Lower Bounds for Factoring Integral-Generically, with Room for Improvement

Daniel R. L. Brown[*]

January 29, 2010

Abstract

An integral-generic factoring algorithm is, loosely speaking, a constant sequence of ring operations that computes an integer whose greatest common divisor with a given integral random variable n, such as an RSA public key, is non-trivial. Formal definitions for generic factoring will be stated. Integral-generic factoring algorithms seem to include versions of trial division and Lenstra's elliptic curve method. Abstract lower bounds on the number of such ring operations will be given. Concrete lower bounds on the abstract bounds are also given, but prove to be too weak for any cryptologic assurance.

Key Words: Factoring, Straight Line Programs.

1 Introduction

An integral-generic factoring algorithm is, loosely speaking, a constant sequence of ring operations that computes an integer whose greatest common divisor with a given integral random variable n, such as an RSA public key, is non-trivial. Formal definitions for generic factoring will be stated. Integral-generic factoring algorithms seem to include versions of trial division and Lenstra's elliptic curve method. Abstract lower bounds on the number of such ring operations will be given. Concrete lower bounds on the abstract bounds are also given, but prove to be too weak for any cryptologic assurance.

1.1 Previous Work

Shoup [Sho97] introduced the generic group model and proved lower bound the difficulty of solving the discrete logarithm problem in this model. Damgård and Koprowski [DK09] extended Shoup's model to groups with unknown order, and imply a lower bound of the difficulty of group order in this model. Their lower bound is larger than the cost of existing factoring algorithms, but such algorithms are not restricted to group operations only. Leander and Rupp [LR06] introduced the generic ring model, and Aggarwal and Maurer [AM09] proved that the RSA problem is as difficult as the factoring problem, in this model.

Boneh and Venkatesan [BV98] essentially proved that a generic factoring algorithm with an RSA oracle could be converted into one without an RSA oracle.

[*]Certicom Research

Upper bounds on the difficulty of generic factoring are provided by examples of generic factoring algorithms. The trial division factoring algorithm has a version which is integral-generic. Lenstra's [Len87] elliptic curve method has version that is an integral-generic factoring algorithm, and which may have similar running time.

Shamir [Sha79] provides an asymptotic upper bound of $O(\log n)$ arithmetic steps to factor n. This very low upper bound proves that factoring can only have very low lower bound in terms of number of arithmetic steps. However, Shamir's arithmetic steps include integer division, which is the floor of rational division, and these steps are excluded from the generic ring model: Shamir's algorithm is not a generic factoring algorithm and a lower bound on the number of steps in a generic factoring algorithm could exceed Shamir's upper bound on arithmetic steps.[1] Furthermore, the steps in Shamir's algorithm involve arbitrarily large integers, whereas in the generic factoring, the steps measured are operations on inputs of fixed bit size. Upon conversion to bit operations, Shamir's bound becomes quite large.

Lipton [Lip94] provides effectively yet another kind of upper bound: showing the factoring is no harder than computing, with a straight line program, polynomials with many distinct rational roots.

Cheng [Che04] recasts Lenstra's elliptic curve method factoring as an upper bound on the straight line complexity of certain integers (which we shall call the length of integers). Our lower bounds may be expressed in terms of the length of integers, we study lower bounds for these. Cheng [Che04] later speculates about lower bounds on the lengths of integers and the possibility of certain factoring algorithms, but does not give an explicit lower bound on factoring.

Moreira [Mor97] provides lower bounds on the length of almost all integers. Lower bounds of the lengths of integers are exactly what is needed in this paper, but because Moreira's bound only applies to almost all integers, it is not directly applicable, unless some heuristic assumptions are made. An adversary is not bound to choose a compute integer, but may seek one that can be computed more efficiently than average.

Shub and Smale [SS96] show that the short length of certain (sets of) integers is equivalent to an algebraic version of $NP \neq P$. Assuming the hardness of the latter also provides a conditional lower bound on the lengths of certain integers. Perhaps this conditional lower bound can be extended to become a lower bound on integral-generic factoring algorithms.

Arguably, the lower bounds in this paper are implicit in any of the papers above related to factoring. In this case, this paper merely serves to state explicitly what has been implied before. In so doing, this paper encourage further research in this direction.

1.2 Organization of the paper

Section 2 provides definitions for generic factoring algorithms and other notions needed. Section 3 gives lower on the lengths of integral-generic factoring algorithms, which seem to be new.

Appendix B suggests how to possibly obtain bounds on lengths of more general generic factoring algorithm. Appendix A reviews some upper bounds on the lengths of integral-generic factoring algorithm, based on real world examples. Appendix C considers philosophical questions about the lack of any good lower bounds for factoring. Appendix D gives example of lengths of integers.

[1]The extra operations in Shamir's algorithm, integer division, can be used to implement the Euclidean algorithm for finding modular inverses, which seems to be hard to do with mere ring operations, unless factoring is easy.

2 Definitions

This section defines integral-generic factoring algorithms and the length of integers, which we use to lower bound the length of such algorithms. The definitions actually include more general kinds of factoring algorithms, for the purpose of completeness and as suggestion for future research.

2.1 Rings

Rings in this paper shall be unital, that is, they shall have a multiplicative identity indicated as 1. Rings in this paper shall be efficient, that is, efficient algorithms exist to implement addition, subtraction and multiplication. The main rings of interest in this paper are the ring $\mathbb{Z}$ of integers and the ring $\mathbb{Z}/(n)$ of[2] integers modulo n.

When clear from context open intervals are indicated (a, b), closed by $[a, b]$, semi-open by $(a, b]$ and $[a, b)$. We mainly apply these interval notations to describe subsets of $\mathbb{Z}$.

For the sake of generality, we also consider pseudodivision operations. A pseudodivision ring R is a ring with an extra operation defined: pseudodivision. For inputs $r, s \in R$, the pseudodivision algorithm gives an output $t = r/s$. The pseudodivision is termed successful if $r = st$. We may assume an artificial convention that in a pseudodivision ring, a pseudodivision operation to compute $r/s = 0$ whenever it is not successful. The actual probability of successful pseudodivision will not be defined here, because this probability will not be needed. For $\mathbb{Z}$ the success rate will be very low, but for $\mathbb{Z}/n$ it can be quite high. Again, we assume that the pseudodivision operation is efficient.

Given any (pseudodivision) ring R, we will want to consider a random predicate $\pi_R : R \to \{0, 1\}$. This may regarded as random variable drawn uniformly from the set Π_R of all such functions, that is, $\Pi_R = \{0, 1\}^R$.

2.2 Generic Ring Programs

A *generic ring program* P is a finite sequence consisting L steps $\sigma_1, \ldots, \sigma_L$. Three types of steps are permitted:

1. Operational: a step which is a triple $\sigma_k = (\circ_k, i_k, j_k)$ where $\circ_k \in \{+, -, \times, /\}$ and $i_k, j_k \in \mathbb{Z} \cap (-\infty, k)$.
2. Relational: a step which is a quintuple $\sigma_k = (\circ_k, i_k, j_k, l_k, m_k)$ where $\circ_k \in \{=\}$ and $i_k, j_k, l_k, m_k \in \mathbb{Z} \cap (-\infty, k)$.
3. Representational: a step which is quadruple $\sigma_k = (\circ_k, i_k, j_k, l_k)$ where $\circ_k \in \{\$\}$ and $i_j, j_k, l_k \in \mathbb{Z} \cap (-\infty, k)$.

The *length* of P is L. Let I_P be the set of negatives of the negative integers among the integers i_k, j_k, l_k, m_k. The cardinality of I_P is the *width* w of P. We will always assume that I_P consists of the w smallest positive integers, that is, $I_P = \mathbb{Z} \cap [0, w] \cap (0, \infty)$.

If $\circ_k \neq \$$ for all k, then P is said to be *deterministic*. If $\circ_k \notin \{\$, =\}$ for all k, then P is said to be *straight line*, and the terms *generic ring* is omitted (giving *straight line program*). If $\circ_k \neq /$ for all k, then P is said to be *divisionless*. If a generic ring program P is divisionless, straight line and has width 0, then we say that P is an *integral-generic program*.

[2]Here, (n) indicates the ideal generated by n.

Given a pseudodivision ring R, and a program P above, and an integer $m \leqslant L$, we define a function $P_m : R^w \times \Pi_R \to R^m$, such that,

$$P_m(r_1, \ldots, r_w; \pi_R) = (s_1, \ldots, s_m) \tag{1}$$

evaluated as follows.

1. For each positive integer $k \leqslant w$, if any, let $x_{-k} = r_k$.
2. Let $x_0 = 1$.
3. For integers $1 \leqslant k \leqslant L$, if step σ_k is
   (a) operational, let $x_k = x_{i_k} \circ_k x_{j_k}$,
   (b) relational, let $x_k = x_{i_k}$ if $x_{l_k} = x_{m_k}$ and let $x_k = x_{j_k}$ otherwise.
   (c) representational, let $x_k = x_{i_k}$ if $\pi(x_{l_k}) = 1$ and let $x_k = x_{j_k}$ otherwise.
4. For $1 \leqslant l \leqslant m$ let $s_l = x_{L-m+l}$.

Note that more generally, we can designate any m steps to be output steps, rather than just the last m steps. This can be useful when an earlier step yields a desired output. Algorithms employing the program P can then terminate the execution of P earlier. Note that an integral-generic program simply computes some fixed integer(s), or more precisely, their image in the ring R under the natural homomorphism $\mathbb{Z} \to R$ (which exists because R has a multiplicative identity). Unless otherwise stated, we assume that an integral-generic factoring algorithm has breadth 1.

2.3 Generic Factoring Algorithms

Let n be an integer-valued random variable that one wishes to factor. Let P be a generic ring program of width w and length L. Consider the following probability space:

1. The random variable n.
2. The random variable $(r_1, \ldots, r_w)$ uniformly distributed in the set R^w, where $R = \mathbb{Z}/n$.
3. The random variable π_R.

Evaluate $P_m(r_1, \ldots, r_w) = (s_1, \ldots, s_m)$. Let $d_i = \gcd(n, s_i)$. If $\{d_1, \ldots, d_m\} \not\subseteq \{1, n\}$, then we say that P_m has found a factor n.

Suppose P_m finds a factor of n with probability at least μ over the probability space defined above. Then P_m may be called a *generic factoring* algorithm for n with width w, length L, breadth m and success probability μ. More precisely, the algorithm is given the value of random variable n, and then

1. Chooses random inputs $(r_1, \ldots, r_w) \in R^w$ uniformly.
2. Chooses random function π_R.
3. Computes $(s_1, \ldots, s_m) = P_m(r_1, \ldots, r_w; \pi_R)$.

4. Computes all $d_i = \gcd(n, s_i)$.
5. Returns $\{d_1, \dots, d_s\} \setminus \{1, n\}$.

Under the assumptions above, then with probability μ, we the output of this algorithm is non-empty. All its members, if any, will be a proper factor of n. A further generalization of generic factoring algorithms allows non-uniform distributions of the inputs r_i, and perhaps even the predicate function π_R, though such factoring algorithms will not be considered in this paper.

Qualifiers to generic ring programs such as deterministic, straight line, divisionless, and integral may also be applied to generic factoring algorithms. This paper will focus on integral-generic factoring algorithms.

2.4 Length of an integer

The length $\lambda(n)$ of the shortest zero-width divisionless straight line program that computes a given integer n is the *length* of the integer n. Shub and Smale [SS96] and Moreira [Mor97] write $\tau(n)$ for $\lambda(n)$, Moreira calls it the *cost* of the integer, while Cheng [Che04] calls it *complexity*.

Some weak bound for the length of a positive integer n are

$$1 + \log_2(\log_2(n)) \leqslant \lambda(n) \leqslant 2\log_2(n) \tag{2}$$

The lower bound is strict except at $n = 2^{2^m}$. The lower bound follows by comparison to the straight line whose first step a double, and all other steps squarings of the previous value. The upper bound is obtained from double-and-add algorithm.

Moreira proves that, for almost all n, the lower bound

$$\lambda(n) \geqslant \frac{\log n}{\log\log n} \tag{3}$$

and a slightly upper bound that holds for all sufficiently large n. Note that here log is to the natural base.

3 Lower Bounds for Integral-Generic Factoring

A breadth-1 integral-generic factoring algorithm A computes the homomorphic image of some integer I, via integral-generic program P, in the ring $\mathbb{Z}/(n)$ where n is the integer to factored. It follows that the length L of this factoring algorithm is at least $L \geqslant \lambda(I)$.

3.1 An Abstract Lower Bound

Assuming that A has success rate μ for random variable n, the probability that $\gcd(n, I) \notin \{1, n\}$ is at least μ. Let $I(n, \mu)$ be the set of integers I with this property. Therefore, we have a lower bound on the length L of an integral-generic factoring algorithm:

$$L \geqslant \Lambda(n, \mu) = \min_{I \in I(n,\mu)} \lambda(I) \tag{4}$$

Of course, with no concrete knowledge about $\Lambda(n, \mu)$, this remains merely an abstract lower bound.

3.2 A Weak Concrete Lower Bound

In this section, for special cases of n and μ of potential interest, we find a concrete lower bound below the abstract lower bound $\Lambda(n,\mu)$. The concrete lower bound, though, does not provide much assurance.

Suppose that $\mu = 1/2$. Suppose that the random variable n is generated as $n = pq$ where p and q are identically distributed independent random variables uniformly distributed in the set S of primes integers in the interval $(2^s, 2^{s+1})$. For the sake of simplicity, suppose that the number of primes in S is even. (If not, just remove one prime.) For specificity, we denote this random variable by n_s. This n_s is somewhat similar to the distribution of RSA public key moduli.

The set $I(n_s, 1/2)$ is the set of integers which are divisible by exactly half of the primes in S. Using the lower bound $\lambda(n) \geqslant 1 + \log_2 \log_2(n)$, we get a lower bound

$$\Lambda(n_s, 1/4) \geqslant 1 + \log_2 \log_2 \frac{(2^s + |S|/2)!}{2^s!}, \tag{5}$$

by noting that for $I \in I(n_s, 1/2)$, we must have I at least the product of the first $|S|/2$ numbers larger than 2^s.

The right-hand side of (5) is not entirely concrete, because $|S|$ may be unknown. Rigorous concrete lower bounds can be provided, but in this paper, mere approximations will be given, but the rigorous lower bounds should be quite similar. From the prime number theorem, we derive an approximation $|S|/2 \approx \frac{2^{s+1}}{4\log(2^{s+1})} = \frac{2^{s-1}}{(s+1)\log(2)}$. Let $2^s + |S|/2 = 2^s(1+\epsilon)$ where $\epsilon \approx 1/(2(s+1)\log(2))$. Taking Stirling's approximation, that $\log(n!) = n\log(n) - n + \frac{1}{2}\log(n) + \frac{1}{2}\log(\pi)$, we have an approximation:

$$\log \frac{(2^s(1+\epsilon))!}{2^s!} \approx 2^s \epsilon \log 2^s - \epsilon 2^s + 2^s(1+\epsilon)\log(1+\epsilon)$$

Using the fact that ϵ is small, we get an approximation $\log(1+\epsilon) \approx \epsilon$. Throwing away dominated terms, gives a crude approximation:

$$\log \frac{(2^s(1+\epsilon))!}{2^s!} \approx 2^{s-1}$$

Therefore, we get a lower bound for L of approximately s.

3.3 Failure of Applying Moreira's Lower Bound

A heuristic one could try to apply is to assume that Moreira's bound holds for all $I \in I(n,\mu)$, since it holds for almost all integers. This heuristic would lead to the following bound:

$$\Lambda(n_s, 1/2) \geqslant \frac{\log}{\log\log}\left(\frac{(2^s + |S|/2)!}{2^s!}\right), \tag{6}$$

which, under the approximations above, amounts to something like $\Lambda(n_s, 1/2) \geqslant 2^{s-1}/s$. This lower bound seems too high, because integral-generic versions of Lenstra's elliptic curve method of factoring seem to have lower lengths than this heuristic lower bound. Therefore assuming the Moreira lower bound over all of $I(n,\mu)$ seems like an incorrect heuristic.

4 Conclusion

A linear lower bound on the length of integral-generic factoring algorithms was given. A linear lower bound can be given for the number of bit operations for the addition and multiplication operations of the ring corresponding to the number to be factored. This results in a quadratic lower bound for the number of bit operations for an integral-generic factoring algorithm. This lower bound is no better than the cost of RSA decryption operation. So, there is room for improvement. It is natural to hope that the lower bounds can be improved, and such would be worthy exercise even if one only ever obtains a polynomial lower bound.

References

[AM09] D. Aggarwal and U. Maurer. *Breaking RSA generically is equivalent to factoring.* In A. Joux (ed.), *Advances in Cryptology — EUROCRYPT 2009*, no. 5479 in LNCS, pp. 36–53. IACR, Springer, Apr. 2009.

[BV98] D. Boneh and R. Venkatesan. *Breaking RSA may be easier than factoring.* In K. Nyberg (ed.), *Advances in Cryptology — EUROCRYPT '98*, no. 1403 in LNCS, pp. 59–71. IACR, Springer, May 1998. http://crypto.stanford.edu/~dabo/abstracts/no_rsa_red.html.

[Che04] Q. Cheng. *On the ultimate complexity of factorials.* Theoretical Computer Science, **326**(1–3):419–429, Oct. 2004.

[DK09] I. Damgård and M. Koprowski. *Generic lower bounds for root extraction and signature schemes in general groups.* In L. Knudsen (ed.), *Advances in Cryptology — EUROCRYPT 2002*, no. 2332 in LNCS, pp. 256–271. IACR, Springer, Apr. 2009.

[Len87] H. W. Lenstra. *Factoring integers with elliptic curves.* Annals of Mathematics, **126**:649–673, 1987.

[Lip94] R. J. Lipton. *Straight line complexity and integer factorization.* In L. M. Adleman and M.-D. Huang (eds.), *Algorithmic Number Theory*, no. 877 in LNCS, pp. 71–79. Springer, may 1994.

[LR06] G. Leander and A. Rupp. *On the equivalence of RSA and factoring regarding generic ring algorithms.* In X. Lai and K. Chen (eds.), *Advances in Cryptology — ASIACRYPT 2006*, no. 4284 in LNCS, pp. 241–251. IACR, Springer, Dec. 2006.

[Mor97] C. G. T. D. A. Moreira. *On asymptotic estimates for arithmetic cost functions.* Proceedings of the American Mathematical Society, **125**(2):347–353, Feb. 1997.

[Sha79] A. Shamir. *Factoring numbers in $O(\log n)$ arithmetic steps.* Information Processing Letters, **8**(1):28–31, 1979.

[Sho97] V. Shoup. *Lower bounds for discrete logarithms and related problems.* In W. Fumy (ed.), *Advances in Cryptology — EUROCRYPT '97*, no. 1233 in LNCS, pp. 256–266. IACR, Springer, May 1997.

[SS96] M. Shub and S. Smale. *On the intractability of Hilbert's nullstellensatz and an algebraic version of "$NP \neq P$"*. Duke Mathematical Journal, **81**(1):47–54, 1996.

A Upper Bounds for and Examples of Integral-Generic Factoring

This section gives examples of, and thereby upper bounds for the length of, a generic factoring algorithm, .

A.1 Trial division

In this section, versions of the trial division factoring algorithm are described as integral-generic factoring algorithms.

Suppose that the random integer variable n we wish to factor is bounded in the interval $[1, N^2]$ with $N \geqslant 3$. For the sake of simplicity, assume that n is uniformly distributed in this interval.

The integer-generic program P_N computes $N!$, the image of this integer in whatever ring it is evaluated, and has length $L = 2N - 3$. Its steps are

$$\begin{aligned} \sigma_1 &= (+, 0, 0), \\ \sigma_2 &= (+, 1, 0), \\ \sigma_{2k-4} &= (+, 2k-6, 0), \qquad \forall k \geqslant 4, \\ \sigma_{2k-3} &= (\times, 2k-4, 2k-5), \qquad \forall k \geqslant 3. \end{aligned}$$

To see that P computes $N!$, verify that $x_0 = 1$ and $x_2 = 2$. Verify by induction that $x_{2k-4} = k$ for $k \geqslant 3$. Next, verify by induction that $x_{2k-3} = k!$ for $k \geqslant 2$.

A breadth-1 generic factoring algorithm based on this program P_N computes $d = \gcd(n, N!)$. If n is composite and but has a prime factor larger than N, then $d \notin \{1, n\}$, and the factoring algorithm will succeed. So, this generic factoring algorithm succeeds with some probability μ_N. For large enough N, we have $\mu_N \approx 1 - \rho(2) = \log(2)$, where ρ is the Dickman function.

A.2 Lenstra's elliptic curve method

We conjecture that the following version of Lenstra's elliptic curve method factoring algorithm [Len87] is an integral-generic factoring algorithm, with approximately the same complexity as Lenstra's original algorithm.

The program has B preliminary phases, followed by one final phase.

In preliminary phase i, two large integers x and y are selected, and computed via steps of an integral-generic program. The range of the random variables x and y are chosen to be sufficiently large that their distribution is almost uniform modulo the prime factors of n. Perhaps x and y could be chosen uniformly from the set $[1, E(n)^2]$, where $E(n)$ is the expected value of the number n to factor. This may be sufficient to ensure the desired success rate of this algorithm.

We then consider x and y as coordinates of a point on elliptic curve, $y^2 = x^3 - 3x + b$ for some b. We first solve for b. Then, using the standard elliptic curve arithmetic we compute the point $M(x, y)$ where M is a very large smooth number. We use projective coordinates, representing all points by three ring elements.

Preliminary phase i phases ends in a point whose projective z-coordinate z_i. The final phase computes the product $z_1 z_2 \dots z_B$.

Clearly this describes an breadth-1 integral-generic factoring algorithm. We conjecture that choice of B and M similar to the corresponding values in Lenstra's original description of the elliptic curve factoring algorithm, that this integral-generic factoring algorithm has a non-negligible success rate μ.

In particular, we conjecture B and M can be chosen such that $\mu \approx 1/2$ and the total length L needed is about $O(\exp((1 + o(1))\sqrt{(\log p)(\log\log p)}))$ where p is the size of the smallest prime factor of n.

B Towards Lower Bounds for Generic Factoring

For the sake of clarity, Section 3 focused on integral-generic factoring algorithms only. In this section, we suggest how lower bounds for integral-generic factoring algorithms might be generalized to lower bounds for generic factoring. Some of the techniques for doing this may have already been describe by Aggarwal and Maurer [AM09], Boneh and Venkatesan [BV98], and Cheng [Che04].

A lower bound for the length L of a generic factoring algorithm should be expressed in terms of other parameters: n, w, m, μ, and the qualifiers algorithm, whether divisionless, deterministic or straight line.

B.1 Broader algorithms

Suppose a generic factoring algorithm has breadth m and success rate μ. By picking one of its m outputs at random and designating to be the sole output, then we obtain a breadth-1 generic factoring algorithm with success rate at least μ/m. Therefore, lower bounds for broader generic factoring algorithm may be derived from lower bounds for breadth-1 factoring algorithms of lower success rate.

Another way to reduce breadth is to choose random subset of size c of the m outputs, and multiply these outputs. If c is small enough, then there is some probability that can bound in terms μ and m, that exactly one of the c outputs finds a factor in n.

Typically, in a broad generic factoring algorithm of cryptologic interest, the probability of each factor revealing the output are similar and somewhat independent, and this can be used to advantage to efficiently reduce the breadth.

B.2 Wider algorithms

Suppose a generic factoring algorithm has width w and length L. Furthermore, suppose that n is bounded in size. Then we can replace each input variable with a segment of operational steps that have been chosen to compute a random integer whose distribution is nearly uniform with respect to every value taken by n. (True uniformity would require these integers to be very large, and therefore possibly to be very long.)

If sufficient closeness to uniformity can be achieved, then one can argue that the success rate of the resulting derived width-0 generic factoring algorithm is not substantially diminished. The resulting length of the program is approximately $L + wU(n, \mu)$ where $U(n, \mu)$ is the minimum average length needed to achieve the requisite closeness to uniformity. (Upon further analysis, it may turn out that the number of additional steps also depends on w.)

B.3 Generic algorithms with division

Loosely speaking, by doubling the length of a program, divisions can be removed, by maintaining ratios. A ratio reveals a factor if and only if its numerator reveals a factor.

B.4 Deterministic generic factoring

Suppose that a factoring algorithm is deterministic but not straight line. In this case, its program uses relational steps.

As above, we wish to remove these steps, yet still be able to factor. We may be able to use the idea of Aggarwal and Maurer of classifying such steps as critical or non-critical. Non-critical steps, over random n, almost always give the same result. These steps may be removed harmlessly to their most frequent value, which can be observed. More precisely, the steps are not removed, but rather replaced by null steps multiplying a previous step by 1.

Critical steps can be replaced at random. Suppose that there are c critical steps. Then for random n, and random choice of critical step decisions, we get another algorithm with success at least $\mu 2^{-c}$. Perhaps more sophisticated techniques can be used to less severely degrade the success rate.

B.5 Non-deterministic generic algorithms

Similarly to replacement of random inputs by random operational steps in an extended program, it seems that one may be able to replace representational steps by random steps.

C Schools of Thought

One rather strict school of thought in cryptology is that a cryptographic technique should only be considered secure to the extent that it can be proved secure. This is mainly applied to advance a preference between two cryptographic technique, where the one with better proofs of security deemed as secure. Such preference may in fact be contrary to a well-established history of remaining unbroken under heavy use and intense scrutiny. A rationale may be that an unforeseen attack may just be waiting around the corner.

Nevertheless, many such proofs of security actually rely on unproven security assumptions, such as factoring being hard, whose security is assured primarily on well-established history of reaming unbroken under heavy and intense scrutiny. Strict adherence to the above school of thought demands seeking proofs of security for such assumptions or else other assumptions that have security proofs. For example, strictly speaking this school of thought should prefer discrete logarithm cryptography (DLC) over integer factoring cryptography (IFC), because the former has better security proofs, provided Shoup's proof, whereas integer factoring cryptography has no useful proofs. That Shoup's proof for DLC have a limitations is a given, but IFC has no proofs at all, so the case for DLC is better than nothing.

D Lambda Examples

This section gives some examples of λ computations.

First, $\lambda(83) = 5$. The only length 5 integral-generic program computing 83 is:

$$((+,0,0),(+,0,1),(\times,2,2),(\times,3,3),(+,1,4)).$$

Second, $\lambda(720) = 6$. Seven integral-generic programs compute 720, with intermediate values given by one of the five rows of the following array:

1	2	3	9	27	729	720
1	2	3	9	81	80	720
1	2	3	9	81	729	720
1	2	4	6	24	30	720
1	2	4	16	20	36	720

The last two rows account for two programs each because 4 can be computed as either $2+2$ or 2×2.

The $n \leqslant 10000$ for which $\lambda(n) = \lceil 1 + \log_2 \log_2 n \rceil$ are:

$$2,3,4,5,6,8,9,16,17,18,20,24,25,27,32,36,64,81,256,257,258,260,272,288,$$
$$289,320,324,400,512,576,625,729,1024,1296,4096,6561.$$