RSA Key Generation with Verifiable Randomness

Ari Juels[1] and Jorge Guajardo[2]

[1] RSA Laboratories
Bedford, MA, USA
ajuels@rsasecurity.com
[2] Department of Electrical Engineering and Information Sciences
Ruhr-Universität Bochum, Germany
guajardo@crypto.ruhr-uni-bochum.de

Abstract. We consider the problem of proving that a user has selected and correctly employed a truly random seed in the generation of her RSA key pair. This task is related to the problem of *key validation*, the process whereby a user proves to another party that her key pair has been generated securely. The aim of key validation is to pursuade the verifying party that the user has not intentionally weakened or reused her key or unintentionally made use of bad software. Previous approaches to this problem have been *ad hoc*, aiming to prove that a private key is secure against specific types of attacks, e.g., that an RSA modulus is resistant to elliptic-curve-based factoring attacks. This approach results in a rather unsatisfying laundry list of security tests for keys.
We propose a new approach that we refer to as *key generation with verifiable randomness* (KEGVER). Our aim is to show in zero knowledge that a private key has been generated at random according to a prescribed process, and is therefore likely to benefit from the full strength of the underlying cryptosystem. Our proposal may be viewed as a kind of distributed key generation protocol involving the user and verifying party. Because the resulting private key is held solely by the user, however, we are able to propose a protocol much more practical than conventional distributed key generation. We focus here on a KEGVER protocol for RSA key generation.

Key words: certificate authority, key generation, non-repudiation, public-key infrastructure, verifiable randomness, zero knowledge

1 Introduction

In this paper, we consider the problem of demonstrating that a public key PK is well selected, in other words, that it has been chosen so as to benefit strongly from the security properties of the underlying cryptosystem. This problem has been typically refered to in the literature as that of *key validation*. Interest in key validation arises when a user registers a public key PK of some kind with a certificate authority (CA) or presents it for use in some other application, such

D. Naccache and P. Paillier (Eds.): PKC 2002, LNCS 2274, pp. 357–374, 2002.

as a group signature scheme. The structure of PK offers only limited assurance about the strength of the corresponding private key SK. For example, in the RSA cryptosystem, it may be that the public modulus n is long, ensuring security against the general number field sieve. At the same time, one of the two component primes may be short, creating vulnerability to elliptic-curve-based factoring attacks. Thus, it is easily possible for a user to generate SK of some weak form so as to render it vulnerable to any of a range of common attacks, without the knowledge of the CA. If SK is, say, a private signing key, then a malicious user of this sort can seek to repudiate transactions based on digital signatures generated using SK, claiming that the vulnerability of SK led to key compromise. The user might, for instance, place an order for a purchase of stock, and then repudiate it if the market subsequently goes down. Weakness in a key may alternatively result because a user has made inappropriate use of the same "stale" key across multiple platforms. For example, a user might choose to make use of the same key for her magazine subscription as for her financial transactions. Finally, and probably most importantly, a weak key may be produced by bad software. Faulty or malicious software might induce a subtle weakness by using a "stale" prime in RSA keys, i.e., using the same component prime in different moduli. As demonstrated by Young and Yung [36], malicious software might create a key that appears to have been correctly generated, but is primed for theft by the creator of the software, a process dubbed "kleptography". A software package may also generate a key that is weak simply because of faulty programming. This last is of perhaps the greatest concern to security architects.

Such concerns and the liability risks they create for certificate issuers have been a recurrent issue in standards bodies for some time, and have thus served as an impetus for investigation into key validation techniques. A key validation protocol aims at enabling a user to prove to a verifying party, with minimal information leakage, that her private key SK has a particular security property that may not be evident from inspection of PK. For example, researchers have proposed protocols enabling the possessor of an RSA private key to prove to a CA with little information leakage that the corresponding public modulus n is the product of two primes p and q of roughly equal length. Such a protocol is included in the appendix to the ANSI X9.31 standard for digital signatures used in financial services applications [2].

Note, however, that an RSA key can also be constructed in such a way that it is vulnerable to any of an arbitrarily long list of special-form factoring algorithms: examples of ones popular in the literature include the so-called $p-1$ attack and $p+1$ attack [27]. Recognizing that a host of different types of attacks against the RSA cryptosystem are possible, ANSI X9.31 for example includes discussion of a range of key validation tests. It is clear from the outset, though, that this kind of *ad hoc* approach is fundamentally limited: One can always devise a new type of attack and corresponding key validation protocol to add to the list, and no set of litmus tests can guard against use of a stale key.

In this paper, we propose a novel alternative to or enhancement of key validation that we refer to as *key generation with verifiable randomness*, and denote

for brevity by KEGVER. A KEGVER protocol shows not that a key is resistant to a list specific attacks, but instead that the key has been generated as an honest party would, and is therefore unlikely to be weak with respect to any known attack and unlikely to be stale. The starting point for our approach may be thought of as an ideal process in which a *trusted dealer* or *trusted third party* (TTP) generates a key pair (SK, PK) according to a universally agreed upon process, e.g., the example methods presented in the IEEE P1363 standard [1]. In this ideal process, the TTP sends (SK, PK) privately to the user and PK to the CA. The user is assured here that the privacy of her key SK is as good as if she had generated it herself. The CA is assured that the key pair (SK, PK) was generated securely, namely according to published guidelines, and therefore benefits from the full strength of the underlying cryptosystem. It should be noted that TTPs form a component of many secret sharing and key distribution schemes, e.g., [32]. The role of the TTPs in these schemes, however, is to effect a correct sharing of secrets. In our ideal process, it is to ensure correct key generation.

Of course, involvement by a TTP in real-world settings is generally undesirable and impractical. It is well known, however, that such a TTP can be simulated by the user and CA alone using fundamental cryptographic techniques known as *general secure function evaluation* [21,35]. While offering rigorously provable security characteristics, such techniques remain highly impractical, particularly for such computationally intensive operations as key generation. Our contribution in this paper is a technique that simulates the TTP efficiently in a practical sense. The one drawback to our proposal is that it involves a slight weakening of the ideal process: The user is able to influence the TTP to a small (but negligible) degree. We believe that our proposal is of great practical interest, and note that it can even be achieved in a non-interactive setting. We focus on KEGVER protocols for RSA in this paper.

A capsule description of our KEGVER protocol for RSA is as follows. The user and CA jointly select random integers x and y; these integers are known to the user, but not the CA. The user then produces an RSA modulus n. She proves to the CA that n is a Blum integer and the product of two primes, p and q. She furthermore proves that p and q lie in intervals $[x, x + l]$ and $[y, y + l]$ for some public parameter l, i.e., that they are "close" to x and y. The parameter l is selected to be small enough to constrain the user in her construction of the modulus n, but large enough to ensure that she can very likely find primes in the desired intervals. Secure, joint generation of x and y, judicious selection of l, and a number of implementation details form the crux of our work in this paper. As an additional contribution, we propose new definitions required to characterize the security of a KEGVER protocol.

1.1 Previous Work

While general secure function evaluation and zero-knowledge proof techniques are largely impractical, researchers have devised a number of efficient protocols to prove specific properties of public keys. One of the earliest such protocols is due to van de Graaf and Peralta [33], who present a practical scheme for

proving in zero knowledge that an integer n is of the form $p^r q^s$ for primes p and q such that $p, q \equiv 3 \bmod 4$ and the integers r and s are odd. Boyar *et al.* [7] show how to prove that an integer n is square-free, i.e., is not divisible by the square of any prime factor. Together, these two proof protocols demonstrate that an integer n is a Blum integer, i.e., an RSA modulus that $n = pq$ such that $p, q \equiv 3 \bmod 4$. Gennaro *et al.* [19] build on these two protocols to demonstrate a proof system showing that a number n is the product of quasi-safe primes, i.e., that $n = pq$ such that $(p-1)/2$ and $(q-1)/2$ are prime powers (with some additional, technical properties). Camenisch and Michels [8] extend these proof techniques still further, demonstrating a protocol for proving that an RSA integer is the product of two safe primes, i.e., primes p and q such that $(p-1)/2$ and $(q-1)/2$ are themselves primes. While asymptotically efficient, however, this last protocol is not very practical.

Chan *et al.* [11][1] and Mao [25] provide protocols for showing that an RSA modulus n consists of the product of two primes p and q of large size. Liskov and Silverman [23] describe a protocol interesting for its direct use of number-theoretic properties of n to show that p and q are of nearly equal length. Fujisaki and Okamoto [15,16] present related protocols for proving in statistical zero knowledge that a committed integer lies within a given range. All of these protocols are largely superseded for practical purposes by the work of Boudot [6], who, under the Strong RSA Assumption, demonstrates highly efficient, statistical zero-knowledge protocols for proving that a committed number lies in a given range. The Boudot protocols permit proofs of very precise statements about the sizes of p and q.

Loosely stated, all of these protocols demonstrate that a committed number (or public key) lies in a particular set or language. Our aim, which may be viewed as complementary, is to show that a committed number has been selected from a given set *at random* according to some publicly specified process. Thus, these previous protocols, and particularly the Boudot protocols, are useful in the construction of our KEGVER scheme. Our focus in this paper, however, is on the additional apparatus required to make broader statements about adherence to a prescribed key-generation protocol.

A simple approach to ensuring freshness in RSA key generation is for the CA to select a random string s of, say, 100 bits, and require that the leading bits of the public key PK be equal to s. This method, however, has several drawbacks. It only ensures freshness in a narrow sense: While the CA can be assured with high probability that the user has not registered PK before, there is no assurance that the user has not re-used one of the constituent primes of the modulus before. Moreover, by constraining the form of PK, the CA naturally constrains the possible set of private keys SK, leading to some degradation in security. Finally, the required alteration to the key generation process limits compatibility with current prime generation techniques.

[1] The original version of this paper contained a technical flaw, and was subsequently republished as a GTE technical report.

A broader but still simple approach proposed for verification of the key generation process is to derive all underlying randomness from application of a pseudo-random number generator to a random initial seed s. Of course, this approach only provides *ex post facto* arbitration of potential disputes, as revelation of s also discloses the private key.

Distributed key generation is an area closer in spirit to our work in this paper, and can in fact serve directly to achieve a KEGVER scheme for RSA. (For discrete-log based systems, the idea is straightforward and very practical, but we do not have space enough to describe it here.) The best basis is a distributed key generation protocol presented by Boneh and Franklin [5] and further explored most recently in, e.g., [10,20,24]. In this protocol, a minimum of three players (or two, in some variants) jointly generate an RSA modulus n. At the end of this protocol, each of the players holds a share of the corresponding private key. No player learns the whole private key at any point.

To see how such a distributed key generation protocol for RSA can serve as the basis for a KEGVER protocol, consider the two-party case. The idea here is to have the CA act as one player and the user as the other. At the end of the protocol, the CA sends its private key share to the user, who is able then to reconstruct and verify the correctness of the entire private key. This approach enables the CA to be assured that n is generated according to a prescribed protocol, e.g., that p and q are generated uniformly at random from a prescribed range. Likewise, the user in this case can be assured that her private key is not exposed to the CA or to an eavesdropper. The idea for the three-party (or multi-party) case is analogous.

The main drawback to distributed key generation for RSA is that it is quite slow. Malkin *et al.* [24] present experiments involving a highly optimized version of the three-party Boneh and Franklin protocol [5]. These experiments suggest that about 6 minutes of work is required to generate a 1024-bit modulus across the Internet using fast servers. In contrast, convention generation of a 1024-bit RSA on a fast workstation requires less than a second [34]. The basic Boneh and Franklin protocol, moreover, is not secure against active adversaries, and thus would not be suitable by itself as the basis for a KEGVER protocol. Instead, it would be necessary to employ a variant with robustness against malicious players, e.g., [24]. These variants are even less efficient than that of Boneh and Franklin. It is possible to construct a non-interactive KEGVER protocol based on distributed RSA key generation by having the user simulate other players (by analogy with our discrete-log-based example above). The overall costs and complexity of such an approach remain high, however.

Since our aim is not sharing, but rather correct generation of a private key, we adopt an approach in this paper rather different in its technical details from distributed key generation. As a result, we are able to present a KEGVER protocol for RSA that is quite efficient and also has a natural, fully non-interactive variant.

1.2 Our Approach

Let us sketch the intuition behind our KEGVER protocol for RSA, expanding on our capsule description in the introduction to the paper. One common technique for generating a component prime of an RSA modulus n is to pick a random starting point r in an appropriate range, and apply a primality test to successive candidate integers greater than r until a (highly probable) prime p is found. This basic approach may be enhanced by means of sieving or other techniques, but is essentially the same in almost all systems in use today. The pivotal idea behind our KEGVER scheme is for the user and CA to generate r jointly in such a way that r has three properties: (1) r is selected uniformly at random from an appropriate interval; (2) The user knows r; and (3) The CA holds a commitment to r, but does not know r itself. The user performs the same process to derive a starting point s for a second component prime.

Ideally, we would then like the user to furnish an RSA modulus n and prove that the constituent primes p and q are the smallest primes larger than r and s. In the absence of any known practical technique to accomplish this, we adopt a slightly weaker approach. We restrict the user to selection of a modulus n that is a Blum integer, i.e., the product of primes p and q such that $p, q \equiv 3 \bmod 4$. We use well known protocols to have the user prove in zero knowledge that n is indeed a Blum integer. We then employ techniques for proofs involving committed integers, and for range proofs in particular. These enable the user to prove in zero knowledge that p is "close" to r and that q is "close" to s.

As a result of this last proof and the fact that r and s are generated jointly, the user is greatly restricted in her choice of p and q. In particular, she must choose each of these primes from a small interval generated uniformly at random. The result is that the user has very little flexibility in her choice of n, and must therefore select a modulus n nearly as strong as if she had adhered honestly to the prescribed key generation protocol. As a tradeoff against the high efficiency of our protocol, a malicious user does in fact have a little "wiggle room" in her choice of n, but this is small for practical purposes. At the same time, use of zero-knowledge (and statistical zero-knowledge) protocols ensures that the CA gains no information about the private key other than that contained in n itself.

Although we do not dilate on the idea in our paper, we note also that KEGVER can also be employed by a user as a local check against "kleptographic" attacks by an RSA key-generation module [36]. For this, the user employs a separate KEGVER module (generated by a separate entity) to check the correct behavior of the key-generation module.

Organization

In section 2, we present formal definitions for the notion of key generation with verifiable randomness, along with brief description of the cryptographic and conceptual building blocks. We present protocol details in section 3. We offer security and performance analyses in sections 4 and 5 respectively. Due to space limitations, we have omitted many details from this version of the paper. A full version is available from the authors on request.

2 Definitions

Thusfar we have described a KEGVER protocol as one in which the user, through joint computation with a CA, is constrained to produce keys in a manner "close" to honest adherence to some standard key generation algorithm. Our first task is to characterize formally this notion of "closeness". We assume for the sake of simplicity a cryptosystem in which every public key PK has a unique corresponding private key SK. We refer to a probability d as *overwhelming* in parameter l if for any polynomial *poly* there is some L such that $d > 1 - \frac{1}{|poly(l)|}$ for $l > L$. We let $\in_U$ denote uniform random selection from a set.

We begin by defining key generation and key generation with verifiable randomness. We let keygen denote a key generation algorithm that takes as input a soundness parameter t and a key-size parameter k. With probability overwhelming in t, the algorithm outputs a well-formed private/public key pair (SK, PK). The length of the public key is specified by k: For example, in our RSA-based key generation algorithm, it is convenient to let the output key length be $2k-1$ or $2k$ bits. Let $\mathcal{PK}_k$ denote the set of public keys specified by key-size parameter k, i.e., the set of all such possible outputs PK of keygen. We assume that membership in $\mathcal{PK}_k$ is efficiently computable without knowledge of SK. We let $P_{k,t}$ denote the probability distribution induced by keygen over $\mathcal{PK}_k$ for parameter k, and let $P_{k,t}(PK)$ be the probability associated with key PK in particular.

A KEGVER protocol involves the participation of a user and a CA. The protocol takes as input a key-size parameter k and security parameters l, m, and t. Here, l and t are soundness parameters, while m is a security parameter governing statistical hiding of committed values, as we explain below. If the protocol is successful, the public output of the protocol is a public key $PK \in \mathcal{PK}_k$, and the private output to the user is a corresponding private key SK. Otherwise, the protocol fails, and we represent the public output by $\emptyset$. The probability of protocol failure when the participants are honest is characterized by security parameter l. We let $Q_{k;l,m,t}$ denote the probability distribution induced by output PK over $\mathcal{PK}_k$ by the KEGVER protocol when the two participants are honest. We say that the CA *accepts* if the CA is persuaded that PK has been properly generated; otherwise the CA *rejects* the protocol output.

Definition 1. *Let $Q^A_{k;l,m,t}$ be probability distribution induced by the output of* KEGVER *with fixed key-size parameter k and security parameters l, m and t over executions accepted by an honest CA when the user is represented by an algorithm A (not necessarily honest). We say that* KEGVER *is a* μ-sound KEGVER protocol *for* keygen *if, for any algorithm A with running time polynomial in k, we have*

$$\max_{PK \in \mathcal{P}_k} \frac{Q^A_{k;l,m,t}(PK)}{P_{k,t}(PK)} \leq \mu. \tag{1}$$

□

This definition specifies the soundness of KEGVER, stating that a dishonest user can generate a given key with probability only μ times that of an honest user executing keygen. For small μ, this means that it is infeasible for a dishonest

user to persuade the CA to accept the output of the protocol unless its output distribution is similar to that of keygen. Suppose, for example, that keygen is a standard RSA key-generation algorithm for which the RSA assumption [27] is believed to hold. Then if μ is polynomial in k, it is hard for an attacker to weaken her own key effectively in KEGVER.[2] We make the following observation; all quantities here are relative to key-size k, while security parameters are fixed.

Observation 1. *Suppose there exist polynomial-time algorithms A and B such that with non-negligible probability, $B(PK) = SK$ for pairs (SK, PK) distributed according to $Q^A_{k;l,m,t}$. Then if μ is polynomial, it follows that there is a polynomial-time B' such that $B'(PK) = SK$ with non-negligible probability over $P_{k,t}$, and thus that the RSA assumption does not hold on* keygen. □

The other feature we want is for KEGVER is privacy. In particular, we do not want the CA to obtain any (non-negligible) information about SK other than PK. To make this notion more precise, let us consider the following experiment with an adversary A_1. Adversary A_1 engages (not necessarily honestly) in protocol KEGVER with an honest user with parameters k and m. If the protocol is successful, i.e., outputs a public key PK, then A_1 computes and outputs a guess of the corresponding private key SK at the conclusion of the protocol. Let us then consider a second adversary A_2 that is given a public key $PK \in_{Q_{k;l,m,t}} \mathcal{PK}_k$, i.e., a public key drawn from the distribution specified by $Q_{k;l,m,t}$. This adversary likewise computes a guess at the corresponding private key SK, but without the benefit of a transcript from execution of KEGVER.

Definition 2. *We say that* KEGVER *is* private *if for any polynomial-time adversary A_1, there exists a polynomial-time adversary A_2 such that for any polynomial poly there is an L such that $m > L$ implies*

$$\mathsf{pr}[A_1 \text{ guesses } SK] - \mathsf{pr}[A_2 \text{ guesses } SK] < 1/|poly(m)|. \tag{2}$$

□

This definition states informally that by participating in protocol KEGVER yielding public key PK, a CA – or an arbitrary eavesdropper – gains only a non-negligible advantage in its ability to compute the private key SK.

We can extend this definition to consider an adversary A_1 that engages adaptively in some polynomial number of invocations of KEGVER. As we assume independent randomness for each invocation of the protocols in this paper, this extended definition is no stronger for our purposes than Definition 2.

[2] Of course, a malicious user seeking to weaken her own key can tailor an efficient attack algorithm A for factoring n and then promulgate A. For example, A may contain implicit knowledge of one of the component primes of n. The case for repudiation will be difficult to support in such cases, though, as A will be self-indicting.

2.1 Building Blocks

In our scheme, we work over a group $\mathcal{G}$ published by the CA, with order $o(\mathcal{G})$ unknown to the user. We describe $\mathcal{G}$ in the paper as being of "unknown order", as contrasted with a group of "known order", i.e., order known to all players. Additionally, the order $o(\mathcal{G})$ must be larger than the maximum value of the target public RSA key n to be generated by the user. Note that if $o(\mathcal{G})$ is small, this may permit the user to cheat, but will not in fact degrade user privacy. Thus it is in the interest of the CA to choose $\mathcal{G}$ with the appropriate order. In our scheme, it is convenient for the group $\mathcal{G}$ to be generated as a large subgroup of Z_N^* for an RSA modulus N with unpublished factorization. Because of exploitation of the properties of a group of unknown order, many of our sub-protocols rely for security on the Strong RSA Assumption.

The CA additionally publishes two generators of $\mathcal{G}$, denoted by g and h. These generators are selected such that $\log_g h$ and $\log_h g$ are unknown to the user. We believe that the best setup for our protocol is one in which the CA lets $N = PQ$, where $P = 2P' + 1$ and $Q = 2Q' + 1$ for primes P' and Q', and selects $\mathcal{G}$ as the cyclic group of order $2P'Q'$, i.e., the group of elements with Jacobi symbol 1.[3] The CA would then, e.g., select $g, h \in_U \mathcal{G}$. The CA proves to users that $<g>=<h>$. This is accomplished through proofs of knowledge of $\log_g h$ and $\log_h g$, as described below. Since the CA has freedom in selecting N and can therefore manipulate the orders of the groups generated by g and h, however, these proofs of knowledge require t rounds with binary challenges. (This is equivalent to a cut-and-choose proof.) This involves a non-negligible overhead, but the proofs need only be generated by the CA once and checked by each user only upon key registration.

Fujisaki-Okamoto commitment scheme: This commitment scheme, introduced in [15], is essentially a variant on the commitment scheme of Pedersen [28], but adjusted for application to groups $\mathcal{G}$ of unknown order of the form described above. The Fujisaki-Okamoto scheme is statistically hiding in a security parameter m. To commit to a value $x \in Z$, the user selects a commitment factor $w \in_U \{-2^m N + 1, 2^m N - 1\}$. She then computes the commitment $C(x, w) = g^x h^w \bmod N$. For further details, see [6,15]. Note that this commitment scheme is only certain to be hiding provided that the CA has selected g and h such that $<g>=<h>$. The Fujisaki-Okamoto commitment scheme is binding assuming the hardness of factoring, i.e., that the user cannot factor N.

Proof of knowledge of discrete log: Suppose that for some $a \in <g>$, a prover wishes to prove knowledge of $x \in Z_N$ such that $y = g^x \bmod N$. The prover may use a variant of the Schnorr proof of knowledge [31] as follows, with soundness parameter t and privacy parameter m. The prover selects $z \in_U [1, 2^m N]$ and computes $w = g^z$. The verifier computes a challenge $c \in_U [0, 2^t - 1]$. The prover

[3] One must select $\mathcal{G}$ carefully. For example, while certain papers, e.g., [6], state that $\mathcal{G}$ can be any large subgroup of Z_N^*, Mao and Lim [26] provide some caveats on such subgroups with prime order.

returns $r = cx + z$ (over Z). The verifier checks that elements of the prover's proof are in $<g>$. (In our setting, where N is a safe prime and $<g>$ is the cyclic group of order $2P'Q'$, as described above, the verifier checks that elements have Jacobi symbol 1). Then the verifier checks the equality $g^r = wy^c \bmod N$. The protocol is statistical zero-knowledge provided that $1/2^m$ is negligible. It is sound under the Strong RSA Assumption [15]; without breaking this assumption, a cheating prover is able to succeed with probability at most 2^{-t+1} [8]. This proof of knowledge may be rendered non-interactive if the challenge is generated as $c = H(N \parallel g \parallel y \parallel w)$ for an appropriate hash function H. Security may then be demonsrated upon invocation of the random oracle model on H. We assume use of non-interactive proofs in our protocols, and write $POK\{x : y = g^x\}$ to denote a proof of knowledge of the form described here.

Generalized proofs of knowledge of discrete log: As shown in [13,14], it is possible to construct efficient, general, monotone boolean predicates on statements of knowledge of discrete logs. Efficient proofs across multiple bases are also possible. In [8], it is observed that these general proof techniques may be applied to the setting we describe here involving groups of unknown order. We employ here the notation developed by Camenisch and Stadler [9], in which a proof statement is written in the form $POK\{variables : predicate\}$, where *predicate* is a monotone boolean formula on statements of knowledge of discrete logs, potentially over multiple bases. For example, a proof of equality of two values represented by commitments C_1 and C_2 would be written as follows: $POK\{a, r_1, r_2 : (C_1 = g^a h^{r_1}) \bigwedge (C_2 = g^a h^{r_2})\}$. More recently, Damgård and Fujisaki [30] study a generalization of Fujisaki-Okamoto committments and proofs of knowledge for these, making some minor corrections to [15]. A related generalization permits proof that one committed value is equal to the product of two other committed values [8,25].

Interval proof: An *interval proof* is a statistical zero-knowledge proof that a committed value lies within some explicitly specified interval. For commitment $C = g^x h^r$, for example, the prover may wish to prove that $x \in [0, 2^{512}]$. Boudot [6] presents two highly efficient interval proof techniques. We consider here the interval proof in [6] *without tolerance*. The goal is for the prover to demonstrate, for explicit integers a and b such that $b > a$ and on commitment C of value x that $x \in [a, b]$. We represent this by $POK\{x, r : (C = g^x h^r) \bigwedge (x \in [a, b])\}$. Soundness here depends on the Strong RSA Assumption.

Blum integer proof: As noted above, combination of the protocols in [7,33] yields an efficient proof that an integer n is a Blum integer. We denote this proof protocol by $\mathsf{Blum}(n)[t]$, where t is a security parameter. If successful, the protocol yields output 'yes', otherwise output 'no'. The protocol can be either interactive or non-interactive. The soundness of the protocol is overwhelming in t, while the computational and communication costs are linear in t.

3 Protocol

We take as our starting point the following algorithm keygen for RSA key generation. We assume that keygen takes as input an even-valued key-size parameter k (essentially half of the modulus length). We also assume the availability of a probabilistic algorithm primetest(z, t), that takes as input an integer z and soundness parameter t; this algorithm outputs 'yes' if the input element is prime and otherwise, with overwhelming probability in t, outputs 'no'. For technical reasons, our protocol keygen generates RSA moduli n that are Blum integers.

Algorithm keygen$(e, k)[t]$
$r \in_u [2^{k-1}, 2^k - 1]$;
$s \in_u [2^{k-1}, 2^k - 1]$;
while $gcd(e, r-1) > 1$ or $r \not\equiv 3 \bmod 4$
or primetest$[r, t]$ = 'no'
$r \leftarrow r + 1$;
while $gcd(e, s-1) > 1$ or $s \not\equiv 3 \bmod 4$
or primetest$[s, t]$ = 'no'
$s \leftarrow s + 1$;
$p \leftarrow r; q \leftarrow s$;
$d \leftarrow e^{-1} \bmod (p-1)(q-1)$;
output $(n = pq, d)$;

The algorithm keygen outputs with overwhelming probability in k and t a Blum integer (and thus RSA modulus) n with a bit length of $2k-1$ or $2k$. Note that for the sake of efficiency, one would generally use a sieving technique in practice to compute p and q. Adoption of sieving would have no impact, however, on the output of the algorithm. Another common practice is to fix a target bit length for n and adjust the intervals for p and q accordingly. We specify keygen as above for simplicity of presentation.

3.1 KEGVER Protocol

We are now ready to present the details of our KEGVER protocol for RSA key generation. Prior to execution of the protocol, the CA publishes key-size parameter k and security parameters l, m, and t, along with an RSA modulus N such that $|N| > 2k+1$, and whose factorization it keeps private. The CA additionally publishes g and h of a subgroup $\mathcal{G}$ of Z_N^* such that $|o(\mathcal{G})| > 2k+1$, and a proof $Proof_1 = POK\{a, b : (g^a = h) \bigwedge (h^b = g)\}$. As explained above, the ability of the CA to select N means that the soundness of this proof of knowledge depends upon execution with binary challenges over t rounds.

We begin by introducing a sub-protocol unigen. This protocol enables the user and the CA jointly to select a value $z \in [A, B]$ such that if at least one party is honest, z is distributed across $[A, B]$ uniformly at random. As may be seen from the properties of the building blocks, the soundness of the protocol depends on both the Strong RSA Assumption and the discrete log assumption

over $\mathcal{G}$, while privacy is statistical in m. The public output of the protocol is a commitment C_z; the private output, revealed to the user, is z. We let $[A, B]$ denote the input bounds and (n, t) denote the security parameters. We write $(C_z, z) \leftarrow \mathsf{unigen}[A, B](n, t)$ to denote output of public value C_z and private value z from the protocol.

Protocol $\mathsf{unigen}[A, B](m, t)$**.**

1. The user checks the correctness of $Proof_1$, which demonstrates that h and g generate the same group. If $Proof_1$ is incorrect, she aborts.
2. Let $L = B - A + 1$. The user selects $v \in_U [0, L-1]$ and $w_v \in_U [-2^m N, 2^m N]$. She computes $C_v = C(v, w_v)$, and sends C_v to the CA.
3. The CA selects $u \in_U [0, L-1]$ and sends u to the user.
4. The user checks that $u \in [0, L-1]$. If not, she aborts.
5. If $v + u \geq L$, then $o = g^L$; otherwise $o = 0$. The user selects $w_o \in_U [-2^m N, 2^m N]$, computes $C_o = C(o, w_o)$, and sends C_o to the CA.
6. The user executes $Proof_o = POK\{a : h^a = (C_o/g^L) \bigvee (h^a = C_o)\}$. This demonstrates that C_o represents a commitment of g^L or of 0.
7. Let $C_{z'} = C_v g^u / C_o$, a quantity computable by both the user and the CA. The user executes $Proof_{z'} = POK\{a, b : (C_{z'} = g^a h^b) \bigwedge (a \in [0, L-1])\}$. Together, $Proof_o$ and $Proof_{z'}$ demonstrate that $C_{z'}$ represents a commitment of $(u + v) \bmod L$.
8. If the CA is unable to verify either $Proof_o$ or $Proof_{z'}$, then the CA aborts. Otherwise, the public output of the protocol is $C_z = C_{z'} g^A$, and the private output is $z = ((u + v) \bmod L) + A$.

Given unigen as a building block, we are ready to present the full protocol for KEGVER. The basic strategy is for the user and CA to employ unigen to generate r and s, private values from which the user initiates a search for primes p and q. The user then proves, by way of commitments on her private values, that p and q are "close" to r and s respectively, and then that $n = pq$ is a Blum integer. The pair $[e, k]$ is input such that e represents the public exponent and k specifies the bit length of p and q, and thus n. Security parameters are l, m and t. The public output of the protocol is $n = pq$, while the private output is (p, q).

Protocol $\mathsf{KEGVER}[e, k](l, m, t)$**.**

1. $(C_r, r) \leftarrow \mathsf{unigen}[2^{k-1}, 2^k - 1](m, t)$.
2. $(C_s, s) \leftarrow \mathsf{unigen}[2^{k-1}, 2^k - 1](m, t)$. (Note that the expensive verification step 1 in unigen can be omitted here, as it was already executed in the previous invocation.)
3. The user generates a prime $p \geq r$ meeting the conditions: (1) $gcd(e, p-1) = 1$; (2) $p \equiv 3 \bmod 4$; and (3) $p - r$ is minimal. If $p - r > l$, the user aborts, and the protocol output is $\emptyset$.
4. The user generates a prime $q \geq s$ meeting the conditions: (1) $gcd(e, q-1) = 1$; (2) $q \equiv 3 \bmod 4$; and (3) $q - s$ is minimal. If $q - s > l$, the user halts, and the protocol output is $\emptyset$.

5. The user selects $w_p, w_q \in_U [-2^m N, 2^m N]$ and computes $C_p = C(p, w_p)$ and $C_q = C(q, w_q)$.
6. The user sends C_p to the CA and proves $POK\{a, b : (C_p/C_r = g^a h^b) \bigwedge (a \in [0, l])\}$, and analogously for C_q.
7. The user sends $n = pq$ to the CA and proves $POK\{a, b, c, d : (C_p = g^a h^b) \bigwedge (C_q = g^c h^d) \bigwedge (g^n = g^{ac})\}$. In other words, the user proves that C_p and C_q are commitments to factors of n.
8. The user executes $\mathsf{Blum}(n)[t]$.
9. If the CA is unable to verify one or more proofs, or if $\mathsf{Blum}(n)[t]$ outputs 'no', the CA rejects and the protocol output is $\emptyset$. Otherwise, the public output of the protocol is n and the private output, obtained by the user, is (p, q).

Non-interactive variant of KEGVER*:* The protocol KEGVER can be rendered non-interactive by having the user execute all proofs non-interactively and generate the value u in unigen as $H(C_v)$ for an appropriate hash function H. In this case, we really have two algorithms KEGVER_{user} and KEGVER_{CA}, where KEGVER_{user} produces a public key PK and proof transcript T, and KEGVER_{CA} decides whether to accept or not to accept a key/transcript pair (PK', T'). To guard against reuse of stale keys, a CA may require that the hash function H be keyed uniquely to that CA. Of course, this does not prevent intentional subsequent use of stale keys with CAs that do not adopt such a precaution.

Definition 1 must be altered for the non-interactive case. In particular, we define the probability $Q^A_{k;l,m,t}$ so that the probability distribution over keys PK' yielded by polynomial-time attack algorithm A also produces an accompanying transcript T' accepted by the CA. The algorithm A can of course run KEGVER_{user} or some variant algorithm any number of times polynomial in k.

4 Security

If $<g>=<h>$, the protocol KEGVER is statistical zero-knowledge with privacy dependent on the parameter m used for the construction of commitments [6,15]. Details on simulator construction for the CA are available in security proofs for the underlying primitives as presented in the literature. If the proof protocols in KEGVER are to be realized non-interactively (as is better for most practical purposes), then the zero-knowledge property depends additionally on a random oracle assumption on an underlying hash function used for challenge generation [29]. In the case that $<g>\neq<h>$, the commitments of the user may not in fact be statistically secure. Hence, the privacy of KEGVER also depends on the soundness of $Proof_1$. The soundness of all proof protocols depends on the challenge sizes and also, for non-interactive proofs, on the random oracle assumption.

The new and critical security issue we focus on here is the choice of security parameter l and its impact on the soundness bound μ of Definition 1. To address this issue, we require use of a number theoretic conjecture regarding the density of primes. Most relevant here the view of prime density offered by Gallagher [17]. Gallagher shows that number of primes in the interval $(x, x + \lambda \ln x]$ is Poisson distributed with mean λ as $x \to \infty$.

The security of our construction depends on a slightly different quantity. In particular, our aim is to find a value l such that for a random k-bit value r, the interval $[r, r+l]$ with overwhelming probability contains a prime p such that $p \equiv 3 \bmod 4$ and $gcd(e, p-1) = 1$. For this, we make two heuristic assumptions. Our first assumption is that the distribution of primes in the range used to construct RSA moduli is roughly Poisson distributed in accordance with the conjecture of Gallagher. We assume, second, that e is an odd prime constant (as is the case in most applications). Finally, let d_1 denote the probability density of primes p of general form; let d_2 denote the probability density of primes p such that $p \equiv 3 \bmod 4$ and $e \nmid (p-1)$. We assume, as one would naturally expect, that $d_2/d_1 = (e-1)/2e$. Let X be a Poisson-distributed random variable with mean λ. The probability that $X = 0$ is $e^{-\lambda}$. Thus we obtain the following conjecture.

Conjecture 1. For large r, the probability that the interval $[r, r+l]$ contains no prime $p \equiv 3 \bmod 4$ such that $e \nmid (p-1)$ is at most $e^{-\lambda}$ for $l = \lambda \ln r(\frac{2e}{e-1})$ or, equivalently, for $\lambda = \frac{l}{\ln r}(\frac{e-1}{2e})$. □

This conjecture yields the following observation on the best parameterization of λ and l in accordance with Definition 1. It is easy to see that this observation extends to the non-interactive variant of KEGVER.

Observation 2. *Suppose that* $\lambda = \omega(\ln \ln r) = \omega(\ln k)$, $t = \omega(\ln k)$ *and* $\lambda \frac{2e}{e-1} \ln r < l = O(k^c)$ *for some constant c. Then the failure probability for an honest user, i.e., the probability that an honest user cannot find suitable primes p and q in* KEGVER *is negligible in k, and the soundness bound* μ *is polynomial in k.* □

Example 1. Let us consider a concrete example involving the generation of 512-bit primes (i.e., roughly a 1024-bit RSA modulus) and public exponent $e = 3$. Choosing $\lambda = 57$ yields a failure probability for an honest user in KEGVER of less than 2^{-80} by Conjecture 1. This corresponds to $l = \lambda \frac{2e}{e-1} \ln r < 60,687$. Clearly, given that at most one in four integers has the form $p \equiv 3 \bmod 4$, the maximum number of primes p in an interval of this size is at most 15,171. It follows then that our KEGVER protocol is μ-sound for $\mu < 15,171^2 = 230,159,241$. This assumes that the soundness parameter t is large enough so that the ability of an attacker to cheat in any zero-knowledge proof is negligible, e.g., $t = 100$.

Stronger concrete security bounds: The concrete security bounds demonstrated in Example 1 above are deceptively weak. First, we note that μ is a bound on the ability of an attacker to distort the output distribution of KEGVER. For this, the ideal strategy is for a malicious user to choose a prime p in the interval $[r, r+l]$ such that the preceding prime p' is as close as possible to p. In fact, though, the aim of a malicious user is entirely different, namely to generate a key that is weak with respect to some attack algorithm or algorithms. Hence, the attacker is much more tightly constrained than our analysis according to Definition 1 suggests at first glance.

We can achieve substantially stronger concrete security bounds by relaxing Definition 1 in a probabilistic sense across intervals. We do not dilate formally

on the idea here. Instead, we note simply that in Example 1 above involving generation of 512-bit primes with $l = 60,687$, the average number of primes of the form $p \equiv 3 \bmod 4$ in the interval $[r, r + l]$ is about 86, and the distribution of such primes is very tightly concentrated around this mean. In fact, under the Gallagher conjecture, the probability that the interval contains more than 250 such primes is well less than 2^{-80}. Thus, given a sufficiently large soundness parameter t (e.g., $t = 100$), the soundness bound $\mu < 250^2 = 62,500$ is a more accurate one for our purposes in Example 1.

5 Performance

One of the desirable features of KEGVER is that it places the bulk of the computational burden (primarily in the protocol B*lum*) on the user, rather than the CA. This preserves the usual balance of computational effort by the two parties. In particular, RSA key generation, which the user must perform in any case, is a computationally intensive task. In contrast, certification of an RSA key by a CA is, in its basic form, a relatively lightweight operation.

We can substantially reduce the computational requirements for the CA through use of such techniques as batch verification, as introduced in [12], and improved in many subsequent works such as [3], combined with addition chains, as explored in, e.g., [4]. We estimate that such enhancements would yield a speedup for the CA of approximately a factor of six. An additional protocol modification we can exploit is elimination of square-freeness proof protocol of van de Graaf and Peralta [33]: It can be proven that the KEGVER protocol itself implicitly enforces the condition of square-freeness with high probability. Due to space limitations, we omit proof of this fact from this version of the paper.

Given these observations, we can express the computational cost of the protocol very roughly in terms of the number of modular exponentiations required by the two parties (disregarding small added costs, such as the fact that Fujisaki-Okamoto commitments require exponents slightly longer than the modulus). Given soundness parameter $t = 100$, the computational requirement for the user in KEGVER is the equivalent of about 153 full modular exponentiations. The overall computational cost for the CA to the equivalent of roughly 10 full modular exponentiations. The transcript size for the full protocol is about 37kB.

We have created an implementation in C of a non-interactive variant of the KEGVER protocol for 1024-bit RSA modulus generation. Timing experiments took place on a Pentium III processor running Windows NT 4.0, with 64 Mbytes of RAM and running at 500 MHz. We compiled our code under gcc version 2.95.3 through use of the UNIX emulation environment Cygwin version 1.3.2. For multiprecision arithmetic, we used the GNU MP library, version 3.1.1. We note that the GMP library computes exponentiations via the sliding-window method for exponentiation [27] which provides roughly a 20–30% speed-up over the binary method for exponentiation. In addition, we implemented routines for double exponentiation using the method of simultaneous multiple exponentiation attributed to Shamir in [18]. Due to time constraints in the construction of

the prototype, triple exponentiations were implemented simply through one call each to the double exponentiation and the single exponentiation routines, with multiplication of the partial results. In addition, in all of the computations by the CA (verifier), we employ the Chinese Remainder Theorem (CRT).

Table 1. Time Critical Proofs/Protocols in KEGVER

Proof/Protocol	# times called	Prover (sec)	Verifier (msec)
unigen	2	2.7	509
rangeproof (long)	(2)	(2.4)	(438)
rangeproof (short)	2	1.7	343
Blum Proof	1	1.3	201
KEGVER	–	10.9	2.05 sec

Table 1 summarizes the timings of the critical proofs and protocols in KEGVER. We denote the range proofs by the generic label rangeproof; the label "long" indicates a relatively expensive proof over a large interval, and "short", one on a small interval. The second column in Table 1 indicates the number of times that the specified protocol is called by KEGVER (either user or verifier). There are two calls to unigen; these include two range proofs, whose timings are provided in the next row. (Parentheses indicate that the associated calls and timings are subsumed by calls to unigen.) There are also two independent invocations of short range proofs, one for each of the primes in the RSA modulus. These latter proofs correspond to Step 6 in KEGVER. We observe that roughly 86% of the time required for unigen is in fact accounted for by an invocation of the associated (long) range proofs. Together, invocations of the cryptographic protocols unigen,Blum, and rangeproof (short) account for about 92% of the time required to perform KEGVER, the remainder accounted for by non-cryptographic operations. This is true for both the user and CA. For further details on our experiments, we refer the reader to the full version of this paper.

We did not implement batch verification or addition chains in this prototype. Additionally, we employed a range-proof protocol known as SZKrange$^+$ [22] in lieu of the Boudot protocol; the latter is about twice as fast in this setting. (We did this not for technical reasons, but due to intellecutal property concerns regarding the Boudot protocol.) Through use of batch proofs and the Boudot protocol, we believe it possible to achieve roughly a factor of 10 improvement in the performance of the verifier (i.e., CA) protocol. This would reduce the execution time to about 205 msec, making KEGVER highly practical.

Acknowledgments

Thanks to Markus Jakobsson, Burt Kaliski, Ron Rivest, Bob Silverman, and the anonymous referees of this paper for their comments and suggestions.

References

1. IEEE Std. 1363-2000. *Standard Specifications for Public-Key Cryptography.* The Institute of Electrical and Electronics Engineers, 2000.
2. ANSI X9.31 2001. *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (X9.31).* American National Standards Institute (ANSI), 2001.
3. M. Bellare, J.A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98.* Springer-Verlag, 1998. LNCS no. 1403.
4. D. Bleichenbacher. Addition chains for large sets, 1999. Unpublished manuscript.
5. D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In B. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, pages 425–439. Springer-Verlag, 1997. LNCS no. 1294.
6. F. Boudot. Efficient proofs that a committed number lies in an interval. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT '00*, pages 431–444, 2000. LNCS no. 1807.
7. J. Boyar, K. Friedl, and C. Lund. Practical zero-knowledge proofs: Giving hints and using deficiencies. *Journal of Cryptology*, 4(3):185–206, 1991.
8. J. Camenisch and M. Michels. Proving that a number is the product of two safe primes. In J. Stern, editor, *Advances in Cryptology –EUROCRYPT '99*, pages 107–122. Springer-Verlag, 1999. LNCS no. 1592.
9. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, pages 410–424. Springer-Verlag, 1997. LNCS no. 1294.
10. D. Catalano, R. Gennaro, and S. Halevi. Computing inverses over a shared secret modulus. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT '00*, pages 445–452. Springer-Verlag, 2000. LNCS no. 1807.
11. A. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. In K. Nyberg, editor, *Advances in Cryptology –EUROCRYPT '98*, pages 561–575. Springer-Verlag, 1998. LNCS no. 1403. Revised version available as GTE tech. report.
12. L. Chen, I. Damgård, and T.P. Pedersen. Parallel divertibility of proofs of knowledge (extended abstract). In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, pages 140–155. Springer-Verlag, 1994. LNCS no. 950.
13. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y.G. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, pages 174–187. Springer-Verlag, 1994. LNCS no. 839.
14. A. de Santis, G. di Crescenzo, G. Persiano, and M. Yung. On monotone formula closure of SZK. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 454–465. IEEE Press, 1994.
15. E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In B. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, pages 16–30. Springer-Verlag, 1997. LNCS no. 1294.
16. E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '98*, pages 32–46. Springer-Verlag, 1998.
17. P.X. Gallagher. On the distribution of primes in short intervals. *Mathematika*, 23:4–9, 1976.

18. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.
19. R. Gennaro, D. Micciancio, and T. Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 67–72, 1998.
20. N. Gilboa. Two party RSA key generation. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, pages 116–129. Springer-Verlag, 1999. LNCS no. 1666.
21. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87*, pages 218–229. ACM Press, 1987.
22. A. Juels. $SZKrange^{+}$: Efficient and accurate range proofs. Technical report, RSA Laboratories, 1999.
23. M. Liskov and B. Silverman. A statistical-limited knowledge proof for secure RSA keys, 1998. Manuscript.
24. M. Malkin, T. Wu, and D. Boneh. Experimenting with shared generation of RSA keys. In *1999 Symposium on Network and Distributed System Security (SNDSS)*, pages 43–56, 1999.
25. W. Mao. Verifiable partial sharing of integer factors. In *Selected Areas in Cryptography (SAC '98)*. Springer-Verlag, 1998. LNCS no. 1556.
26. W. Mao and C.H. Lim. Cryptanalysis in prime order subgroups of Z_n^*. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98*, pages 214–226. Springer-Verlag, 1998. LNCS no. 1514.
27. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
28. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, pages 129–140. Springer-Verlag, 1991. LNCS no. 576.
29. D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, pages 287–398. Springer-Verlag, 1996. LNCS 1070.
30. I. Damgård and E. Fujisaki. An integer commitment scheme based on groups with hidden order, 2001. IACR eArchive.
31. C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.
32. A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
33. J. van de Graaf and R. Peralta. A simple and secure way to show the validity of your public key. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, pages 128–134. Springer-Verlag, 1987. LNCS no. 293.
34. M. Wiener. Performance comparison of public-key cryptosystems. *Cryptobytes*, 4(1), 1998.
35. A.C. Yao. Protocols for secure computations (extended abstract). In *FOCS '82*, pages 160–164, 1982.
36. A. Young and M. Yung. Kleptography: Using cryptography against cryptography. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, pages 62–74. Springer-Verlag, 1997. LNCS no. 1233.