Hashing Solutions Instead of Generating Problems: On the Interactive Certification of RSA Moduli

Benedikt Auerbach[1] and Bertram Poettering[2(✉)]

[1] Ruhr University Bochum, Bochum, Germany
benedikt.auerbach@rub.de
[2] Royal Holloway, University of London, London, UK
bertram.poettering@rhul.ac.uk

Abstract. Certain RSA-based protocols, for instance in the domain of group signatures, require a prover to convince a verifier that a set of RSA parameters is well-structured (e.g., that the modulus is the product of two distinct primes and that the exponent is co-prime to the group order). Various corresponding proof systems have been proposed in the past, with different levels of generality, efficiency, and interactivity.

This paper proposes two new proof systems for a wide set of properties that RSA and related moduli might have. The protocols are particularly efficient: The necessary computations are simple, the communication is restricted to only one round, and the exchanged messages are short. While the first protocol is based on prior work (improving on it by reducing the number of message passes from four to two), the second protocol is novel. Both protocols require a random oracle.

1 Introduction

A common property of cryptographic primitives in the domain of public-key cryptography (PKC) is that there is, in most cases, a natural distinction between a secret-key holder (SKH) and a public-key holder (PKH). For instance, in the digital signature (DS) context the SKH is the signer, and in public-key encryption (PKE) the SKH is the receiver; the verifier and the sender, respectively, are PKHs. The security properties of such schemes are typically focused on protecting primarily the SKH: In the signature context, unforgeability means that the signer cannot be impersonated by an adversary, and security notions for PKE require that messages encrypted to the receiver remain confidential. Thus, naturally, the SKH has a vital interest in its keys being properly generated, i.e., in a way covered by the security model, while this is only of secondary importance to the PKH.

In some PKC applications, however, also parties not holding the secret key might require assurance about that the key material has been generated in a

The full version of this article can be found in the IACR eprint archive as article 2018/013.

M. Abdalla and R. Dahab (Eds.): PKC 2018, LNCS 10770, pp. 403–430, 2018.
https://doi.org/10.1007/978-3-319-76581-5_14

proper way. Typical examples arise in multi-party settings where the SKH manages a set of mutually distrusting parties who require protection from each other. For instance, in group signature schemes there is a group manager that issues certificates to registered parties, allowing them to sign messages on behalf of the whole group. While the resulting signatures should in principle be anonymous (cannot be linked to the particular signer), to prevent misuse there is often a traceability feature that allows the group manager to revoke the anonymity of a signer by creating a publicly-verifiable non-interactive proof that testifies that an indicated signer created a particular signature. If such a tracing option exists, the group manager should however not be able to falsely accuse a member of having signed some document. Many group signature schemes have been proposed in the past, but some of them (e.g., [1]) provably provide the latter property only if the group manager's keys are properly formed.[1] Other settings where trust in the secret keys generated by other parties is required include e-cash [13], cryptographic accumulators [9], undeniable signatures [18], double-authentication preventing signatures [2,27].

If a cryptographic scheme is solely based on the discrete logarithm problem (DLP) in a prime-order group, checking that keys of the type $X = g^x$ are well-formed is a trivial job (because *all* keys are well-formed). In the RSA setting the situation is more subtle: Given parameters (N, e), before assuming the security of the system the PKH might want to be convinced that the following questions can be answered affirmatively: (1) does N have precisely two prime divisors, (2) is N square-free, (3) is e coprime to $\varphi(N)$, i.e., is the mapping $m \mapsto m^e \bmod N$ a bijection (rather than lossy). Further, in some settings it might be necessary to know (4) whether $N = pq$ is a safe-prime modulus, i.e., whether $(p-1)/2$ and $(q-1)/2$ are primes by themselves. In settings specifically based on the hardness of factoring an additional question might be (5) whether squaring is a bijection on $\mathbf{QR}(N)$, more specifically (6) whether N is a Blum integer, and even more specifically (7) whether N is a Rabin–Williams integer.[2]

What are known approaches for convincing participants of the validity of predicates like the ones listed above? In some research papers corresponding arguments are just missing [1], or they are side-stepped by explicitly assuming honesty of key generation in the model [2]. Other papers refer to works like [10] that propose non-interactive proof systems for convincing verifiers of the validity of such relations. Concretely, [10] provides a NIZK framework for showing that an RSA number is the product of two safe primes. While powerful, the NIZK technique turns out to be practically not usable: The argument is over the intermediate results of four Miller–Rabin tests, a large number of range tests,

[1] Concretely, the protocol from [1] is presented in the safe-prime-RSA setting where $N = pq$ with $p = 2p' + 1, q = 2q' + 1$ such that p, q, p', q' are all primes. Some of the security properties of [1] hold in respect to the CDH problem in $\mathbb{Z}_N^*$. If $N = pq$ and thus $\mathbb{Z}_N^* = \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ as it should, CDH is arguably hard. However, if the group manager announces a malformed N that is made up of a large number of (small) prime factors, solving CDH becomes easy.

[2] An RSA modulus $N = pq$ is a Blum integer if $p \equiv q \equiv 3 \pmod 4$, and it is a Rabin–Williams integer if $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$.

etc., making the resulting proof string prohibitively long. Another approach is to pick prime numbers, moduli, and exponents in a certain way such that showing specific properties becomes feasible with number-theoretic techniques. Working with restricted parameter classes might however remove standard conformance and render implementations less efficient; for instance, the authors of [23] develop tools for showing that the mapping $m \mapsto m^e$ is a permutation, but these tools work only for fairly large values of e.

A third approach is tightly connected with the number-theoretic structures that motivate the requirements for the conditions listed above. (It is less general than the NIZK approach of [10] but usually does not require picking parameters in a specific way.) For instance, if an application of RSA requires that e be coprime to $\varphi(N)$ then this is for a specific reason, namely that information shall not be lost (but remain recoverable) when raising it to the power of e. Thus, instead of abstractly checking the $e \mid \varphi(N)$ relation, a corresponding check could be centered precisely around the information-loss property of the exponentiation operation. Our results are based on this strategy. Our techniques are inspired by, and improving on, prior work that we describe in detail in the following.

1.1 Interactive Zero-Knowledge Testing of Certain Relations

We reproduce results of Gennaro et al. [19]. As a running example, consider the question of whether $e \mid \varphi(N)$ holds, where N is an RSA modulus and e a small prime exponent. The relation holds if and only if the mapping $x \mapsto x^e \bmod N$ is bijective, characterized by all $y \in \mathbb{Z}_N^*$ having an eth root. This motivates an (interactive) protocol in which a prover convinces a verifier of relation $e \mid \varphi(N)$ by first letting the verifier pick a random value $y \in \mathbb{Z}_N^*$ and send it to the prover, then letting the prover (who knows the factorization of N) compute the eth root $x \in \mathbb{Z}_N^*$ of y and return it to the verifier, and finally letting the verifier accept if and only if $x^e = y \bmod N$. Prover and verifier may run multiple repetitions of this protocol, each time with a fresh challenge y. If the prover is able to return a valid response for each challenge, then the verifier is eventually convinced of the $e \mid \varphi(N)$ claim. Indeed, if $e \nmid \varphi(N)$, then only about one of e elements of $\mathbb{Z}_N^*$ have an eth root, so the protocol would detect this with high probability and a cheating prover would be caught.

Note that if the protocol would be deployed in precisely the way we described it, it would be of limited use. The reason is that it is not zero-knowledge; in particular, the prover would effectively implement an 'eth root oracle' for values y arbitrarily picked by the verifier, and this would likely harm the security of most applications. The proposal of [19] considers fixing this by making sure that challenges y are picked in a sufficiently random way. Concretely, the full protocol [19, Sect. 4.1] involves four message passes as follows: (1) the verifier picks $y_1 \in \mathbb{Z}_N^*$ and sends a commitment to this value to the prover, (2) the prover picks $y_2 \in \mathbb{Z}_N^*$ and sends this value to the verifier, (3) the verifier opens the commitment; both parties now compute $y \leftarrow y_1 y_2$, (4) the prover computes the eth root of y and sends it to the verifier. Unfortunately, the security analysis of [19] does not

cover the full protocol; rather it restricts attention to only the last prover-to-verifier message and shows that it is zero-knowledge under the assumption that value y "can be thought as provided by a trusted third party" [19, Sect. 2.3]. We stress that a proof for the full four-message protocol is not immediate: Proving it zero-knowledge seems to require assuming an extractability property of the commitment scheme (so that the simulator can find 'the right' y_2 value), and the increased interactiveness calls for a fresh analysis in a concurrent communication setting anyway (if the protocol shall be of practical relevance). Neither of these issues is mentioned, let alone resolved, in [19].

1.2 Our Results

We construct practical protocols for convincing a verifier that certain relevant number-theoretic properties hold for RSA parameters. This includes statements on the number of prime factors of the modulus, its square-freeness, etc. Concretely, we propose two generic protocol frameworks that can be instantiated to become proof systems for many different relations: The first framework is based on [19] and effectively compresses the first three messages of the full protocols into a single one by, intuitively speaking, using a random oracle to implement the mentioned trusted third party. Precisely, continuing our running example, we let the verifier only specify a random seed r and let both parties derive value y as per $y \leftarrow H(r)$ via a random oracle. The random oracle model turns out to be strong enough to make the full protocol sound and zero-knowledge. Because of the reduced number of message passes, concurrency is not an issue.

The second framework is similar in spirit but uses the random oracle in a different and novel way. Here, the challenge y can be freely picked by the verifier (no specific distribution is required), the prover again computes the eth root x of it, but instead of sharing x with the verifier it only discloses the hash $H(x)$ of it. Note that, unless the verifier knows value x anyway, if H behaves like a random oracle then the hash value does not leak anything.

We highlight that the second protocol has two important advantages over the first: (1) The first protocol requires a random oracle that maps into the 'problem space' (here: challenge space $\mathbb{Z}_N^*$). However, for some number-theoretic tests, e.g., whether N is a Blum integer, the problem space we (and [19]) work with is $\mathbf{QR}(N)$, i.e., the set of quadratic residues modulo N, and for such spaces it is unclear how to construct a random oracle mapping into them. Note that, in contrast, the second protocol does not require hashing into any particular set. (2) Some number-theoretic relations allow for an easier check when the second framework is used. For instance, identifying Blum integers involves the prover computing the four square roots that quadratic residues always have. In the first protocol framework, returning all four square roots is prohibitive as this would immediately allow for factorizing N. In the second framework, however, hash values of all square roots can be returned without doing harm to security.

Please consider Sect. 5 for the full list of number-theoretic properties for which we provide a proof system.

1.3 Related Work

We note that techniques similar to ours appear implicitly or explicitly in a couple of prior works. For instance, the validation of RSA parameters is a common challenge in password-based key agreement; in particular, adversaries might announce specially crafted parameters (N, e) that would lead to partial password exposure. The work of Zhu et al. [34] addresses this, but without a formal analysis, by pursuing approaches that are similar to our second protocol instantiated with one particular number-theoretic relation. The work of [28] provides a security analysis of [34]. (It seems, however, that the analysis is incomplete: The output length of the hash function does not appear in the theorem statement, but for short output lengths the statement is obviously wrong.) We conclude by noting that both [28,34], and also a sequence of follow-up works in the domain of password-based key agreement, employ variants of our two protocols in an ad-hoc fashion, and not at the generic level and for the large number of number-theoretic problems as we do.

A higher level of abstraction, also in the domain of password-based key agreement, can be found in the work of Catalano et al. [11]. Their work considers exclusively our first approach. Further, while considering soundness and zero-knowledge definitions for language problems, their constructions are not on that level but directly targeting specific number-theoretic problems.

Considering proof systems not relying on random oracles, basically any desired property of an RSA modulus can be proven by employing general zero-knowledge proof systems for NP languages [8,20,21]. However, these protocols are usually less efficient than proof systems designed to establish a particular property. Thus a vast amount of papers provides systems of the latter type. Targeted properties include that an RSA modulus N has factors of approximately equal size [6,12,16,17,24] or is the product of two safe primes [10]. The approach of having the prover provide solutions to number-theoretic problems is taken in several proof systems. Concretely, there are protocols of this type proving that N is square-free [7,19], has at most two prime factors [5,19,25,29], satisfies a weakened definition of Blum integer [5,29], is the product of two almost strong primes [19]. A shortcoming common to the protocols deciding whether N has at most two prime factors is that they either have a two-sided error or have to impose additional restrictions on N, the first leading to an increased number of repetitions of the protocol in order to achieve security, the latter to artificially restricted choices of N.

Bellare and Yung [3] show that any trapdoor permutation can be certified, i.e., they provide a protocol to prove that a function is invertible on an overwhelming fraction of its range. Kakvi et al. [23] show that given an RSA modulus N and an exponent e such that $e \geq N^{1/4}$ Coppersmith's method can be used to efficiently determine whether the RSA function $x \mapsto x^e$ defines a permutation on $\mathbb{Z}_N^*$. However, their result does not apply to exponents of size smaller than $N^{1/4}$. A proof for RSA key generation with verifiable randomness is given in [22].

The protocol makes use of the protocols of [7,29] as subroutines and relies on a trusted third party. Benhamouda et al. [4] provide a protocol proving in the random oracle model that at least two of the factors of a number N were generated using a particular prime number generator. However, in order to achieve security the construction requires N to be the product of many factors, which usually is prohibitive in the RSA setting.

We note that a topic in cryptography somewhat connected to our work is the fraudulent creation of parameters. More specifically, the works in [30–33] consider Kleptography, i.e., the creation of asymmetric key pairs by an adversary-modified generation algorithm such that, using a trapdoor, the adversary can recover the secret key from the public key. Preventing such attacks is not the goal of our work, and our protocols will indeed not succeed in catching properly performed Kleptography.

By nothing-up-my-sleeves (NUMS) parameter generation one subsumes techniques to propose parameters for cryptosystems in an explainable and publicly reproducible way. For instance, the internal constants of the hash functions of the SHA family are derived from the digits of the square and cube roots of small prime numbers, making the existence of trapdoors (e.g., for finding collisions) rather unlikely. While we do not advise against NUMS techniques, we note that using them restricts freedom in parameter generation and thus might break standard conformance and lead to less efficient systems. Moreover, and more relevantly in the context of our work, NUMS techniques typically apply to DL-based cryptosystems and not to RSA-based ones.

1.4 Organization

The overall focus of this work is on providing practical methods for proving certain properties of RSA-like parameter sets. Our interactive proof systems, however, follow novel design principles that promise finding application also outside of the number-theoretic domain. We thus approach our goal in a layered fashion, by first exposing our proof protocols such that they work for abstract formulations of problems and corresponding solutions, and then showing how these formalizations can be instantiated with the number-theoretic relations we are interested in.

More concretely, the structure of this article is as follows: In Sect. 2 we fix notation and recall some general results from number theory. In Sect. 3 we formulate a variant of the is-word-in-language problem and connect it to problems and solutions in some domain; we further introduce the concept of a challenge-response protocol for proving solutions of the word problem. In Sect. 4 we study two such protocols: Hash-then-Solve, which is inspired by the work of [19], and Solve-then-Hash, which is novel. Finally, in Sect. 5 we show how RSA-related properties can be expressed as instances of our general framework so that they become accessible by our proof systems.

2 Preliminaries

We fix notation and recall basic facts from number theory.

2.1 Notation

Parts of this article involve the specification of program code. In such code we use assignment operator '$\leftarrow$' when the assigned value results from a constant expression (including from the output of a deterministic algorithm), and we write '$\leftarrow_{\$}$' when the value is either sampled uniformly at random from a finite set or is the output of a randomized algorithm. In a security experiment, the event that some algorithm A outputs the value v is denoted with $A \Rightarrow v$. In particular, $\Pr[A \Rightarrow 1]$ denotes the probability, taken over the coins of A, that A outputs value 1. We use bracket notation to denote associative arrays (a data structure that implements a 'dictionary'). For instance, for an associative array A the instruction $A[7] \leftarrow 3$ assigns value 3 to memory position 7, and the expression $A[2] = 5$ tests whether the value at position 2 is equal to 5. Associative arrays can be indexed with elements from arbitrary sets. When assigning lists to each other, with '_' we mark "don't-care" positions. For instance, $(a, \_) \leftarrow (9, 4)$ is equivalent to $a \leftarrow 9$ (value 4 is discarded). We use the ternary operator known from the C programming language: If C is a Boolean condition and e_1, e_2 are arbitrary expressions, the expression "C `?` e_1 `:` e_2" evaluates to e_1 if C holds, and to e_2 if C does not hold. We further use Iverson brackets to convert Booleans to numerical values. That is, writing "$[C]$" is equivalent to writing "C `?` 1 `:` 0". If A is a randomized algorithm we write $[A(x)]$ for the set of outputs it produces with non-zero probability if invoked on input x. If u, v are (row) vectors of values, $u \,\|\, v$ denotes their concatenation, i.e., the vector whose first elements are those of u, followed by those of v. We use symbol $\uplus$ to indicate when the union of two sets is a disjoint union.

2.2 Number Theory

We write $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{P} \subseteq \mathbb{N}$ for the set of natural numbers and prime numbers, respectively. For every natural number $N \in \mathbb{N}$ we denote the set of prime divisors of N with $\mathbb{P}(N)$. Thus, for any $N \in \mathbb{N}$ there exists a unique family $(\nu_p)_{p \in \mathbb{P}(N)}$ of multiplicities $\nu_p \in \mathbb{N}$ such that

$$N = \prod_{p \in \mathbb{P}(N)} p^{\nu_p}.$$

We denote with $\mathrm{odd}(N)$ the 'odd part' of N, i.e., what remains of N after all factors 2 are removed; formally, $\mathrm{odd}(N) = \prod_{p \in \mathbb{P}(N), p \neq 2} p^{\nu_p}$.

Consider $N \in \mathbb{N}$ and the ring $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. The multiplicative group $\mathbb{Z}_N^*$ of $\mathbb{Z}_N$ has order $\varphi(N) = \prod_{p \in \mathbb{P}(N)} (p-1) p^{\nu_p - 1}$, where φ is Euler's totient function. By the Chinese Remainder Theorem (CRT) there exists a ring isomorphism

$$\psi \colon \mathbb{Z}_N \xrightarrow{\sim} \bigtimes_{p \in \mathbb{P}(N)} \mathbb{Z}_{p^{\nu_p}}.$$

For $N, e \in \mathbb{N}$ consider the exponentiation mapping $x \mapsto x^e \bmod N$. This mapping is 1-to-1 on $\mathbb{Z}_N^*$ iff $\gcd(e, \varphi(N)) = 1$. The general statement, that holds for all N, e, is that the exponentiation mapping is L-to-1 for

$$L = \prod_{p \in \mathbb{P}(N)} \gcd(e, \varphi(p^{\nu_p})). \tag{1}$$

We write $\mathbf{QR}(N)$ for the (group of) quadratic residues (i.e., squares) modulo N.

3 Challenge-Response Protocols for Word Problems

We define notions of languages, statements, witnesses, and a couple of algorithms that operate on such objects. We then introduce the notion of a challenge-response protocol for the word problem in such a setting.

3.1 Associating Problems with the Words of a Language

Statements, candidates, witnesses. Let Σ be an alphabet and let $\mathcal{L} \subseteq \mathcal{U} \subseteq \Sigma^*$ be languages. We assume that deciding membership in $\mathcal{U}$ is efficient, while for $\mathcal{L}$ this might not be the case. Each element $x \in \Sigma^*$ is referred to as a *statement*. A statement x is a *candidate* if $x \in \mathcal{U}$. A statement x is *valid* if $x \in \mathcal{L}$; otherwise, it is *invalid*. (Thus, in general there coexist valid and invalid candidates.) For all candidates x we assume a (possibly empty) set of witnesses $\mathcal{W}_x$ such that valid candidates are characterized by having a witness: $\forall x \in \mathcal{U} \colon |\mathcal{W}_x| \geq 1 \iff x \in \mathcal{L}$.

Relating problems with candidates. For all candidates $x \in \mathcal{U}$ let $\mathcal{P}_x$ be a problem space and $\mathcal{S}_x$ a solution space, where we require that deciding membership in $\mathcal{P}_x$ is efficient. Let $\mathcal{R}el_x \subseteq \mathcal{P}_x \times \mathcal{S}_x$ be a relation that (abstractly) matches problems with solutions. For any problem $P \in \mathcal{P}_x$ we write $\mathcal{S}ol_x(P) := \{S \mid (P, S) \in \mathcal{R}el_x\} \subseteq \mathcal{S}_x$ for the set of its solutions. Not necessarily all problems are solvable, so we partition the problem space as $\mathcal{P}_x = \mathcal{P}_x^+ \cup \mathcal{P}_x^-$ such that precisely the elements of $\mathcal{P}_x^+$ have solutions: $P \in \mathcal{P}_x^+ \iff |\mathcal{S}ol_x(P)| \geq 1$ and, equivalently, $P \in \mathcal{P}_x^- \iff \mathcal{S}ol_x(P) = \emptyset$. We extend relation $\mathcal{R}el_x$ to $\mathcal{R}el_x^* := \mathcal{R}el_x \cup (\mathcal{P}_x^- \times \{\bot\})$ by marking problems without solution with the special value $\bot$, and we extend notion $\mathcal{S}ol_x$ to $\mathcal{S}ol_x^*$ such that for all $P \in \mathcal{P}_x^-$ we have $\mathcal{S}ol_x^*(P) = \{\bot\}$. We require that every candidate has at least one problem-solution pair: $\forall x \in \mathcal{U} \colon |\mathcal{R}el_x| \geq 1$.

We assume four efficient algorithms, Verify, Sample, Sample*, and Solve, that operate on these sets. Deterministic algorithm Verify implements for all candidates the indicator function of $\mathcal{R}el$, i.e., decides whether a problem and a solution are matching. More precisely, Verify takes a candidate $x \in \mathcal{U}$, a problem $P \in \mathcal{P}_x$, and a potential solution $S \in \mathcal{S}_x$ for P, and outputs a bit that indicates whether (P, S) is contained in $\mathcal{R}el_x$ or not. Formally, $\forall x \in \mathcal{U}, (P, S) \in \mathcal{P}_x \times \mathcal{S}_x \colon \mathrm{Verify}(x, P, S) = 1 \iff (P, S) \in \mathcal{R}el_x$. Algorithm Sample is randomized, takes a candidate $x \in \mathcal{U}$, and outputs a (matching) problem-solution pair $(P, S) \in \mathcal{R}el_x$. Algorithm Sample* is randomized, takes a

candidate $x \in \mathcal{U}$, and outputs a pair $(P, S) \in \mathcal{R}el_x^*$ (note that $S = \perp$ if $P \in \mathcal{P}_x^-$). Finally, deterministic algorithm Solve takes a (valid) statement $x \in \mathcal{L}$, a witness $w \in \mathcal{W}_x$ for it, and a problem $P \in \mathcal{P}_x$, and outputs the subset of $\mathcal{S}_x$ that contains *all* solutions of P. (If no solution exists, Solve outputs the empty set.) Formally, $\forall x \in \mathcal{L}, w \in \mathcal{W}_x, P \in \mathcal{P}_x\colon \text{Solve}(x, w, P) = \mathcal{S}ol_x(P)$.

If we write $\mathcal{P} = \bigcup \mathcal{P}_x$, $\mathcal{S} = \bigcup \mathcal{S}_x$, $\mathcal{R}el = \bigcup \mathcal{R}el_x$, $\mathcal{R}el^* = \bigcup \mathcal{R}el_x^*$, $\mathcal{W} = \bigcup \mathcal{W}_x$, where the unions are over all $x \in \mathcal{U}$, a shortcut notation for the syntax of the four algorithms is

$$\begin{array}{rcccl} \mathcal{U} \times \mathcal{P} \times \mathcal{S} & \to & \text{Verify} & \to & \{0,1\} \\ \mathcal{U} & \to & \text{Sample} & \to_{\$} & \mathcal{R}el \\ \mathcal{U} & \to & \text{Sample}^* & \to_{\$} & \mathcal{R}el^* \\ \mathcal{L} \times \mathcal{W} \times \mathcal{P} & \to & \text{Solve} & \to & \text{Powerset}(\mathcal{S}) \end{array}$$

Number of solutions, spectrum, solvable-problem density. Note that different problems $P \in \mathcal{P}^+$ have, in general, different numbers of solutions. For any set $\mathcal{M} \subseteq \mathcal{U}$ of candidates, the *spectrum* $\#\mathcal{M}$ collects the cardinalities of the solution sets of all solvable problems associated with the candidates listed in $\mathcal{M}$. Formally,

$$\#\mathcal{M} := \{|\mathcal{S}ol_x(P)| : x \in \mathcal{M}, P \in \mathcal{P}_x^+\}.$$

Consequently, $\max \#\mathcal{L}$ is the largest number of solutions that solvable problems associated with valid candidates might have, and $\min \#(\mathcal{U} \setminus \mathcal{L})$ is the smallest number of solutions of solvable problems associated with invalid candidates. Further, for a set $\mathcal{M} \subseteq \mathcal{U}$ the *solvable-problem density distribution* $\Delta\mathcal{M}$, defined as

$$\Delta\mathcal{M} := \{|\mathcal{P}_x^+|/|\mathcal{P}_x| : x \in \mathcal{M}\},$$

indicates the fractions of problems that are solvable (among the set of all problems), for all candidates in $\mathcal{M}$. Most relevant in this article are the derived quantities $\min \Delta\mathcal{L}$ and $\max \Delta(\mathcal{U} \setminus \mathcal{L})$.

Uniformity notions for sampling algorithms. For the two sampling algorithms defined above we introduce individual measures of quality. For Sample we say it is *problem-uniform* (on invalid candidates) if for all $x \in \mathcal{U} \setminus \mathcal{L}$ the problem output by Sample(x) is uniformly distributed in $\mathcal{P}_x^+$. Formally, for all $x \in \mathcal{U} \setminus \mathcal{L}, P' \in \mathcal{P}_x^+$ we require that

$$\Pr[(P, \_) \leftarrow_{\$} \text{Sample}(x) : P = P'] = 1/|\mathcal{P}_x^+|.$$

Further we say that Sample is *solution-uniform* (on invalid candidates) if for all $x \in \mathcal{U} \setminus \mathcal{L}$ and each pair (P, S) output by Sample(x), solution S is uniformly distributed among all solutions for P. Formally, we require that for all $x \in \mathcal{U} \setminus \mathcal{L}, (P', S') \in [\text{Sample}(x)]$ we have

$$\Pr[(P, S) \leftarrow_{\$} \text{Sample}(x) : S = S' \mid P = P'] = 1/|\mathcal{S}ol_x(P')|.$$

For Sample^* we say it is *problem-uniform* (on valid candidates) if for all $x \in \mathcal{L}$ the problem output by $\text{Sample}^*(x)$ is uniformly distributed in $\mathcal{P}_x$. Formally, for all $x \in \mathcal{L}, P' \in \mathcal{P}_x$ we require that

$$\Pr[(P, \_) \leftarrow_\$ \text{Sample}^*(x) : P = P'] = 1/|\mathcal{P}_x|.$$

Further we say that Sample^* is *solution-uniform* (on valid candidates) if for all $x \in \mathcal{L}$ and each pair (P, S) output by $\text{Sample}^*(x)$, the solution S is uniformly distributed among all solutions of P (if a solution exists at all, i.e., if $S \neq \perp$). Formally, we require that for all $x \in \mathcal{L}, (P', S') \in [\text{Sample}^*(x)]$ we have

$$\Pr[(P, S) \leftarrow_\$ \text{Sample}^*(x) : S = S' \mid P = P'] = 1/|\mathcal{S}ol_x^*(P')|.$$

3.2 Challenge-Response Protocols

In the context of Sect. 3.1, a *challenge-response protocol* (CRP) for $(\mathcal{L}, \mathcal{U})$ specifies a (verifier) state space $\mathcal{S}t$, a challenge space $\mathcal{C}h$, a response space $\mathcal{R}sp$, and efficient algorithms $\text{V}_1, \text{P}, \text{V}_2$ such that $\text{V} = (\text{V}_1, \text{V}_2)$ implements a stateful verifier and P implements a (stateless) prover. In more detail, algorithm V_1 is randomized, takes a candidate $x \in \mathcal{U}$, and returns a pair (st, c), where $st \in \mathcal{S}t$ is a state and $c \in \mathcal{C}h$ a challenge. Prover P, on input of a valid statement $x \in \mathcal{L}$, a corresponding witness $w \in \mathcal{W}_x$, and a challenge $c \in \mathcal{C}h$, returns a response $r \in \mathcal{R}sp$. Finally, deterministic algorithm V_2, on input a state $st \in \mathcal{S}t$ and a response $r \in \mathcal{R}sp$, outputs a bit that indicates acceptance (1) or rejection (0). An overview of the algorithms' syntax is as follows.

$$\begin{array}{rcccl} \mathcal{U} & \to & \text{V}_1 & \to_\$ & \mathcal{S}t \times \mathcal{C}h \\ \mathcal{L} \times \mathcal{W} \times \mathcal{C}h & \to & \text{P} & \to_\$ & \mathcal{R}sp \\ \mathcal{S}t \times \mathcal{R}sp & \to & \text{V}_2 & \to & \{0,1\} \end{array}$$

We define the following correctness and security properties for CRPs.

Correctness. Intuitively, a challenge-response protocol is correct if honest provers convince honest verifiers of the validity of valid statements. Formally, we say a CRP is δ-*correct* if for all valid candidates $x \in \mathcal{L}$ and corresponding witnesses $w \in \mathcal{W}_x$ we have

$$\Pr\left[(st, c) \leftarrow_\$ \text{V}_1(x); r \leftarrow_\$ \text{P}(x, w, c) : \text{V}_2(st, r) \Rightarrow 1\right] \geq \delta.$$

If the CRP is 1-correct we also say it is perfectly correct.

Soundness. Intuitively, a challenge-response protocol is sound if (dishonest) provers cannot convince honest verifiers of the validity of invalid statements. Formally, a CRP is ε-*sound* if for all invalid candidates $x \in \mathcal{U} \setminus \mathcal{L}$ and all (potentially unbounded) algorithms P^* we have

$$\Pr\left[(st, c) \leftarrow_\$ \text{V}_1(x); r \leftarrow_\$ \text{P}^*(x, c) : \text{V}_2(st, r) \Rightarrow 0\right] \geq \varepsilon.$$

If the CRP is 1-sound we also say it is perfectly sound. To quantity $1 - \varepsilon$ we also refer to as the soundness error.

Zero-knowledge. Intuitively, a challenge-response protocol is (perfectly) zero-knowledge if (dishonest) verifiers do not learn anything from interacting with (honest) provers, beyond the fact that the statement is valid. Formally, a CRP is *(perfectly) zero-knowledge* if there exists a simulator S such that for all (potentially unbounded) distinguishers D, all valid candidates $x \in \mathcal{L}$, and all corresponding witnesses $w \in \mathcal{W}_x$, we have

$$|\Pr[\mathrm{D}^{\mathrm{P}(x,w,\cdot)} \Rightarrow 1] - \Pr[\mathrm{D}^{\mathrm{S}(x,\cdot)} \Rightarrow 1]| = 0.$$

Here, with $\mathrm{P}(x, w, \cdot)$ and $\mathrm{S}(x, \cdot)$ we denote oracles that invoke the prover algorithm P on input x, w, c and the simulator S on input x, c, respectively, where challenge c is in both cases provided by distinguisher D on a call-by-call basis.

In Sect. 4 we study two frameworks for constructing challenge-response protocols of the described type. The analyses of the corresponding protocols will be in the random oracle model, meaning that the algorithms $\mathrm{V}_1, \mathrm{P}, \mathrm{V}_2$ have access to an oracle H implementing a function drawn uniformly from the set of all functions between some fixed domain and range. Also the above correctness and security definitions need corresponding adaptation by (1) extending the probability spaces to also include the random choice of H, and (2) giving all involved algorithms, i.e., $\mathrm{V}_1, \mathrm{P}, \mathrm{V}_2, \mathrm{P}^*, \mathrm{D}$, oracle access to H. In the zero-knowledge definition, simulator S simulates both P and H.

4 Constructing Challenge-Response Protocols

In Sect. 3 we linked the word decision problem of a language to challenge-response protocols (CRP). Concretely, if $\mathcal{L} \subseteq \mathcal{U}$ are languages, a corresponding CRP would allow a prover to convince a verifier that a given candidate statement is in $\mathcal{L}$ rather than in $\mathcal{U} \setminus \mathcal{L}$. In the current section we study two such protocols, both requiring a random oracle. The first protocol, **Hash-then-Solve**, is inspired by prior work but significantly improves on it, while the second protocol, **Solve-then-Hash**, is novel. The bounds on correctness and security of the two protocols are, in general, incomparable. In the following paragraphs we give a high-level overview of their working principles.

Let $x \in \mathcal{U}$ be a (valid or invalid) candidate statement. In the protocol of Sect. 4.1 a random oracle H is used to generate problem instances for x as per $P \leftarrow \mathrm{H}(r)$, where r is a random seed picked by the verifier. If P has a solution S, the prover recovers it and shares it with the verifier who accepts iff the solution is valid. (If P has multiple solutions, the prover picks one of them at random.) Note that solving problems is in general possible also for invalid candidates, but the idea behind this protocol is that it allows for telling apart elements of $\mathcal{L}$ and $\mathcal{U} \setminus \mathcal{L}$ if the fraction of solvable problems among the set of all problems associated with valid candidates is strictly bigger than the fraction of solvable problems among all problems associated with invalid candidates, i.e., if $\min \Delta\mathcal{L} > \max \Delta(\mathcal{U} \setminus \mathcal{L})$. (As we show in Sect. 5, this is the case for some interesting number-theoretic decision problems.)

We now turn to the protocol of Sect. 4.2. Here, the random oracle is not used to generate problems as above. Rather, the random oracle is used to hash solutions into bit strings. Concretely, the verifier randomly samples a problem P with corresponding solution S. It then sends P to the prover who derives the set of all solutions for it; this set obviously includes S. The prover hashes all these solutions and sends the set of resulting hash values to the verifier. The latter accepts if the hash value of S is contained in this set. Note that finding the set of all solutions for problems is in general possible also for invalid candidates, but the protocol allows for telling apart valid from invalid candidates if (solvable) problems associated with valid candidates have strictly less solutions than problems associated with invalid candidates, i.e., if $\max \#\mathcal{L} < \min \#(\mathcal{U} \setminus \mathcal{L})$. Indeed, if the verifier does not accept more hash values than the maximum number of solutions for valid statements, a cheating prover will make the verifier accept only with a limited probability, while in the valid case the verifier will always accept. (We again refer to Sect. 5 for number-theoretic problems that have the required property.)

Let us quickly compare the two approaches. In principle, whether they are applicable crucially depends on languages $\mathcal{L},\mathcal{U}$ and the associated problem and solution spaces. Note that the random oracles are used in very different ways: in the first protocol to ensure a fair sampling of a problem such that no solution is known a priori (to neither party), and in the second protocol to hide those solutions from the verifier that the latter does not know anyway. That the random oracle in the first protocol has to map into the problem space might represent a severe technical challenge as for some relevant problem spaces it seems unfeasible to find a construction for such a random oracle.[3] In such cases the second protocol might be applicable.

4.1 A GMR-Inspired Protocol: Hash-then-Solve

A general protocol framework for showing that certain properties hold for a candidate RSA modulus (that it is square-free, Blum, etc.) was proposed by Gennaro, Micali, and Rabin in [19]. Recall from the discussion in the introduction that the full version of their protocol has a total of four message passes and involves both number-theoretic computations and the use of a commitment scheme. In this section we study a variant of this protocol where the commitment scheme is implemented via a random oracle. The benefit is that the protocol becomes more compact and less interactive. Concretely, the number of message passes decreases from four to two.

Let $\mathcal{L} \subseteq \mathcal{U} \subseteq \Sigma^*$ be as in Sect. 3.1, and let $l \in \mathbb{N}$ be a security parameter. Let $(\mathrm{H}_x)_{x\in\mathcal{U}}$ be a family of hash functions (in the security reduction: random oracles) such that for each $x \in \mathcal{U}$ we have a mapping $\mathrm{H}_x\colon \{0,1\}^l \to \mathcal{P}_x$. Consider the challenge-response protocol with algorithms $\mathrm{V}_1, \mathrm{P}, \mathrm{V}_2$ as specified in Fig. 1. The idea of the protocol is that the verifier picks a random seed r which it

[3] For instance if the problem space is the set of quadratic residues modulo some composite integer.

communicates to the prover and from which both parties deterministically derive a problem as per $P \leftarrow \mathrm{H}_x(r)$. The prover, using its witness, computes the set $\mathbf{S}$ of all solutions of P, denotes one of them with S, and sends S to the verifier. (If P has no solution, the prover sends $\perp$.) The verifier accepts (meaning: concludes that $x \in \mathcal{L}$) iff $S \neq \perp$ and S is indeed a solution for P. Importantly, while the prover selects the solution S within set $\mathbf{S}$ in a deterministic way (so that for each seed r and thus problem P it consistently exposes the same solution even if queried multiple times), from the point of view of the verifier the solution S is picked uniformly at random from the set of all solutions of P. This behavior is implemented by letting the prover make its selection based on an additional random oracle that is made private to the prover by including the witness w in each query. Theorem 1 assesses the correctness and security of the protocol.

Protocol Hash-then-Solve

Verifier (on input $x \in \mathcal{U}$)		**Prover** (on input $x \in \mathcal{L}, w \in \mathcal{W}_x$)
00 $r \leftarrow_{\$} \{0,1\}^l$		⋮
01 Send r	$\longrightarrow$	06 Receive r
⋮		07 $P \leftarrow \mathrm{H}_x(r)$
		08 $\mathbf{S} \leftarrow \mathrm{Solve}(x, w, P)$
		09 $S \leftarrow (\mathbf{S} \neq \emptyset)\ ?\ \$_P(\mathbf{S}) : \perp$
02 Receive S	$\longleftarrow$	10 Send S
03 If $S = \perp$: Return 0		
04 $P \leftarrow \mathrm{H}_x(r)$		
05 Return $\mathrm{Verify}(x, P, S)$		

Fig. 1. Hash-then-Solve: Random-oracle based version of the GMR protocol from [19]. Specifications of the three CRP algorithms can be readily extracted from the code: algorithm V_1 is in lines 00–01, algorithm V_2 is in lines 02–05, and algorithm P is in lines 06–10. The expression of the form $S \leftarrow \$_P(\mathbf{S})$ in line 09 is an abbreviation for $S \leftarrow \mathrm{RO}(x, w, P, \mathbf{S})$, where $\mathrm{RO}\colon \{0,1\}^* \rightarrow \mathbf{S}$ is a (private) random oracle.

Theorem 1. *The Hash-then-Solve protocol defined in Fig. 1 is δ-correct and ε-sound and perfectly zero-knowledge, where*

$$\delta = \min \Delta(\mathcal{L}) \qquad \text{and} \qquad \varepsilon = 1 - \max \Delta(\mathcal{U} \setminus \mathcal{L}),$$

if hash functions $(\mathrm{H}_x)_{x \in \mathcal{U}}$ are modeled as random oracles. For this result we assume that the Sample* *algorithm is both problem-uniform and solution-uniform.*

Proof. Correctness. Let $x \in \mathcal{L}$ and $w \in \mathcal{W}_x$. Since H_x is modeled as a random oracle, problem P assigned in line 07 is uniformly distributed in $\mathcal{P}_x$. Set $\mathbf{S}$ from line 08 is empty if $P \in \mathcal{P}_x^-$ and contains elements if $P \in \mathcal{P}_x^+$. The probability that the prover outputs a solution, and that the verifier accepts it in line 05, is thus precisely $|\mathcal{P}_x^+|/|\mathcal{P}_x|$. A lower bound for this value is $\delta = \min \Delta(\mathcal{L})$.

Soundness. Let $x \in \mathcal{U} \setminus \mathcal{L}$. A necessary condition for the verifier to accept in line 05 is that there exists a solution to problem $P = \mathrm{H}_x(r)$, i.e., that $P \in \mathcal{P}_x^+$. Since H_x is modeled as a random oracle, P is uniformly distributed in $\mathcal{P}_x$. The probability of P having a solution is thus $|\mathcal{P}_x^+|/|\mathcal{P}_x|$. This value is at most $\max \Delta(\mathcal{U} \setminus \mathcal{L})$. Thus $\varepsilon = 1 - \max \Delta(\mathcal{U} \setminus \mathcal{L})$ is a lower bound for the probability of the verifier not accepting in a protocol run.

Zero-knowledge. We show that the protocol is zero-knowledge by specifying and analyzing a simulator S. Its code is in Fig. 2. The prover oracle $\mathrm{P}(x, w, \cdot)$ and the random oracle $\mathrm{H}_x(\cdot)$ are simulated by algorithms $\mathrm{P}_{\mathrm{sim}}$ and $\mathrm{H}_{\mathrm{sim}}$, respectively. Associative array $\mathbf{R}$ reflects the input-output map of the random oracle and is initialized such that all inputs map to special value $\perp$. If $\mathrm{H}_{\mathrm{sim}}$ is queried on a seed r, a fresh problem-solution pair is sampled using the Sample* algorithm, the pair is registered in $\mathbf{R}$, and the problem part is returned to the caller. Note that by the assumed problem-uniformity of Sample$^*(x)$ this is an admissible implementation of a random oracle that maps to set $\mathcal{P}_x$.

The task of the $\mathrm{P}_{\mathrm{sim}}$ algorithm is to return, for any seed r, a uniformly picked solution for the problem $P = \mathrm{H}_x(r)$; if no solution exists, the oracle shall return $\perp$. This is achieved by returning the solution part of the problem-solution pair that was sampled using Sample* when processing the random oracle query $\mathrm{H}_x(r)$. Note that this argument uses both the solution uniformity of Sample* and the fact that the P algorithm from Fig. 1 is deterministic and in particular always outputs the same solution if a seed is queried multiple times to a $\mathrm{P}(x, w, \cdot)$ prover. □

Oracle $\mathrm{P}_{\mathrm{sim}}(r)$	**Oracle** $\mathrm{H}_{\mathrm{sim}}(r)$
00 If $\mathbf{R}[r] = \perp$:	06 If $\mathbf{R}[r] = \perp$:
01 $(P, S) \leftarrow_{\$}$ Sample$^*(x)$	07 $(P, S) \leftarrow_{\$}$ Sample$^*(x)$
02 $\mathbf{R}[r] \leftarrow (P, S)$	08 $\mathbf{R}[r] \leftarrow (P, S)$
03 $(P, S) \leftarrow \mathbf{R}[r]$	09 $(P, S) \leftarrow \mathbf{R}[r]$
04 If $S = \perp$: Return $\perp$	10 Return P
05 Return S	

Fig. 2. Simulator S. Associative array $\mathbf{R}$ is initialized as per $\mathbf{R}[\cdot] \leftarrow \perp$, i.e., such that all values initially map to $\perp$. Note that lines 00–02 become redundant if one requires (w.l.o.g.) that $\mathrm{H}_{\mathrm{sim}}(r)$ is always queried before $\mathrm{P}_{\mathrm{sim}}(r)$.

4.2 Our New Protocol: Solve-then-Hash

We propose a new challenge-response protocol for the word decision problem in languages. Like the one from Sect. 4.1 it uses a random oracle, but it does so in a quite different way: The random oracle is not used for generating problems, but for hashing solutions. The advantage is that constructing a random oracle that maps into a problem space might be difficult (for certain problem spaces), while hashing solutions to bit strings is always easy.

Let $\mathcal{L} \subseteq \mathcal{U} \subseteq \Sigma^*$ be as in Sect. 3.1. Let $\mathcal{H}$ be a finite set and $\mathrm{H}\colon \{0,1\}^* \to \mathcal{H}$ a hash function (in the security reduction: a random oracle). The idea of the protocol is that the verifier samples a problem-solution pair (P, S) and communicates the problem to the prover, the latter then, using its witness, computes the sets $\mathbf{S}$ of all solutions of P and $\mathbf{h}$ of hash values of these solutions, and returns set $\mathbf{h}$ to the verifier, and the verifier finally checks whether the hash value h of S is contained in this set. An important detail is that the prover uses pseudorandom bit-strings to pad the returned set of hash values to constant-size: If $k = \max \#\mathcal{L}$ is the maximum number of solutions of problems associated with valid candidates, then the prover exclusively outputs sets $\mathbf{h}$ of this cardinality. The algorithms of the corresponding challenge-response protocol are specified in Fig. 3. (Note that when transmitting $\mathbf{h}$ from the prover to the verifier an encoding has to be chosen that hides the order in which elements were added to $\mathbf{h}$.) The analysis of our protocol is in Theorem 2. The main technical challenge of the proof is that it has to deal with collisions of the random oracle (two or more solutions might hash to the same string).

Protocol Solve-then-Hash

Verifier (on input $x \in \mathcal{U}$)		**Prover** (on input $x \in \mathcal{L}, w \in \mathcal{W}_x$)
00 $(P, S) \leftarrow_{\$} \mathrm{Sample}(x)$		
01 $h \leftarrow \mathrm{H}(P, S)$		
02 Send P	$\longrightarrow$	06 Receive P (abort if $P \notin \mathcal{P}_x$)
		07 $\mathbf{S} \leftarrow \mathrm{Solve}(x, w, P)$
		08 $\{S_1, \ldots, S_t\} \leftarrow \mathbf{S}$
		09 $h_1, \ldots, h_t \leftarrow \mathrm{H}(P, S_1), \ldots, \mathrm{H}(P, S_t)$
		10 $h_{t+1}, \ldots, h_k \leftarrow \$_P^{t+1}(\mathcal{H}), \ldots, \$_P^k(\mathcal{H})$
		11 $\mathbf{h} \leftarrow \{h_1, \ldots, h_k\}$
03 Receive $\mathbf{h}$	$\longleftarrow$	12 Send $\mathbf{h}$ (hiding the order of elements)
04 Require $\lvert\mathbf{h}\rvert \leq k$		
05 Return $[h \in \mathbf{h}]$		

Fig. 3. Solve-then-Hash: Our new challenge-response protocol. We assume $k = \max \#\mathcal{L}$. Specifications of the three CRP algorithms can be readily extracted from the code: algorithm V_1 is in lines 00–02, algorithm V_2 is in lines 03–05, and algorithm P is in lines 06–12. In line 08, the cardinality of set $\mathbf{S}$ is denoted with t. Expressions of the form $h \leftarrow \$_v^u(\mathcal{H})$ in line 10 are abbreviations for $h \leftarrow \mathrm{RO}(x, w, u, v)$, where $\mathrm{RO}\colon \{0,1\}^* \to \mathcal{H}$ is a (private) random oracle.

Theorem 2. *Let $k = \max \#\mathcal{L}$, $m = \min \#(\mathcal{U} \setminus \mathcal{L})$, and $M = \max \#(\mathcal{U} \setminus \mathcal{L})$, such that $k \leq m \leq M$. Then the Solve-then-Hash protocol defined in Fig. 3 is perfectly correct and ε-sound and perfectly zero-knowledge, where*

$$\varepsilon = 1 - \big(k/m + k/|\mathcal{H}| + (\min(M, q))^2/|\mathcal{H}|\big) \approx 1 - k/m,$$

if H *is modeled as a random oracle and q is the maximum number of random oracle queries posed by any (dishonest) prover* P^**. For this result we assume that the* Sample *algorithm is both problem-uniform and solution-uniform.*

Proof. Correctness. Let $x \in \mathcal{L}$ and $w \in \mathcal{W}_x$. Then for (P, S) from line 00 we have $S \in \mathbf{S}$ in line 07. Further, as $x \in \mathcal{L}$ we have $t \leq k = \max \#\mathcal{L}$ in line 08 and thus $|\mathbf{h}| \leq k$ in line 04 and $h \in \mathbf{h}$ in line 05. Thus V_2 accepts with probability 1.

Soundness. Let $x \in \mathcal{U} \setminus \mathcal{L}$ be an invalid candidate and P^* a (malicious) prover. Let *Win* denote the event that P^* succeeds in finding a response $\mathbf{h}$ such that verifier V_2 accepts, i.e. the event $\{(h, P) \leftarrow_{\$} \mathrm{V}_1(x); \mathbf{h} \leftarrow_{\$} \mathrm{P}^*(x, P) : \mathrm{V}_2(h, \mathbf{h}) \Rightarrow 1\}$. Recall that $\mathcal{S}ol_x(P)$ denotes the set of solutions of problem P, and let $S_1, \ldots, S_l \in \mathcal{S}ol_x(P)$ denote the solutions to the problem on which P^* queries random oracle H, i.e., the elements such that P^* queries for $\mathrm{H}(P, S_i)$ with $i \in \{1, \ldots, l\}$. We define $Col = \{\exists i \neq j : \mathrm{H}(P, S_i) = \mathrm{H}(P, S_j)\}$ as the event that the hash values of at least two of the queried solutions collide. We have

$$\begin{aligned}\Pr[\mathit{Win}] &= \Pr[\mathit{Win} \mid \mathit{Col}] \Pr[\mathit{Col}] + \Pr[\mathit{Win} \mid \neg \mathit{Col}] \Pr[\neg \mathit{Col}] \\ &\leq \Pr[\mathit{Col}] + \Pr[\mathit{Win} \mid \neg \mathit{Col}] .\end{aligned}$$

We conclude that $\Pr[\mathit{Win}] < k/m + k/|\mathcal{H}| + (\min(M, q))^2/|\mathcal{H}|$ by showing that

$$\text{(a)}\ \Pr[\mathit{Col}] < (\min(M, q))^2/|\mathcal{H}|$$

and

$$\text{(b)}\ \Pr[\mathit{Win} \mid \neg \mathit{Col}] \leq k/m + k/|\mathcal{H}|.$$

For claim (a), note that $x \in \mathcal{U} \setminus \mathcal{L}$ implies that the set $\mathcal{R}el_x(P)$ of solutions of problem P has at most $\max \#(\mathcal{U} \setminus \mathcal{L}) = M$ elements. P^* makes at most q queries to H. Hence $l \leq \min(M, q)$. We obtain

$$\begin{aligned}\Pr[\mathit{Col}] &= \Pr[\exists i \neq j : \mathrm{H}(P, S_i) = \mathrm{H}(P, S_j)] \\ &\leq l^2 \Pr[\mathrm{H}(P, S_1) = \mathrm{H}(P, S_2)] \leq \min(M, q)^2/|\mathcal{H}|,\end{aligned}$$

where the last two inequalities hold since H is modeled as a random oracle.

We conclude the proof by showing claim (b). Recall that S is the solution sampled alongside problem P. Since algorithm Sample is solution-uniform, S is distributed uniformly in $\mathcal{S}ol_x(P)$, which implies that $\mathrm{H}(P, S)$ is uniformly distributed in $\{\mathrm{H}(P, S') : S' \in \mathcal{S}ol_x(P)\}$. Note that $|\mathcal{S}ol_x(P)| \geq m = \min \#(\mathcal{U} \setminus \mathcal{L})$ and that —conditioned on $\neg \mathit{Col}$— all values $\mathrm{H}(P, S')$ that P^* knows are distinct. Conditioned on the events $S \in \{S_1, \ldots, S_l\}$ and $\neg \mathit{Col}$, prover P^* guesses $\mathrm{H}(P, S)$ with probability at most $1/l$. If, on the other hand, $S \notin \{S_1, \ldots, S_l\}$, then $\mathrm{H}(P, S)$ is uniformly distributed from P^*'s point of view. Hence its best chance of guessing it is $1/|\mathcal{H}|$. Note that $\Pr[S \in \{S_1, \ldots, S_l\}] \leq l/m$. Summing up —conditioned on $\neg \mathit{Col}$— P^*'s chance of correctly guessing $\mathrm{H}(P, S)$ is bounded by $l/m \cdot 1/l + 1/|\mathcal{H}| = 1/m + 1/|\mathcal{H}|$. Event *Win* according to line 04 cannot occur if $\mathbf{h}$ contains more than k elements, so we obtain $\Pr[\mathit{Win} \mid \neg \mathit{Col}] \leq k/m + k/|\mathcal{H}|$.

Zero-knowledge. We show that the protocol is zero-knowledge by specifying and analyzing a simulator S. Its code is in Fig. 4. The prover oracle $\mathrm{P}(x, w, \cdot)$ and the random oracle $\mathrm{H}(\cdot, \cdot)$ are simulated by algorithms $\mathrm{P}_{\mathrm{sim}}$ and $\mathrm{H}_{\mathrm{sim}}$, respectively. For oracle H we assume w.l.o.g. that it is not queried twice on the same input.

Initialization
00 For all $P \in \mathcal{P}_x$:
01 $\quad \mathbf{R}_{\mathrm{U}}[P] \leftarrow \varepsilon$
02 $\quad \mathbf{R}_{\mathrm{F}}[P] \leftarrow \$_P^1(\mathcal{H}), \ldots, \$_P^k(\mathcal{H})$

Oracle $\mathrm{P}_{\mathrm{sim}}(P)$
03 $\mathbf{h} \leftarrow \mathbf{R}_{\mathrm{U}}[P] \,\|\, \mathbf{R}_{\mathrm{F}}[P]$
04 Return $\mathbf{h}$ (hiding the order of elements)

Oracle $\mathrm{H}_{\mathrm{sim}}(P, S)$
05 If $\mathrm{Verify}(x, P, S) = 0$:
06 $\quad h \leftarrow_{\$} \mathcal{H}$; Return h
07 $h_1, \ldots, h_{t-1} \,\|\, h_t, \ldots, h_k \leftarrow \mathbf{R}_{\mathrm{U}}[P] \,\|\, \mathbf{R}_{\mathrm{F}}[P]$
08 $\mathbf{R}_{\mathrm{U}}[P] \,\|\, \mathbf{R}_{\mathrm{F}}[P] \leftarrow h_1, \ldots, h_t \,\|\, h_{t+1}, \ldots, h_k$
09 Return h_t

Fig. 4. Simulator S for the protocol of Fig. 3. We require (w.l.o.g.) that $\mathrm{H}_{\mathrm{sim}}(\cdot)$ is queried at most once on each input. Expressions of the form $h \leftarrow \$_v^u(\mathcal{H})$ in line 02 are abbreviations for $h \leftarrow \mathrm{RO}(u, v)$, where $\mathrm{RO}\colon \{0,1\}^* \to \mathcal{H}$ is a (private) random oracle. In line 07, the lengths of vectors $\mathbf{R}_{\mathrm{U}}[P]$ and $\mathbf{R}_{\mathrm{F}}[P]$ are $t-1$ and $k-t+1$, respectively. In line 08, the new lengths of vectors $\mathbf{R}_{\mathrm{U}}[P]$ and $\mathbf{R}_{\mathrm{F}}[P]$ are t and $k-t$, respectively.

Core components of our simulator are the associative arrays $\mathbf{R}_{\mathrm{U}}[\cdot]$ and $\mathbf{R}_{\mathrm{F}}[\cdot]$ that associate problems with used and fresh random hash values, respectively. The simulator starts with initializing for each problem a vector of k-many fresh hash values.[4] Oracle $\mathrm{H}_{\mathrm{sim}}$ on input a problem-solution pair (P, S) checks whether S is a solution to P. If not, a random hash value is returned. Otherwise the vector of (fresh) hash values $\mathbf{R}_{\mathrm{F}}[P]$ associated to P is retrieved. The first element of this vector is taken as the response of the random oracle query; however, before the response is output, the element is appended to the vector of (used) hash values $\mathbf{R}_{\mathrm{U}}[P]$ associated to P. Note this procedure will never fail (i.e., never a value has to be taken from $\mathbf{R}_{\mathrm{F}}[P]$ after the list is emptied) since there are at most $k = \max \#\mathcal{L}$ solutions to P. Queries to $\mathrm{P}_{\mathrm{sim}}$ on input P are responded with the set $\mathbf{h}$ of all elements contained in $\mathbf{R}_{\mathrm{F}}[P]$ and $\mathbf{R}_{\mathrm{U}}[P]$, which by definition of $\mathrm{H}_{\mathrm{sim}}$ stays unchanged throughout the simulation. Since these elements are initialized as random hash values, responses to queries to $\mathrm{P}_{\mathrm{sim}}$ have the correct distribution. Furthermore, for every $S \in \mathcal{S}ol_x(P)$ we have that $\mathrm{H}_{\mathrm{sim}}(P, S)$ is contained in $\mathrm{P}_{\mathrm{sim}}(P)$. Summing up, the output of $\mathrm{P}_{\mathrm{sim}}$ and $\mathrm{H}_{\mathrm{sim}}$ is correctly distributed and simulator S provides distinguisher D with a perfect simulation of $\mathrm{P}(x, w, \cdot)$. □

4.3 Generalizing the Analysis of the Solve-then-Hash Protocol

We generalize the statement of Theorem 2, making it applicable to a broader class of languages. Recall that our protocol from Sect. 4.2 decides membership in a language $\mathcal{L} \subseteq \mathcal{U}$ if for every (invalid) candidate $x \in \mathcal{U} \setminus \mathcal{L}$ and every solvable problem $P \in \mathcal{P}_x^+$ the number $|\mathcal{S}ol_x(P)|$ of solutions to P exceeds the maximum number $\max \#\mathcal{L}$ of solutions to problems associated with valid candidates. We next relax this condition by showing that for soundness it already suffices if the

[4] Of course it is inefficient to assign to each $P \in \mathcal{P}_x$ a vector of values ahead of time. However, our code can easily be implemented in an equivalent form that uses lazy sampling.

expected value of $|\mathcal{S}ol_x(P)|$ (over randomly sampled $P \in \mathcal{P}_x^+$) exceeds $\max \#\mathcal{L}$. In order to do so, we associate to $\mathcal{L}$ and $\mathcal{U}$ the function $\varepsilon_{\mathcal{L},\mathcal{U}} \colon [0,1] \to \mathbb{R}^+$ such that

$$\varepsilon_{\mathcal{L},\mathcal{U}}(\gamma) := \min\{\varepsilon' \mid \forall x \in \mathcal{U} \setminus \mathcal{L} \colon \Pr[P \leftarrow_{\$} \mathcal{P}_x^+ : \max \#(\mathcal{L})/|\mathcal{S}ol_x(P)| \leq \varepsilon'] \geq \gamma\},$$

i.e., the function that associates to each probability value $\gamma \in [0,1]$ the smallest factor ε' such that for every invalid x a uniformly sampled problem with probability of at least γ has at least $\max \#(\mathcal{L})/\varepsilon'$ solutions.

In Theorem 3 we give a correspondingly refined soundness analysis of the Solve-then-Hash protocol. Note that, as the protocol itself did not change, the correctness and zero-knowledge properties do not require a new analysis. Note further that $\varepsilon_{\mathcal{L},\mathcal{U}}(1) = \max \#(\mathcal{L})/\min \#(\mathcal{U} \setminus \mathcal{L})$, and that thus the soundness analysis of Theorem 2 is just the special case of Theorem 3 where $\gamma = 1$.

Theorem 3. *Let $k = \max \#\mathcal{L}$ and $M = \max \#(\mathcal{U} \setminus \mathcal{L})$ such that $k \leq M$. Then for every $\gamma \in [0,1]$ the Solve-then-Hash protocol defined in Fig. 3 is perfectly correct and ε-sound and perfectly zero-knowledge, where*

$$\varepsilon = 1 - \big(\varepsilon_{\mathcal{L},\mathcal{U}}(\gamma) + (1-\gamma)/(1-c) + k/|\mathcal{H}| + c\big) \approx \gamma - \varepsilon_{\mathcal{L},\mathcal{U}}(\gamma),$$

if H *is modeled as a random oracle, q is the maximum number of random oracle queries posed by any (dishonest) prover* P^* *and $c = (\min(M,q))^2/|\mathcal{H}|$. For this result we assume that the* Sample *algorithm is both problem-uniform and solution-uniform.*

Proof. The correctness and zero-knowledge property of the protocol were already shown in the proof of Theorem 2. We thus show the bound on the soundness error. Fix $\gamma \in [0,1]$ and let $\varepsilon_{\mathcal{L},\mathcal{U}} = \varepsilon_{\mathcal{L},\mathcal{U}}(\gamma)$. Let $x \in \mathcal{U} \setminus \mathcal{L}$ be an invalid candidate and P^* a (malicious) prover. Let *Win* denote the event that P^* succeeds in finding a response $\mathbf{h}$ such that verifier V_2 accepts, i.e. the event $\{(h,P) \leftarrow_{\$} \mathrm{V}_1(x); \mathbf{h} \leftarrow_{\$} \mathrm{P}^*(x,P) : \mathrm{V}_2(h,\mathbf{h}) \Rightarrow 1\}$. Recall that $\mathcal{S}ol_x(P)$ denotes the set of solutions of problem P, and let $S_1, \ldots, S_l \in \mathcal{S}ol_x(P)$ denote the solutions to the problem on which P^* queries random oracle H, i.e., the elements such that P^* queries for $\mathrm{H}(P,S_i)$ with $i \in \{1,\ldots,l\}$. We define $Col = \{\exists i \neq j : \mathrm{H}(P,S_i) = \mathrm{H}(P,S_j)\}$ as the event that the hash values of at least two of the queried solutions collide. We have

$$\begin{aligned}\Pr[Win] &= \Pr[Win \mid Col] \Pr[Col] + \Pr[Win \mid \neg Col] \Pr[\neg Col] \\ &\leq \Pr[Col] + \Pr[Win \mid \neg Col].\end{aligned}$$

We conclude that $\Pr[Win] < \varepsilon_{\mathcal{L},\mathcal{U}} + (1-\gamma)/(1-c) + k/|\mathcal{H}| + c$ by showing that

$$\text{(a)}\ \Pr[Col] < (\min(M,q))^2/|\mathcal{H}| = c$$

and

$$\text{(b)}\ \Pr[Win \mid \neg Col] \leq \varepsilon_{\mathcal{L},\mathcal{U}} + (1-\gamma)/(1-c) + k/|\mathcal{H}|.$$

Claim (a) follows as in the proof of Theorem 2. In order to prove (b) we denote by PG the event that the problem P given as input to P^* by the verifier is "good" in the sense of having many solutions, i.e. the event $\{\max \#(\mathcal{L})/|\mathcal{S}ol_x(P)| \leq \varepsilon_{\mathcal{L},\mathcal{U}}\}$. We have

$$\begin{aligned}\Pr[\mathit{Win} \mid \neg \mathit{Col}] &= \Pr[\mathit{Win} \mid \neg \mathit{Col} \wedge \mathit{PG}] \Pr[\mathit{PG} \mid \neg \mathit{Col}] \\ &\quad + \Pr[\mathit{Win} \mid \neg \mathit{Col} \wedge \neg \mathit{PG}] \Pr[\neg \mathit{PG} \mid \neg \mathit{Col}] \\ &\leq \Pr[\mathit{Win} \mid \neg \mathit{Col} \wedge \mathit{PG}] + \Pr[\neg \mathit{PG} \mid \neg \mathit{Col}] \\ &\leq \Pr[\mathit{Win} \mid \neg \mathit{Col} \wedge \mathit{PG}] + \Pr[\neg \mathit{PG}] / \Pr[\neg \mathit{Col}].\end{aligned}$$

As stated above, we have $\Pr[\neg \mathit{Col}] \geq 1 - c$. Further, by problem-uniformity, P is distributed uniformly on $\mathcal{P}_x^+$ and by the definition of $\varepsilon_{\mathcal{L},\mathcal{U}}$ we have $\Pr[\neg \mathit{PG}] \leq 1 - \gamma$. Hence $\Pr[\neg \mathit{PG}] / \Pr[\neg \mathit{Col}] \leq (1-\gamma)/(1-c)$ and it remains to show that $\Pr[\mathit{Win} \mid \neg \mathit{Col} \wedge \mathit{PG}] \leq \varepsilon_{\mathcal{L},\mathcal{U}} + k/|\mathcal{H}|$. Since S is sampled with (solution-uniform) Sample, it is distributed uniformly on $\mathcal{S}ol_x(P)$, which implies that $\mathrm{H}(P,S)$ is uniformly distributed on $\{\mathrm{H}(P,S') : S' \in \mathcal{S}ol_x(P)\}$. Recall that $k = \max \#\mathcal{L}$. If event PG occurs then $|\mathcal{S}ol_x(P)| \geq k/\varepsilon_{\mathcal{L},\mathcal{U}}$. Further —conditioned on $\neg \mathit{Col}$— all values $\mathrm{H}(P,S')$ that P^* knows are distinct. Conditioned on the events $S \in \{S_1, \ldots, S_l\}$, PG and $\neg \mathit{Col}$ prover P^* guesses $\mathrm{H}(P,S)$ with probability at most $1/l$. If, on the other hand, $S \notin \{S_1, \ldots, S_l\}$, then from P^*'s point of view $\mathrm{H}(P,S)$ is uniformly distributed on $\mathcal{H}$. Hence in this case its best chance of guessing it is $1/|\mathcal{H}|$. Note that $\Pr[S \in \{S_1, \ldots, S_l\} \mid \neg \mathit{Col} \wedge \mathit{PG}] \leq l \cdot \varepsilon_{\mathcal{L},\mathcal{U}}/k$. Summing up —conditioned on $\neg \mathit{Col}$ and PG— prover P^*'s chance of correctly guessing $\mathrm{H}(P,S)$ is bounded by $l\varepsilon_{\mathcal{L},\mathcal{U}}/k \cdot 1/l + 1/|\mathcal{H}| = \varepsilon_{\mathcal{L},\mathcal{U}}/k + 1/|\mathcal{H}|$. Event Win according to line 04 cannot occur if $\mathbf{h}$ contains more than k elements, so we obtain $\Pr[\mathit{Win} \mid \neg \mathit{Col}] \leq \varepsilon_{\mathcal{L},\mathcal{U}} + k/|\mathcal{H}|$. □

5 Challenge-Response Protocols in the Domain of Number-Theory

We provide several protocols to prove number theoretic properties of a number $N \in \mathbb{N}$, the corresponding witness being the factorization of N. More formally, we consider the universe

$$\mathcal{L}_{\mathrm{odd}} = \{N \in \mathbb{N} : \nu_2 = 0; |\mathbb{P}(N)| \geq 2\}$$

of odd numbers, which have at least two prime factors. Note that $\mathcal{L}_{\mathrm{odd}}$ can be efficiently decided. We associate problem and solution spaces as defined in Sect. 3.1 to several languages $\mathcal{L} \subseteq \mathcal{L}_{\mathrm{odd}}$, hence obtaining membership checking protocols via Theorems 1 and 2. In most cases the problem and solution space associated to a statement $N \in \mathcal{L}_{\mathrm{odd}}$ are defined as $\mathbb{Z}_N^*$, while the defining relation $\mathcal{R}el_N$ for problem b and solution a is of the type $b \equiv a^e \bmod N$, where the exponent e is chosen according to the number theoretic property of N we want to prove. Equation (1) of Sect. 2.2 serves as a primary tool to deduce bounds on $\max \#(\mathcal{L})$ and $\min \#(\mathcal{L}_{\mathrm{odd}} \setminus \mathcal{L})$. Defining $\mathcal{R}el_N$ in the described way enables

us to to sample from it as follows. Algorithm Sample first chooses a solution a uniformly from $\mathcal{S}_N = \mathbb{Z}_N^*$. Then the corresponding problem b is set to a^e. In this way a is uniformly distributed on $\mathcal{S}ol_N(b)$ and the proposed algorithm samples *solution-uniformly* (for both valid and invalid candidates) as required for the Solve-then-Hash protocol of Sect. 4.2.

For some of the considered languages the map $a \mapsto a^e$ defines a permutation on $\mathbb{Z}_N^*$ for every valid statement $N \in \mathcal{L}$. In this case every problem is solvable, we hence have $\mathcal{P}_N^+ = \mathcal{P}_N$, and the described sampling algorithm also fulfills the property of *problem-uniformity* and can be used in the Hash-then-Solve protocol of Sect. 4.1. For other of the considered languages the space $\mathcal{P}_N^+$ of solvable problems is a proper subset of $\mathcal{P}_N$ and it seems not feasible to construct an algorithm with the desired properties. In this cases only the Solve-then-Hash protocol can be used to decide the language.

Considered languages. We provide a toolbox of protocols checking arguably the most important properties required of RSA-type moduli. An overview of our results is given in Table 1. Combining several of the protocols gives a method to check for properties required of typical applications. For example the property that the RSA map $a \mapsto a^e \bmod N$ defined by numbers (N, e) is "good" can be checked by showing that N has exactly two prime factors and is square free and that e indeed defines a permutation on $\mathbb{Z}_N^*$. If an application requires a feature more specific than the ones we treat, then likely corresponding problem and solution spaces and a corresponding relation can be found. As a starting point we consider the languages

$$\mathcal{L}_{\text{sf}} := \{N \in \mathcal{L}_{\text{odd}} : \gcd(N, \varphi(N)) = 1\}$$
$$\mathcal{L}_{\text{ppp}} := \{N \in \mathcal{L}_{\text{odd}} : |\mathbb{P}(N)| = 2\}$$

of square free numbers and prime power products, i.e. numbers having exactly two prime factors. For both languages the corresponding relation was implicitly

Table 1. Protocols for properties of RSA moduli. Assume $k = \max \#\mathcal{L}$ and $m = \min \#(\mathcal{U} \setminus \mathcal{L})$. Columns seven and eight indicate whether the Hash-then-Solve (HtS) or Solve-then-Hash (StH) protocol can be used to decide $\mathcal{L}$. $\mathcal{L}_{\text{pp}}$ and $\mathcal{L}_{\text{rsa}}$ are intersections of other decidable languages and can be decided by running the corresponding protocols in parallel.

$\mathcal{L}$	$\mathcal{U}$	$\mathcal{P}_N$	$\mathcal{S}_N$	$\mathcal{R}el_N$	k/m	**HtS**	**StH**	Sections
$\mathcal{L}_{\text{sf}}$	$\mathcal{L}_{\text{odd}}$	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	(a^n, a)	1/3	✓	✓	5.1
$\mathcal{L}_{\text{ppp}}$	$\mathcal{L}_{\text{odd}}$	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	(a^2, a)	1/2		✓	5.2
$\mathcal{L}_{\text{per}}$	$\mathcal{L}_{\text{odd}}$	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	(a^e, a)	1/2	✓	✓	5.3
$\mathcal{L}_{\text{pp}}$	$\mathcal{L}_{\text{odd}}$	$(\mathbb{Z}_N^*)^2$	$(\mathbb{Z}_N^*)^2$		1/2		✓	5.4
$\mathcal{L}_{\text{rsa}}$	$\mathcal{L}_{\text{odd}}$	$(\mathbb{Z}_N^*)^3$	$(\mathbb{Z}_N^*)^3$		1/2		✓	5.4
$\mathcal{L}_{\text{blum}}$	$\mathcal{L}_{\text{pp}}$	$\mathbb{Z}_N^*$	$\mathbb{Z}_N^*$	(a^4, a)	1/2		✓	5.5
$\mathcal{L}_{\text{pai}}$	$\mathcal{L}_{\text{pp}}$	$\mathbb{Z}_{n^2}^*$	$\mathbb{Z}_n \times \mathbb{Z}_N^*$	$(f_{(n,g)}(a), a)$	1/2	✓	✓	5.6

given in [19]. Note that by definition of $\varphi(N)$ condition $(\gcd(\varphi(N), N) = 1)$ implies that $\nu_p = 1$ for every $p \in \mathbb{P}(N)$ and hence indeed the number is square free. Due to the choice of the relation it additionally implies that $p \nmid (q-1)$ for every $p, q \in \mathbb{P}(N)$. Intersecting both languages yields the language

$$\mathcal{L}_{\mathrm{pp}} := \{pq \in \mathcal{L}_{\mathrm{odd}} : p, q \in \mathbb{P}, p \neq q, p \nmid (q-1), q \nmid (p-1)\}$$

of prime products. Each N in this language is the product of two distinct primes, a minimal requirement on RSA moduli. We further give relations for the languages

$$\mathcal{L}_{\mathrm{per}} := \{(N, e) \in \mathcal{L}_{\mathrm{odd}} \times \mathbb{N} : a \mapsto a^e \text{ defines a permutation}\}$$
$$\mathcal{L}_{\mathrm{rsa}} := \{(N, e) \in \mathcal{L}_{\mathrm{pp}} \times \mathbb{N} : a \mapsto a^e \text{ defines a permutation}\}$$

of pairs (N, e) such that exponentiation with e defines a permutation on $\mathbb{Z}_N^*$ and N being a prime product such that e defines a permutation on $\mathbb{Z}_N^*$. The relations were implicitly used in [11,34]. Building on the protocol for $\mathcal{L}_{\mathrm{pp}}$ we consider the language

$$\mathcal{L}_{\mathrm{blum}} := \{pq \in \mathcal{L}_{\mathrm{pp}} : p \equiv q \equiv 3 \bmod 4\}$$

of Blum integers, i.e. prime products with both primes being equal to 3 modulo 4. We give problem and solution spaces and a corresponding relation, which up to our knowledge has not been used so far, such that $\mathcal{L}_{\mathrm{blum}}$ can be decided in universe $\mathcal{L}_{\mathrm{pp}}$. Finally, we show that it can be efficiently decided whether the trapdoor function corresponding to Paillier's encryption scheme, which corresponds to pairs (N, g) consisting of a prime product N and an element g of $\mathbb{Z}_{N^2}^*$, indeed defines a bijection. A protocol for this property has up to our knowledge not been given so far. Note that given (N, g) it is assumed to be hard to decide whether the corresponding map is bijective, since it has been shown to be a lossy trapdoor function under the decisional quadratic residuosity assumption [15].

5.1 Deciding $\mathcal{L}_{\mathrm{sf}}$

Consider the language

$$\mathcal{L}_{\mathrm{sf}} := \{N \in \mathcal{L}_{\mathrm{odd}} : \gcd(N, \varphi(N)) = 1\}$$

of square free integers, i.e. of odd numbers such that for every $p, q \in \mathbb{P}(N)$ we have $\nu_p = 1$ and $p \nmid q-1$. We show that $\mathcal{L}_{\mathrm{sf}}$ can be decided in universe $\mathcal{L}_{\mathrm{odd}}$. For a statement $N \in \mathcal{L}_{\mathrm{odd}}$ let the corresponding witness be its factorization. We define the corresponding problem and solution spaces and the defining relation as

$$\begin{aligned} \mathcal{P}_N &= \mathbb{Z}_N^* \\ \mathcal{S}_N &= \mathbb{Z}_N^* \\ \mathcal{R}el_N &= \{(b, a) \in (\mathbb{Z}_N^*)^2 : b \equiv a^N \bmod N\}. \end{aligned}$$

$\mathcal{R}el_N$ is defined via the map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^N$. By Eq. (1) of Sect. 2.2 this map is a bijection exactly if $N \in \mathcal{L}_{\mathrm{sf}}$, i.e. if $\gcd(N, \varphi(N)) = 1$, and, since N is odd, at least 3-to-1 if $N \in \mathcal{L}_{\mathrm{odd}} \setminus \mathcal{L}_{\mathrm{sf}}$. Hence $\max \#(\mathcal{L}_{\mathrm{sf}}) = 1$ and $\min \#(\mathcal{L}_{\mathrm{odd}} \setminus \mathcal{L}_{\mathrm{sf}}) = 3$.

We now describe the corresponding algorithms. Algorithms Sample samples from $\mathcal{R}el_N$ by choosing $a \leftarrow_{\$} \mathbb{Z}_N^*$, setting $b \leftarrow a^N$ and returning the problem-solution pair (b, a). As discussed above, since the solution a is sampled at random and the corresponding problem b is derived from it afterwards, a is uniformly distributed on $\mathcal{S}ol_N(b)$ and Sample is solution-uniform. Verify on input (b, a) checks whether $b \equiv a^n \bmod n$ and responds accordingly. Note that Nth roots modulo N can be efficiently computed given the factorization of N. Hence it is possible to construct the problem solving algorithm Solve and by Theorem 2 language $\mathcal{L}_{\mathrm{sf}}$ can be decided using the Solve-then-Hash protocol.

For every valid statement $N \in \mathcal{L}_{\mathrm{sf}}$ the map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^N$ defining the relation $\mathcal{R}el_N$ is a bijection. Hence in this case every problem $b \in \mathcal{P}_N$ is solvable. Further the problems sampled by Sample are uniformly distributed on $\mathcal{P}_N$ and solutions are uniformly distributed on the corresponding solution set $\mathcal{S}ol_N(b)$. Thus Sample is both problem-uniform and solution-uniform, and therefore fulfills the requirements, which are necessary to be used as sampling algorithm Sample* in the Hash-then-Solve protocol of Sect. 4.1.

5.2 Deciding $\mathcal{L}_{\mathrm{ppp}}$

Consider the language

$$\mathcal{L}_{\mathrm{ppp}} := \{N \in \mathcal{L}_{\mathrm{odd}} : |\mathbb{P}(N)| = 2\}$$

of prime power products, i.e. of odd numbers that have exactly two prime factors. We show that $\mathcal{L}_{\mathrm{ppp}}$ can be decided in universe $\mathcal{L}_{\mathrm{odd}}$. For a statement $N \in \mathcal{L}_{\mathrm{odd}}$ let the corresponding witness be its factorization. We define the corresponding problem and solution spaces and the defining relation as

$$\begin{aligned} \mathcal{P}_N &= \mathbb{Z}_N^* \\ \mathcal{S}_N &= \mathbb{Z}_N^* \\ \mathcal{R}el_N &= \{(b, a) \in (\mathbb{Z}_N^*)^2 : b \equiv a^2 \bmod N\}. \end{aligned}$$

$\mathcal{R}el_N$ is defined via the map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^2$. Since N is odd we obtain by Eq. (1) of Sect. 2.2 that this map is 4-to-1 if $N \in \mathcal{L}_{\mathrm{ppp}}$, i.e. if N has at most 2 distinct prime factors, and at least 8-to-1 if $N \in \mathcal{L}_{\mathrm{odd}} \setminus \mathcal{L}_{\mathrm{ppp}}$. Hence $\max \#(\mathcal{L}_{\mathrm{ppp}}) = 4$ and $\min \#(\mathcal{L}_{\mathrm{odd}} \setminus \mathcal{L}_{\mathrm{ppp}}) = 8$.

We now describe the corresponding algorithms. Algorithm Sample samples from $\mathcal{R}el_N$ by choosing $a \leftarrow_{\$} \mathbb{Z}_N^*$, setting $b \leftarrow a^2$ and returning the problem-solution pair (b, a). Note that Sample is solution-uniform. Verify on input (b, a) checks whether $b \equiv a^2 \bmod N$ and responds accordingly. Note that square roots modulo N can be efficiently computed given the factorization of N. Hence it is possible to construct the problem solving algorithm Solve and by Theorem 2 language $\mathcal{L}_{\mathrm{ppp}}$ can be decided using the Solve-then-Hash protocol.

Let $N \in \mathcal{L}_{\text{ppp}}$ be a valid statement. The set $\mathcal{P}_N^+$ of solvable problems is the set $\mathbf{QR}(N)$ of quadratic residues modulo N. Hence a sampling algorithm Sample* compatible with the Hash-then-Solve protocol of Sect. 4.1 would require that (a) the sampled problems are uniformly distributed in $\mathbb{Z}_N^*$ and (b) if a sampled problem is solvable then it is accompanied by a solution. While both sampling uniformly from $\mathbb{Z}_N^*$ or sampling uniformly from $(b, a) \in \mathcal{R}el_N \subseteq \mathbf{QR}(N) \times \mathbb{Z}_N^*$ is easy, it is unclear how to construct an algorithm with the required properties that does not need access to the factorization of N. The authors of [19] overcome this problem by imposing additional requirements on N. They give a protocol able to verify that $pq = N \in \mathcal{L}_{\text{ppp}}$ such that $p, q \not\equiv 1 \bmod 8$ and $p \not\equiv q \bmod 8$. For this restricted language exactly one element of the set $\{+b, -b, +2b, -2b\}$ has a square root for every $b \in \mathbb{Z}_N^*$. Changing the relation to pairs (b, a), such that a is the root of one of those elements one then defines Sample* to sample (b, a) with algorithm Sample from above and then output $(c\,b, a)$, where $c \leftarrow_{\$} \{+1, -1, +2, -2\}$.

5.3 Deciding $\mathcal{L}_{\text{per}}$

Consider the language

$$\mathcal{L}_{\text{per}} := \{(N, e) \in \mathcal{L}_{\text{odd}} \times \mathbb{N} : a \mapsto a^e \text{ defines a permutation}\}$$

of pairs (N, e) such that the map $a \mapsto a^e$ defines a permutation. We show that $\mathcal{L}_{\text{per}}$ can be decided in universe $\mathcal{L}_{\text{odd}}$. For a statement $N \in \mathcal{L}_{\text{odd}}$ let the corresponding witness be its factorization. We define the corresponding problem and solution spaces and the defining relation as

$$\begin{aligned} \mathcal{P}_N &= \mathbb{Z}_N^* \\ \mathcal{S}_N &= \mathbb{Z}_N^* \\ \mathcal{R}el_N &= \{(b, a) \in (\mathbb{Z}_N^*)^2 : b \equiv a^e \bmod N\}. \end{aligned}$$

$\mathcal{R}el_N$ is defined via the map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^e$. Since this map is a homomorphism, it is at least 2-to-1 if it is not bijective. Hence $\max \#(\mathcal{L}_{\text{sf}}) = 1$ and $\min \#(\mathcal{L}_{\text{odd}} \setminus \mathcal{L}_{\text{sf}}) = 2$.

We now describe the corresponding algorithms. Algorithm Sample samples from $\mathcal{R}el_N$ by choosing $a \leftarrow_{\$} \mathbb{Z}_N^*$, setting $b \leftarrow a^e$ and returning the problem-solution pair (b, a). Note that Sample is both problem-uniform and solution-uniform. Verify on input (b, a) checks whether $b \equiv a^e \bmod N$ and responds accordingly. Note that eth roots modulo N can be efficiently computed given the factorization of N. Hence it is possible to construct the problem solving algorithm Solve and by Theorem 2 language $\mathcal{L}_{\text{per}}$ can be decided using the Solve-then-Hash protocol.

Further, for every valid statement $N \in \mathcal{L}_{\text{per}}$ the map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^e$ defining the relation $\mathcal{R}el_N$ is a bijection. Hence in this case every problem $b \in \mathcal{P}_N$ is solvable. Further the problems sampled by Sample are uniformly distributed on $\mathcal{P}_N$ and solutions are uniformly distributed on the corresponding solution

set $\mathcal{S}ol_N(b)$. Thus Sample is both problem-uniform and solution-uniform, and therefore fulfills the requirements, which are necessary to be used as sampling algorithm Sample* in the Hash-then-Solve protocol of Sect. 4.1.

5.4 Deciding $\mathcal{L}_{\mathbf{pp}}$ and $\mathcal{L}_{\mathbf{rsa}}$

Consider the languages

$$\mathcal{L}_{\mathrm{pp}} := \{pq \in \mathcal{L}_{\mathrm{odd}} : p, q \in \mathbb{P}, p \neq q, p \nmid (q-1), q \nmid (p-1)\}$$

of prime products, i.e. square-free numbers having exactly two prime factors, and

$$\mathcal{L}_{\mathrm{rsa}} := \{(N, e) \in \mathcal{L}_{\mathrm{pp}} \times \mathbb{N} : a \mapsto a^e \text{ defines a permutation}\}$$

of pairs (N, e) such that N is a prime product and the RSA map $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; $a \mapsto a^e$ defines a permutation. We have $\mathcal{L}_{\mathrm{pp}} = \mathcal{L}_{\mathrm{ppp}} \cap \mathcal{L}_{\mathrm{sf}}$ and $\mathcal{L}_{\mathrm{rsa}} = \mathcal{L}_{\mathrm{per}} \cap \mathcal{L}_{\mathrm{ppp}} \cap \mathcal{L}_{\mathrm{sf}}$. The protocols deciding $\mathcal{L}_{\mathrm{sf}}$, $\mathcal{L}_{\mathrm{ppp}}$ and $\mathcal{L}_{\mathrm{per}}$ are all defined with respect to the same universe $\mathcal{L}_{\mathrm{odd}}$. By running them in parallel we hence obtain protocols deciding $\mathcal{L}_{\mathrm{pp}}$ or $\mathcal{L}_{\mathrm{rsa}}$ respectively with respect to $\mathcal{L}_{\mathrm{odd}}$.

5.5 Deciding $\mathcal{L}_{\mathbf{blum}}$

Consider the language

$$\mathcal{L}_{\mathrm{blum}} := \{pq \in \mathcal{L}_{\mathrm{pp}} : p \equiv q \equiv 3 \bmod 4\}$$

of Blum integers. We show that $\mathcal{L}_{\mathrm{blum}}$ can be decided in universe $\mathcal{L}_{\mathrm{pp}}$. For a statement $N \in \mathcal{L}_{\mathrm{pp}}$ let the corresponding witness be its factorization. We define the corresponding problem and solution spaces and the defining relation as

$$\begin{aligned} \mathcal{P}_N &= \mathbb{Z}_N^* \\ \mathcal{S}_N &= \mathbb{Z}_N^* \\ \mathcal{R}el_N &= \{(b, a) \in (\mathbb{Z}_N^*)^2 : b \equiv a^4 \bmod N\}. \end{aligned}$$

Since all statements are elements of $\mathcal{L}_{\mathrm{pp}}$ and hence have two odd prime factors, every square in $\mathbb{Z}_N^*$ has four square roots. Further, if N a is Blum integer then each element of $\mathbf{QR}(N)$ has exactly one root that is again a square. This implies that every problem of $\mathcal{P}^+ = \{b \in \mathbb{Z}_N^* : b \equiv a^4 \text{ for some } a \in \mathbb{Z}_N^*\}$ has four corresponding solutions, i.e. $\max \#(\mathcal{L}_{\mathrm{sf}}) = 2$. If on the other hand $N \in \mathcal{L}_{\mathrm{pp}} \setminus \mathcal{L}_{\mathrm{blum}}$, then every element of the form $b = a^4$ has at least two square roots, which are elements of $\mathbf{QR}(N)$. Hence in this case we obtain $\min \#(\mathcal{L}_{\mathrm{pp}} \setminus \mathcal{L}_{\mathrm{blum}}) = 8$.

We now describe the corresponding algorithms. Algorithm Sample samples from $\mathcal{R}el_N$ by choosing $a \leftarrow_{\$} \mathbb{Z}_N^*$, setting $b \leftarrow a^4$ and returning the problem-solution pair (b, a). Note that Sample is solution-uniform. Verify on input (b, a) checks whether $b \equiv a^4 \bmod N$ and responds accordingly. Note that 4th roots modulo N can be efficiently computed given the factorization of N. Hence it

is possible to construct the problem solving algorithm Solve and by Theorem 2 language $\mathcal{L}_{\text{blum}}$ can be decided using the Solve-then-Hash protocol.

Let $N \in \mathcal{L}_{\text{blum}}$ be a valid statement. Since for Blum integers squaring is a permutation on $\mathbf{QR}(N)$, the space of solvable problems is given by $\mathbf{QR}(N)$. Hence as in the case of the relation for language $\mathcal{L}_{\text{ppp}}$ it seems unfeasible to construct an alternative sampling algorithm Sample* that admits the use of the Hash-then-Solve protocol of Sect. 4.1.

5.6 Deciding $\mathcal{L}_{\text{pai}}$

Let $N \in \mathcal{L}_{\text{pp}}$ and $g \in \mathbb{Z}^*_{N^2}$ such that N divides the order of the group generated by g. In this case the following function associated to N and g, which is used in Paillier's encryption scheme [26], defines a bijection that can be efficiently inverted given the factorization of N.

$$f_{n,g}\colon \begin{cases} \mathbb{Z}_N \times \mathbb{Z}^*_N & \to \mathbb{Z}^*_{N^2} \\ (a_1, a_2) & \mapsto g^{a_1}\, a_2^N \bmod N^2 \end{cases}$$

In this section we show that our protocols can be used to check in universe $\mathcal{L}_{\text{pp}}$, whether a public key (N, g) for the Paillier encryption scheme indeed defines a bijection. Hence consider the language

$$\mathcal{L}_{\text{pai}} := \{(N, g) \in \mathcal{L}_{\text{pp}} \times \mathbb{N} : g \in \mathbb{Z}^*_{N^2}, f_{N,g} \text{ is permutation}\}.$$

Note that the condition $g \in \mathbb{Z}^*_{N^2}$ can be efficiently checked. For a statement $N \in \mathcal{L}_{\text{pp}}$ let the corresponding witness be its factorization. We define the corresponding problem and solution spaces and the defining relation as

$$\begin{aligned} \mathcal{P}_N &= \mathbb{Z}^*_{N^2} \\ \mathcal{S}_N &= \mathbb{Z}_N \times \mathbb{Z}^*_N \\ \mathcal{R}el_N &= \{(b, a) \in \mathcal{P}_{(N,g)} \times \mathcal{S}_{(N,g)} : b \equiv f_{N,g}(a) \bmod N\}. \end{aligned}$$

$\mathcal{R}el_N$ is defined via map $f_{(N,g)}$, which is a homomorphism. Hence if it is not bijective it is at least 2-to-1 and we obtain $\max \#(\mathcal{L}_{\text{sf}}) = 1$ and $\min \#(\mathcal{L}_{\text{odd}} \setminus \mathcal{L}_{\text{sf}}) = 2$.

We now describe the corresponding algorithms. Algorithm Sample samples from $\mathcal{R}el_N$ by choosing $a \leftarrow_{\$} \mathbb{Z}_N \times \mathbb{Z}^*_N$, setting $b \leftarrow f_{(N,g)}(a)$ and returning the problem-solution pair (b, a). Note that Sample is both problem-uniform and solution-uniform. Verify on input (b, a) checks whether $b \equiv f_{(N,g)}(a)$ and responds accordingly. Map $f_{(N,g)}$ can be efficiently inverted given the factorization of N. Hence it is possible to construct the problem solving algorithm Solve and by Theorem 2 language $\mathcal{L}_{\text{pai}}$ can be decided using the Solve-then-Hash protocol.

For every valid statement $N \in \mathcal{L}_{\text{pai}}$ the map $f_{(N,g)}$ defining the relation $\mathcal{R}el_N$ is a bijection. Hence in this case every problem $b \in \mathcal{P}_N$ is solvable. Further the problems sampled by Sample are uniformly distributed on $\mathcal{P}_N$ and solutions are

uniformly distributed on the corresponding solution set $\mathcal{S}ol_N(b)$. Thus Sample is both problem-uniform and solution-uniform, and therefore fulfills the requirements, which are necessary to be used as sampling algorithm Sample* in the Hash-then-Solve protocol of Sect. 4.1.

The constructions can be easily adapted to handle the generalized version of the trapdoor function from [14], which uses domain $\mathbb{Z}_{N^s} \times \mathbb{Z}_N^*$ and range $\mathbb{Z}_{N^{s+1}}^*$ for some $s \in \mathbb{N}$.

Acknowledgments. We are grateful to the anonymous reviewers for their valuable comments. Benedikt Auerbach was supported by the NRW Research Training Group SecHuman. Bertram Poettering conducted part of the research at Ruhr University Bochum, supported by ERC Project ERCC (FP7/615074).

References

1. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_16
2. Bellare, M., Poettering, B., Stebila, D.: Deterring certificate subversion: efficient double-authentication-preventing signatures. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 121–151. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_5
3. Bellare, M., Yung, M.: Certifying permutations: noninteractive zero-knowledge based on any trapdoor permutation. J. Cryptol. **9**(3), 149–166 (1996)
4. Benhamouda, F., Ferradi, H., Géraud, R., Naccache, D.: Non-interactive provably secure attestations for arbitrary RSA prime generation algorithms. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 206–223. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_13
5. Berger, R., Kannan, S., Peralta, R.: A framework for the study of cryptographic protocols. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 87–103. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_9
6. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_31
7. Boyar, J., Friedl, K., Lund, C.: Practical zero-knowledge proofs: giving hints and using deficiencies. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 155–172. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_18
8. Brassard, G., Crépeau, C.: Non-transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond. In: 27th FOCS, pp. 188–195. IEEE Computer Society Press, Toronto, 27–29 October 1986
9. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_5
10. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 107–122. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_8

11. Catalano, D., Pointcheval, D., Pornin, T.: IPAKE: isomorphisms for password-based authenticated key exchange. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 477–493. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_29
12. Chan, A., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 561–575. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054154
13. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 319–327. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_25
14. Damgård, I., Jurik, M.: A generalisation, a simplication and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9
15. Freeman, D.M., Goldreich, O., Kiltz, E., Rosen, A., Segev, G.: More constructions of lossy and correlation-secure trapdoor functions. J. Cryptol. **26**(1), 39–74 (2013)
16. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052225
17. Fujisaki, E., Okamoto, T.: A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 32–46. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054115
18. Gennaro, R., Krawczyk, H., Rabin, T.: RSA-based undeniable signatures. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 132–149. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052232
19. Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: ACM CCS 1998, pp. 67–72. ACM Press, San Francisco, 2–5 November 1998
20. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. J. ACM **38**(3), 691–729 (1991)
21. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 339–358. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_21
22. Juels, A., Guajardo, J.: RSA key generation with verifiable randomness. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 357–374. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45664-3_26
23. Kakvi, S.A., Kiltz, E., May, A.: Certifying RSA. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 404–414. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_25
24. Mao, W.: Verifiable partial sharing of integer factors. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 94–105. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48892-8_8
25. Micali, S.: Fair public-key cryptosystems. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 113–138. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_9
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16

27. Poettering, B., Stebila, D.: Double-authentication-preventing signatures. Int. J. Inf. Sec. **16**(1), 1–22 (2017)
28. Shin, S., Kobara, K., Imai, H.: RSA-based password-authenticated key exchange, revisited. IEICE Trans. Inf. Syst. **91**(5), 1424–1438 (2008)
29. van de Graaf, J., Peralta, R.: A simple and secure way to show the validity of your public key. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 128–134. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_9
30. Young, A., Yung, M.: The dark side of "Black-Box" cryptography or: should we trust Capstone? In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_8
31. Young, A., Yung, M.: Kleptography: using cryptography against cryptography. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 62–74. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_6
32. Young, A., Yung, M.: The prevalence of kleptographic attacks on discrete-log based cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 264–276. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052241
33. Young, A., Yung, M.: A space efficient backdoor in RSA and its applications. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 128–143. Springer, Heidelberg (2006). https://doi.org/10.1007/11693383_9
34. Zhu, F., Wong, D.S., Chan, A.H., Ye, R.: Password authenticated key exchange based on RSA for imbalanced wireless networks. In: Chan, A.H., Gligor, V. (eds.) ISC 2002. LNCS, vol. 2433, pp. 150–161. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45811-5_11