the above scheme is secure against adaptive chosen message attack appears in Appendix B.

5 Zero Knowledge

We provide definitions for zero-knowledge (ZK) proofs in the random oracle and then show how ZK interactive proofs can be made non-interactive in this model. The transformation is efficient, so that we get non-interactive ZK proofs of complexity equal to interactive ZK ones.

5.1 Definitions

Definitions for zero-knowledge in the random oracle model involve a little more than simply "relativizing" the standard ones. What follows extends the formulation in the usual interactive setting [25] as well as the formulation in the common random string model [6, 7].

Setting. For simplicity we discuss proofs for a language $L \in \mathrm{NP}$. Fix a NP relation ρ defining L; a witness for the membership of x in L means a string w satisfying $\rho(x, w) = 1$. A witness selector is a function W which on any input $x \in L$ returns a witness for the membership of x in L.

A verifier is polynomial time function V which given common input x, conversation $\kappa \in \{0,1\}^*$ so far, and a (private) random tape $r \in \{0,1\}^\infty$ returns $V(x, \kappa, r)$ which is either the next message to the prover or a bit indicating his decision to either accept or reject. A prover is a PPT [5] function P which given the common input x, conversation κ so far, and auxiliary input a returns the next message $P_a(x, \kappa)$ to the verifier. (When $x \in L$ the auxiliary input is a witness to this fact, and otherwise it is the empty string). In the random oracle model both prover and verifier take also this oracle.

For any oracle R denote by $\mathtt{conv}(V^R, P_a^R, x, r)$ the space of all (transcripts of) conversations between P_a^R and V^R when the common input is x and V's random tape is $r \in \{0,1\}^\infty$. Denote by $\mathtt{ACC}_V(\kappa, r) \in \{0,1\}$ the verifier's decision on whether or not to accept. Let

$$\mathtt{ACC}(P_a, V, x) = \Pr[\, R \leftarrow 2^\infty;\ r \leftarrow \{0,1\}^\infty;\ \kappa \leftarrow \mathtt{conv}(V^R, P_a^R, x, r) :\ \mathtt{ACC}_V(\kappa, r) = 1 \,]$$

[5] In principle the results in the random oracle model require us to restrict only the number of oracle calls, not the running time of the prover. But at time of instantiation with hash functions running time should be restricted anyway so we make the assumption straight away. Thus we are in the "argument" model of [9].

denote the probability that V accepts in an interaction with P_a on common input x. In proofs and protocols we'll often abuse notation and work only with whatever prefixes of the infinite string r are relevant.

PROOF SYSTEMS. We say that (P, V) is an interactive proof for L, in the random oracle model and with error $\epsilon(n)$, if $\epsilon(n) \leq 1/2$ and the following two conditions hold. The *completeness* condition asks that if $x \in L$ then for all witnesses w to the membership of x in L it is the case that $\texttt{ACC}(V, P_w, x) = 1$. The *soundness* condition asks that for all PPT $\widehat{P}$ and sufficiently long x it is the case that $\texttt{ACC}(\widehat{P}_\Lambda, V, x) \leq \epsilon(|x|)$.

VIEWS. To define zero-knowledge the view of the verifier is first updated to include the random oracle; we define

$$\texttt{rview}(V, P_a, x) = \{\ R \leftarrow 2^\infty;\ r \leftarrow \{0,1\}^\infty;\ \kappa \leftarrow \texttt{conv}(V^R, P_a^R, x, r) :\ (\kappa, r; R)\ \}\ .$$

SIMULATORS. Since the random oracle is part of the view, it must also be part of the output of the simulator; i.e. the simulator is allowed to construct a "simulation" of the oracle. This is analogous to non-interactive zero-knowledge [6, 7] where the simulator is allowed to construct and output a "simulation" of the common random string. However, the random oracle is an infinite object, and so we can't ask the simulator to output it. Instead we allow the simulator to prescribe a small (polynomial sized) piece of the oracle, and have the rest "magically" filled at random. Formally, a simulator is a PPT algorithm which on any input x outputs a triple (κ, r', T) where $T = (x_1, y_1), \ldots, (x_t, y_t)$ is a sequence of pairs of strings with the property that $x_1, \ldots, x_t$ are distinct. The *random oracle completion* operation ROC takes as input T and returns an oracle R which is random subject to the constraint that $R(x_i)$ is prefixed by y_i for all $i = 1, \ldots, t$.[6] It is convenient to similarly define the *random string completion* operation RSC which takes a string $r' \in \{0,1\}^*$ and appends an infinite sequence of random bits. We define the *completion* of $S(x)$ to be the probability space

$$S^c(x) = \{\ (\kappa, r', T) \leftarrow S(x);\ R \leftarrow \texttt{ROC}(T);\ r \leftarrow \texttt{RSC}(r') :\ (\kappa, r; R)\ \}\ .$$

DISTINGUISHERS. A distinguisher is a polynomial sized oracle circuit family $D = \{D_x\}_{x \in L}$. Write $D_x^R(\kappa, r)$ for the output of circuit D_x when given oracle R and inputs κ, r.[7] Let $\texttt{diff}_D(S^c(x), \texttt{rview}(V, P_a, x))$ equal

$$|\Pr[\ (\kappa, r; R) \leftarrow S^c(x) :\ D_x^R(\kappa, r) = 1\ ] - \Pr[\ (\kappa, r; R) \leftarrow \texttt{rview}(V, P_a, x) :\ D_x^R(\kappa, r) = 1\ ]\ .$$

ZERO-KNOWLEDGE. We say that a simulator S is a P-simulator for a verifier $\widehat{V}$ over L if for every distinguisher D, every witness selector W, every constant d and all sufficiently long $x \in L$ it is the case that

$$\texttt{diff}_D(S^c(x), \texttt{rview}(V, P_{W(x)}, x)) \ <\ |x|^{-d}\ .$$

We say that P defines a (computational) ZK protocol over L in the random oracle model if for every verifier $\widehat{V}$ there exists a P-simulator for $\widehat{V}$ over L. Statistical ZK can be defined analogously. (P, V) is a ZK proof for L, in the random oracle model and with error ϵ, if it is a proof system for L with error ϵ and P defines a ZK protocol over L.

[6] It is understood that the operation refers to the oracle in use rather than the "generic" underlying one, so that if we are using a random hash function H then what is returned is an H satisfying this constraint, etc.

[7] Here r will be an infinite string, and giving κ, r as input to D_x means the latter will look at only a finite prefix.

MULTI-THEOREM PROOFS. In applications it is important that we be able to prove polynomially many, adaptively chosen theorems in zero-knowledge, as for zero-knowledge in the common random string model. For simplicity we have stuck above to the one theorem case; in the final paper we will present the general definitions.

PROOFS OF KNOWLEDGE. In the final paper we will also define proofs of knowledge in the random oracle model and show how to construct efficient, non-interactive zero-knowledge proofs of knowledge.

5.2 Protocol

THE PROBLEM. Let (P', V') be a ZK proof for $L \in \text{NP}$, in the standard (i.e. random oracle devoid) model, achieving error probability $1/2$. Let $k(n) = \omega(\log n)$ be given. We want a non-interactive ZK proof (P, V) in the random oracle model which achieves error $\epsilon(n) = 2^{-k(n)}$ while increasing computing time and communicated bits by a factor of at most $O(k(n))$.

SIMPLIFYING ASSUMPTIONS. Like most such ZK proofs assume (P', V') is three moves: $P'_w \rightarrow V' : \alpha$ followed by $V' \rightarrow P'_w : b$ followed by $P'_w \rightarrow V' : \beta$. Here b is a random bit (the first one on the random tape of V' which we now think of as just this bit) and w is the auxiliary input to P'. The message α consists of a set of envelopes and has size $n^{\Theta(1)}$. Some subset of these envelopes is opened according to challenge b, and for any string α there is exactly one value $b \in \{0,1\}$ for which there exists a β such that $\texttt{ACC}_{V'}(\alpha b \beta, b) = 1$. The zero-knowledge is captured by an algorithm S' which given x, b outputs $\alpha b \beta$ such that $\texttt{ACC}_{V'}(\alpha b \beta, b) = 1$ and for any witness selector W the following ensembles are computationally indistinguishable: $\{\, b \leftarrow \{0,1\};\ \alpha b \beta \leftarrow S'(x, b) :\ (\alpha b \beta, b)\,\}_{x \in L}$ and $\{\, \alpha \leftarrow P'_{W(x)}(x, \Lambda);\ b \leftarrow \{0,1\};\ \beta \leftarrow P'_{W(x)}(x, \alpha b) :\ (\alpha b \beta, b)\,\}_{x \in L}$.

THE TRANSFORMATION. Let H: $\{0,1\}^* \rightarrow \{0,1\}^{2k}$ be a random hash function. The new prover P^H_w computes $\alpha_1 \leftarrow P'_w(x, \Lambda);\ \ldots;\ \alpha_{2k} \leftarrow P'_w(x, \Lambda)$; sets b'_i to the i-th bit of $H(\alpha_1 \ldots \alpha_{2k})$; computes $\beta_1 \leftarrow P'_w(x, \alpha_1 b'_1);\ \ldots;\ \beta_{2k} \leftarrow P'_w(x, \alpha_{2k} b'_{2k})$; and sends $(\alpha_1, \ldots, \alpha_{2k}, \beta_1, \ldots, \beta_{2k})$ to V^H. V^H sets b_i to the i-th bit of $H(\alpha_1 \ldots \alpha_{2k})$ and accepts iff $\texttt{ACC}_{V'}(\alpha_i b_i \beta_i, b_i) = 1$ for all i. The fact that the new protocol is non-interactive and as efficient as claimed is clear.

(P, V) IS A ZK PROOF SYSTEM WITH ERROR $2^{-k(n)}$. Completeness is clear. We can show that if $\widehat{P}^H$ makes $T(n)$ oracle queries then $\texttt{ACC}(\widehat{P}_\Lambda, V, x) \leq T(n) \cdot 2^{-2k(n)}$ which is at most $2^{-k(n)}$ for sufficiently long n. For ZK the lack of interaction implies we only need to simulate the view of the honest verifier V, and the corresponding simulator S is as follows. Given $x \in L$ algorithm S chooses $b_1 \leftarrow \{0,1\};\ \ldots;\ b_{2k} \leftarrow \{0,1\}$. Now for each $i = 1, \ldots, 2k$ it lets $\alpha_i b_i \beta_i \leftarrow S'(x, b_i)$. It sets $T = (\alpha_1 \ldots \alpha_{2k}, b_1 \ldots b_{2k})$ and outputs (c, Λ, T). The random oracle completion operation applied to T results in a map H: $\{0,1\}^* \rightarrow \{0,1\}^{2k}$ which is random subject to the constraint that $H(\alpha_1 \ldots \alpha_{2k}) = b_1 \ldots b_k$. Now based on our assumption about S' we can work through the definitions and check that S is a P-simulator for V over L. We omit the details.

COMMENT. Consider the protocol resulting from instantiating the random oracle in the above with a hash function. It is in the standard (random oracle devoid) model, but note we are not claiming it is ZK in the usual sense. Indeed the result of Goldreich and Krawczyk [22] indicates that it is unlikely to be ZK in the usual sense: they show that assuming NP is not in BPP, at least four moves are necessary to give a (black box simulatable, computational) ZK, negligible error proof for an NP-complete language in the standard model.

6 Instantiation

Expanding on the discussion in Section 1.1, here we provide further guidance in instantiating random oracles with primitives like hash functions.

First and foremost, it is not necessary (or desirable) to pay attention to the particulars of the target protocol whose random oracles are being instantiated. All that matters it is how many oracles are used and what are their input/output length requirements. Our thesis is that an appropriate instantiation for a random oracle ought to work for any protocol which did not intentionally frustrate our method by anticipating the exact mechanism which would instantiate its oracles.

A significant amount of care must be taken in choosing a concrete function h to instantiate an oracle. Let us begin with some examples of some things that *don't* work.

Consider first the map MD5. This function does not make a suitable replacement for a random oracle since [41] has observed that for any x there is a y such that for any z, MD5(xyz) can be easily computed given only $|x|$, MD5(x), and z. Structure like this shows up in applications; in particular, [41] points out that this means MD5(ax) cannot be used as a message authentication code of string x under key a.

Trying to overcome difficulties by avoiding a "structured" operation like MD5, one might prefer a "lower level" primitive such as its compression function, μ: $\{0,1\}^{640} \to \{0,1\}^{128}$. This too does not make a suitable replacement for a random oracle, as [8] has demonstrated that collisions can be efficiently found in this map.

Although standard hash functions are too structured to make good random oracles (as illustrated above), one doesn't have to look much further; natural candidates include constructs like the following, or combinations of them:

(1) A hash function with its output truncated or folded in some manner; e.g., $h_1(x)$ = the first 64 bits of MD5(x).

(2) A hash functions with its input lengths suitably restricted; e.g., $h_2(x)$ = MD5(x), where $|x| \leq 400$.

(3) A hash function used in some nonstandard way; e.g., $h_3(x) = \text{MD5}(xx)$.

(4) The "first block compression function" of a cryptographic hash function, e.g., h_4: $\{0,1\}^{512} \to \{0,1\}^{128}$ being the compression of the 512 bit x, when MD5(x) is computed.

As an example, suppose one settles on the (purely heuristic) choice of a map h': $\{0,1\}^{256} \to \{0,1\}^{64}$ defined by $h'(x)$ = the first 64 bits of $h_4((xx) \oplus C)$, for a randomly chosen 512-bit constant C.[8] To extend the domain and range as needed in a given application, one might first define $h''(x) = h'(x\langle 0\rangle) \| h'(x\langle 1\rangle) \| h'(x\langle 2\rangle) \| \cdots$ where $|x| = 224$ and $\langle i\rangle$ is the encoding of i into 64 bits. Next, one 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. Now let $x' = x'_1 \ldots x'_n$, where $|x'_i| = 128$ and define $h(x) = h''(x'_0\langle 0\rangle) \oplus h''(x'_1\langle 1\rangle) \oplus \cdots \oplus h''(x'_n\langle n\rangle)$ yielding a map which, for all practical purposes, takes h: $\{0,1\}^* \to \{0,1\}^\infty$. Of course there are lots of other equally simple ways to instantiate a random oracle; this was only an example.

[8] Choosing C at instantiation-time ensures that the algorithmic goal is "independent" of its choice of oracle; it "separates" the instantiation of the random oracles used by different applications; and it provides a simple means of creating multiple "independent" random oracles.

7 Conclusion

The protocols used in practice have almost always been designed by an iterated process of positing a concrete protocol, searching for a successful attack, finding one, and attempting to close it. This method has not worked well. By an insistence on defining our goals and provably achieving them, modern cryptography offers more to practice than any specific set of results; it is a methodology beyond the process of iterated design to solve poorly specified tasks. Although when using our paradigm one "only" ends up with a result that says "this protocol is secure to the extent that h instantiates a random oracle," still, one has achieved very much more than declaring a protocol sound because no one has as yet come up with a successful attack.

Acknowledgments

Early discussions with Bob Blakley on the license server problem [40] helped crystalize our idea. We got useful suggestions and references from Oded Goldreich, Birgit Pfitzmann, and Steven Rudich. Finally, thanks to the members of the ACM program committee for all of their comments.

Work done while the first author was at the IBM T.J. Watson Research Center, New York, and second author was at IBM Austin.

References

[1] D. Beaver, S. Micali and P. Rogaway, "The round complexity of secure protocols," Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[2] M. Bellare and S. Micali, "How to sign given any trapdoor permutation," *JACM* Vol. 39, No. 1, 214-233, January 1992.

[3] L. Blum, M. Blum and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing* Vol. 15, No. 2, 364-383, May 1986.

[4] M. Blum and S. Goldwasser, "An efficient probabilistic public-key encryption scheme which hides all partial information," *Advances in Cryptology – Crypto* 84 *Proceedings*, Lecture Notes in Computer Science Vol. 196, R. Blakely ed., Springer-Verlag, 1984.

[5] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing*, Vol. 13, No. 4, 850-864, November 1984.

[6] M. Blum, P. Feldman and S. Micali, "Non-interactive zero knowledge and its applications," Proceedings *of the* 20th *Annual Symposium on Theory of Computing*, ACM, 1988.

[7] M. Blum, A. De Santis, S. Micali and G. Persiano, "Non-interactive zero-knowledge proof systems," *SIAM Journal on Computing*, **20**(4), 1084-1118 (December 1991).

[8] B. den Boer and A. Bosselaers, "Collisions for the compression function of MD5," *Advances in Cryptology – Eurocrypt* 93 *Proceedings*, Lecture Notes in Computer Science Vol. 765, T. Helleseth ed., Springer-Verlag, 1993.

[9] G. Brassard, D. Chaum and C. Crépeau, "Minimum disclosure proofs of knowledge," *JCSS* Vol. 37, No. 2, 156–189, October 1988.

[10] I. Damgård, "Towards practical public key cryptosystems secure against chosen ciphertext attacks," *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[11] A. De Santis and G. Persiano, "Zero-knowledge proofs of knowledge without interaction" Proceedings *of the* 33rd *Symposium on Foundations of Computer Science*, IEEE, 1992.

[12] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, 644-654 (November 1976).

[13] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography," Proceedings *of the* 23rd *Annual Symposium on Theory of Computing*, ACM, 1991.

[14] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," *Advances in Cryptology – Crypto* 86 *Proceedings*, Lecture Notes in Computer Science Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986.

[15] U. Feige, A. Fiat and A. Shamir, "Zero knowledge proofs of identity," *Journal of Cryptology,* Vol. 1, pp. 77–94 (1987).

[16] U. Feige, D. Lapidot, and A. Shamir, "Multiple non-interactive zero-knowledge proofs based on a single random string," Proceedings *of the* 31st *Symposium on Foundations of Computer Science*, IEEE, 1990.

[17] Z. Galil, S. Haber and M. Yung, "Symmetric public key cryptosystems," manuscript, July 1989.

[18] O. Goldreich, "A uniform complexity treatment of encryption and zero-knowledge," *Journal of Cryptology,* Vol. 6, pp. 21-53 (1993).

[19] O. Goldreich, "Foundations of cryptography," Class notes, Spring 1989, Technion University.

[20] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM,* Vol. 33, No. 4, 792–807, (1986).

[21] O. Goldreich, S. Goldwasser and S. Micali, "On the cryptographic applications of random functions," *Advances in Cryptology – Crypto* 84 *Proceedings*, Lecture Notes in Computer Science Vol. 196, R. Blakely ed., Springer-Verlag, 1984.

[22] O. Goldreich and H. Krawczyk, "On the composition of zero knowledge proof systems," ICALP 90 *Proceedings*, Lecture Notes in Computer Science Vol. 443 , M. Paterson ed., Springer-Verlag, 1990.

[23] O. Goldreich and L. Levin, "A hard predicate for all one-way functions," Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[24] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. of Computer and System Sciences* 28, 270–299, April 1984.

[25] S. Goldwasser, S. Micali and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. of Comp.*, Vol. 18, No. 1, pp. 186–208, February 1989.

[26] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing,* 17(2):281–308, April 1988.

[27] R. Impagliazzo and S. Rudich, "Limits on the provable consequences of one-way permutations," Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[28] T. Leighton and S. Micali, "Provably fast and secure digital signature algorithms based on secure hash functions," Manuscript, March 1993.

[29] T. Leighton and S. Micali, "New approaches to secret key exchange," *Advances in Cryptology – Crypto* 93 *Proceedings*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed., Springer-Verlag, 1993.

[30] M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Computation,* Vol. 17, No. 2, April 1988.

[31] M. Luby and C. Rackoff, "A study of password security," manuscript.

[32] S. Micali, "CS proofs," Manuscript.

[33] S. Micali, C. Rackoff and B. Sloan, "The notion of security for probabilistic cryptosystems," SIAM J. of Computing, April 1988.

[34] M. Naor and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[35] M. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," MIT Laboratory for Computer Science TR-212, January 1979.

[36] C. Rackoff and D. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[37] R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

[38] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM 21 (1978).

[39] RSA Data Security, Inc., "PKCS #1: RSA Encryption Standard," June 1991.

[40] P. Rogaway and B. Blakley, "An asymmetric authentication protocol," IBM Technical Disclosure Bulletin (1993).

[41] G. Tsudik, "Message authentication with one-way hash functions," IEEE INFOCOM '92.

[42] H. Williams, "A modification of the RSA public key encryption procedure," *IEEE Transactions on Information Theory*, Vol. IT-26, No. 6, November 1980.

[43] A. Yao , "Theory and applications of trapdoor functions," Proceedings *of the* 23rd *Symposium on Foundations of Computer Science*, IEEE, 1982.

[44] Y. Zheng and J. Seberry, "Practical approaches to attaining security against adaptively chosen ciphertext attacks," *Advances in Cryptology – Crypto* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

A Proofs for Encryption

We present proofs of security for some of the encryption schemes. We will assume (wlog) that for any algorithm and any oracle for that algorithm, all queries made of the oracle are distinct.

THE $E(x) = f(r) \parallel G(r) \oplus x$ SCHEME IS POLYNOMIALLY SECURE. The proof is by contradiction. Let $A = (F, A_1)$ be an adversary that defeats the protocol; infinitely often, it gains advantage $\lambda(k)$ for some inverse polynomial λ. We construct an algorithm $M(f, d, y)$ that, when $(f, f^{-1}, d) \leftarrow \mathcal{G}(1^k)$; $r \leftarrow d(1^k)$; $y \leftarrow f(r)$, manages significantly often to compute $f^{-1}(y)$. Algorithm M defines E based on f as specified by our scheme. It simulates the oracle G in the natural way (by itself flipping coins to answer queries) and samples $(m_0, m_1) \leftarrow F^G(E)$. If ever G is asked an r such that $f(r) = y$, then M outputs r and halts. Otherwise, the $F(E)$ terminates and M chooses $\alpha \leftarrow y \parallel s$ for $s \leftarrow \{0,1\}^{|m_0|}$. Then M simulates $A_1^G(E, m_0, m_1, \alpha)$, watching the oracle queries that A_1 makes to see if there is any oracle query r for which $f(r) = y$. If there is, M outputs r. Let A_k be the event that A_1 asks the query $r = f^{-1}(y)$. A_1 has no advantage in distinguishing m_0 and m_1 in the case that A_1 does not ask for the image of G at r. So

$$1/2 + \lambda(k) \quad = \quad \Pr[A \text{ succeeds} \mid \mathsf{A}_k] \cdot \Pr[\mathsf{A}_k] + \Pr\left[A \text{ succeeds} \mid \overline{\mathsf{A}_k}\right] \cdot \Pr\left[\overline{\mathsf{A}_k}\right]$$

is at most $\Pr[\mathsf{A}_k] + 1/2$. Thus $\Pr[\mathsf{A}_k] \geq \lambda(k)$ must be nonnegligible, and M succeeds nonnegligibly often in inverting f.

THE $E(x) = f(r) \parallel G(r) \oplus x \parallel H(rx)$ SCHEME IS SECURE AGAINST CHOSEN CIPHERTEXT ATTACK. Let $A = (F, A_1)$ be an RS-adversary that succeeds with probability $1/2 + \lambda(k)$ for some nonnegligible function $\lambda(k)$. We construct an algorithm $M(f, d, y)$ that computes $f^{-1}(y)$ non-negligibly often, where $(f, f^{-1}, d) \leftarrow \mathcal{G}_*(1^k)$; $r \leftarrow d(1^k)$; $y \leftarrow d(r)$. Algorithm M begins by running $F(E)$ where E is defined from f as specified by our scheme. F takes three oracles, namely, G, H and $D^{G,H}$, whose queries are answered by F as follows. If a query r to G satisfies $f(r) = y$ then M outputs r and halts; else it returns a random string of the appropriate length. If a query rx to H satisfies $f(r) = y$ then M outputs r and halts; else it returns a random string of the appropriate length. To answer query $a \parallel w \parallel b$ to $D^{G,H}$ algorithm M sees if it has already asked some query r of G and ru of H, where $a = f(r)$ and $w = G(r) \oplus u$, and if so returns u; else it returns $\mathsf{invalid}$. If M completes the running of $F(E)$ then it obtains an output (m_0, m_1). Now M runs $A_1(E, m_0, m_1, \alpha)$ where $\alpha = y \parallel w \parallel b$ for $w \leftarrow \{0,1\}^{|m_0|}$ and $b \leftarrow \{0,1\}^k$. Once again, M must simulate the behavior of queries to G, H, and $D^{G,H}$. This is done exactly as before, when F was being run by M.

To see that this construction works, first consider the "real" environment of A running with its oracles. Let A_k denote the event that $a \parallel w \parallel b \leftarrow F(E)$, for some a, w, and b, and A made some oracle call of $G(r)$ or $H(ru)$, where $f(r) = a$. Let L_k denote the event that A_1 asks $D^{G,H}$ some query $a \parallel w \parallel b$ where $b = H(f^{-1}(a) \parallel w \oplus G(f^{-1}(a)))$, but A_1 never asked its H-oracle for the image of $f^{-1}(a) \parallel w \oplus G(f^{-1}(a))$. Let $n(k)$ denote the total number of oracle queries made. It is easy to verify that $\Pr[\mathsf{L}_k] \leq n(k)2^{-k}$. It is also easy to see that

$$\Pr\left[A \text{ succeeds} \mid \overline{\mathsf{L}_k} \wedge \overline{\mathsf{A}_k}\right] = 1/2.$$

Thus $1/2 + \lambda(k) = \Pr[A \text{ succeeds }]$ is bounded above by

$$\Pr[A \text{ succeeds} \mid \mathsf{L}_k]\Pr[\mathsf{L}_k] + \Pr\left[A \text{ succeeds} \mid \overline{\mathsf{L}_k} \wedge \mathsf{A}_k\right]\Pr\left[\overline{\mathsf{L}_k} \wedge \mathsf{A}_k\right] + \Pr\left[A \text{ succeeds} \mid \overline{\mathsf{L}_k} \wedge \overline{\mathsf{A}_k}\right]\Pr\left[\overline{\mathsf{L}_k} \wedge \overline{\mathsf{A}_k}\right]$$

which is at most $n(k)2^{-k} + \Pr[\mathsf{A}_k] + 1/2$. And so

$$\Pr[\mathsf{A}_k] \quad \geq \quad \lambda(k) - n(k)2^{-k}.$$

Now, returning to the simulation of A by M, note that M fails to behave like A with probability bounded by $\Pr[\mathsf{L}_k]$, and so

$$\Pr[M \text{ inverts } f \text{ at } y] \geq \lambda(k) - n(k)2^{-k+1}$$

which is still nonnegligible. This completes the proof.

THE $E(x) = f(r) \parallel G(r) \oplus x \parallel H(rx)$ SCHEME IS NON-MALLEABLE. Intuitively, the presence of a valid tag $H(r'x')$ in an encrypted string α' which is not a copy of an encryption provided to the adversary A acts as a "proof of knowledge" that A "knows" (can recover) x'. Now suppose A, seeing $\alpha = a \parallel w \parallel b$ encrypting $x = G(f^{-1}(a)) \oplus w$, manages to come up with the encryption of a string x' correlated to x. When r is not asked of G, adversary A cannot so correlate x to the (known value) x' because of her having no idea of the value of $G(r)$. Thus A must ask G the image of r reasonably often. Whenever she does this, she has effectively inverted the trapdoor permutation. The argument above can be formalized; we now sketch how to do so.

Given an M-adversary $\mathcal{A} = (F, A)$ and an interesting relation ρ, computed by polynomial time machine M, define the polynomial time algorithm $A_*(E, \pi)$ as follows:

> $A_*(E, \pi)$ computes $x_* \leftarrow \pi(1^k)$; $r_* \leftarrow d(1^k)$; $\alpha_* \leftarrow f(r_*) \parallel G(r_*) \oplus x_* \parallel H(r_*x_*)$; $\alpha'_* \leftarrow A(f, \pi, \alpha_*)$. If $\alpha'_* = \alpha_*$, then A_* outputs the encryption of 0. Otherwise, A_* outputs α'_*.

We will prove that $|\varepsilon(k) - \varepsilon_*(k)|$ is negligible, where these quantities are as in the definition of non-malleability. Some case analysis is required to show this claim. It is based on considering two related experiments, the first to define $\varepsilon(k)$ and the second to define $\varepsilon_*(k)$. We begin by describing Experiment 1. Here $G \leftarrow 2^\infty$; $H \leftarrow 2^\infty$; $(f, f^{-1}, d) \leftarrow \mathcal{G}_*(1^k)$; $E \leftarrow \langle x : r \leftarrow d(1^k) : f(r) \parallel G(r) \oplus x : H(rx) \rangle$; $\pi \leftarrow F^{G,H}(E)$; $x \leftarrow \pi^{G,H}(1^k)$; $r \leftarrow d(1^k)$; $a \leftarrow f(r)$; $w \leftarrow G(r) \oplus x$; $b \leftarrow H(rx)$; $\alpha \leftarrow a \parallel w \parallel b$; and $\alpha' \leftarrow A(E, \pi, \alpha)$. Write $\alpha' = a'w'b'$, $r' = f^{-1}(a')$, and $x' = w' \oplus G(r')$, We are interested in the value of $M^{G,H}(x, x', E, \pi)$ whose expectation, which we denote $\mathbf{E}_1[\rho(x, x')]$, is precisely $\varepsilon(k)$. In performing Experiment 1 we distinguish the following cases:

Case 1: $\alpha' = \alpha$. In this case, $\rho(x, x') = 0$ by our definition of an interesting relation.

Case 2: Assume Case 1 does not hold and $\mathcal{A}$ made no H-oracle query of $r'x'$:

`Case 2a.` $b' = H(r'x')$. This event happens with probability 2^{-k}.

`Case 2b.` $b' \neq H(r'x')$. In this case the encryption is garbled, the decryption is 0, and $\rho(x, x') = 0$ by our definition of an interesting relation.

Case 3: Suppose neither Case 1 nor Case 2 holds.

`Case 3a.` For any string $r'x'$ queried of H with $H(r'x') = b'$, either $f(r') \neq a$, or else $G(r') \oplus x' \neq w'$. Then $\rho(x, x') = 0$.

`Case 3b.` Here α' is a valid encryption and A can extract r' and x'. distinguish. Let λ_1 denote the probability of this case. We distinguish:

Case 3b(i). When A has not made a G-oracle call of r and

Case 3b(i)′ $M^{G,H}$ asks a query r. Let $\epsilon(k)$ be a negligible function bounding the probability of this case. Let λ_2 be the probability of this case.

Case 3b(i)″ $M^{G,H}$ asks no query r.

Case 3b(ii). When A makes a G-oracle call of r. Let $\epsilon(k)$ be a negligible function bounding the probability of this case.

We can upper-bound $\mathbf{E}_1[\rho(x,x')]$ by

$$\mathbf{E}_1[\rho(x,x')] \;\le\; \Pr[\text{Case 2a}]\cdot 2^{-k} + \Pr[\text{Case 3b}]\cdot \mathbf{E}[\rho(x,x')|\text{Case 3b(i)}]+$$
$$\Pr[\text{Case 3b(ii)}] \;\le\; 2^{-k} + \lambda_1(\epsilon(k)+\lambda_2) + \epsilon(k).$$

We now describe Experiment 2. This is defined by $G \leftarrow 2^\infty$; $H \leftarrow 2^\infty$; $(f, f^{-1}, d) \leftarrow \mathcal{G}_*(1^k)$; $E \leftarrow \langle\, x :\; r \leftarrow d(1^k) :\; f(r) \;\|\; G(r)\oplus x \;\|\; H(rx)\,\rangle$; $\pi \leftarrow F^{G,H}(E)$; $x \leftarrow \pi^{G,H}(1^k)$; $x_* \leftarrow \pi^{G,H}(1^k)$; $r_* \leftarrow d(1^k)$; $a_* \leftarrow f(r_*)$; $w \leftarrow G(r_*)\oplus x_*$; $b_* \leftarrow H(r_* x_*)$; $\alpha_* \leftarrow a_* \;\|\; w_* \;\|\; b_*$; $\alpha'_* \leftarrow A(E,\pi,\alpha_*)$. Write $\alpha'_* = a'_* w'_* b'_*$, $r'_* = f^{-1}(a')$, and $x'_* = w'_* \oplus G(r'_*)$ if $\alpha_* = \alpha'_*$, 0 otherwise. We are interested in the value of $M^{G,H}(x, x'_*, E, \pi)$ whose expectation, which we denote $\mathbf{E}_2[\rho(x,x'_*)]$, is precisely $\varepsilon_*(k)$.

In analyzing Experiment 2 we perform the same case analysis as above. An important observation is that in the Experiments 1 and 2, the distribution on third arguments to A is identical. Because of this, $\Pr_1[\text{Case 3b}] = \Pr_2[\text{Case 3b}]$. Also, it is easy to see that $\mathbf{E}_1[\rho(x,x')|\text{Case 3b(i)}''] = \mathbf{E}_2[\rho(x,x')|\text{Case 3b(i)}'']$ One can then lower-bound $\mathbf{E}_2[\rho(x,x'_*)|\text{Case 3b(i)}'']$ by

$$\mathbf{E}_2[\rho(x,x'_*)] \;\ge\; \Pr[\text{Case 3b(i)}'']\cdot \mathbf{E}_2[\rho(x,x'_*)|\text{Case 3b(i)'}] \;\ge\; (\lambda_1 - 2\epsilon(k))\lambda_2.$$

Therefore $|\mathbf{E}[\rho(x,x')] - \mathbf{E}[\rho(x,x'_*)]| \le 4\epsilon(k) + 2^{-k}$, and we are done.

B Proof of Security for the Signature Scheme

Suppose F is an S-adversary successful with a probability $\lambda(k)$ that is not negligible. We construct algorithm $M(f,d,y)$ so that

$$\varepsilon(k) \stackrel{\text{def}}{=} \Pr[\,(f,f^{-1},d)\leftarrow\mathcal{G}_*(1^k);\; x\leftarrow d(1^k);\; y\leftarrow f(x):\; M(f,d,y)=x\,]$$

is not negligible, contradicting the fact that $\mathcal{G}_*$ is a trapdoor permutation generator. $M(f,d,y)$ works as follows. It lets $\text{PK} = f$. It flips coins for F and starts running $F(\text{PK})$. We assume that F makes exactly $n(k)$ queries to H, all distinct, and that if F makes a signing query m then it has already queried $H(m)$; this is easily seen to be wlog. M chooses $t \in \{1,\ldots,n(k)\}$ at random. It then replies to queries as follows:

(1) Let m_i denote the i-th H query that F makes. If $i = t$ then M answers by returning y. Else it chooses $r_i \leftarrow \{0,1\}^k$ and returns $y_i = f(r_i)$.

(2) Suppose F makes signing query m. If $m = m_t$ then M halts, admitting failure. Otherwise M answers with r_i where $i \ne t$ satisfies $m = m_i$.

Let (m,σ) be F's output. If $m \ne m_t$ then M halts admitting failure. Else if $f(\sigma) = m$ then M outputs σ and halts; else again it admits failure. For the analysis, consider the experiment in which t is chosen at random and then F is run with its oracles in the usual manner; that is:

$$R\leftarrow 2^\infty;\; t\leftarrow\{1,\ldots,n(k)\};\; (\text{PK},\text{SK})\leftarrow\mathcal{G}(1^k);\; (m,\sigma)\leftarrow F^{R,\mathsf{Sign}^R(\text{SK},\cdot)}(\text{PK})\;.$$

Let S_k denote the event that F is succesful in this experiment. Note that if F is succesful and its output (m,σ) satisfies $m = m_t$ then m_t was by definition not queried of the signing oracle. It follows that $\varepsilon(k) = \Pr[\,\mathsf{S}_k \wedge (m = m_t)\,]$ where the probability here is over the experiment just defined. The latter probability is estimated as follows. If $m \notin \{m_1,\ldots,m_{n(k)}\}$ then F succeeds

with probability at most 2^{-k}. So

$$\lambda(k) - 2^{-k} \;=\; \sum_{i=1}^{n(k)} \Pr[\, \mathsf{S}_k \;\wedge\; (m = m_i)\,] \,.$$

Since t is chosen at random we have $\Pr[\, \mathsf{S}_k \;\wedge\; (m = m_t)\,] \geq (\lambda(k) - 2^{-k})/n(k)$. Thus $\varepsilon(k) \geq (\lambda(k) - 2^{-k})/n(k)$ which is still not negligible, as desired.