A Practical Attack against Knapsack based Hash Functions (extended abstract)

Antoine Joux[1] and Louis Granboulan[2]

[1] DGA/CELAR
[2] ENS/LIENS

Abstract. In this paper, we show that lattice reduction is a very powerful tool to find collision in knapsack based compression-functions and hash-functions. In particular, it can be used to break the knapsack based hash-function that was introduced by Damgard [3]

1 Introduction

The knapsack problem, is a well-know NP-complete problem that can quite easily be used to construct cryptosystems or hash-functions. Thus many cryptographic functions have been based on this problem, however, lattice reduction is a very powerful tool to break knapsack-based cryptosystems. This was shown by Lagarias and Odlyzko [5], and their result was improved by Coster and al in [2].

In this article, we show that lattice reduction can also be used to find collisions in knapsack-based compression-functions. And we apply this tool to Damgard's hash-function based on such a knapsack compression function. A completely different kind of attack was already presented by P. Camion and J. Patarin in [1], however, it was not implemented, and it permitted to find collisions in the compression function rather than in the full hash function.

Throughout this paper, in order to simplify the analysis of the problem, we suppose that we are granted access to a lattice reduction oracle, that given any lattice produces a shortest vector in this lattice. In practice, this oracle will be replaced either by the LLL algorithm [6] or a blockwise Korkine-Zolatarev algorithm [7]. This approach, which enables us to focus on the reduction of collision search to lattice reduction, without needing to worry about the state of the art in lattice reduction algorithms, is also used in [2].

2 First approach to the reduction technique

In this section, we define a lattice associated to a given knapsack-based compression-function in such a way that collisions correspond to short vectors.

Let us now make a few notations precise, before describing the reduction technique. Given any set of n integers, $a_1, \ldots, a_n$, we can define a integer valued function which given any vector x in $\{0,1\}^n$ computes $S(x) = \sum_{i=1}^{n} a_i x_i$. We can also define the density of S, $d = \frac{n}{\max_i a_i}$. Then $\tau = 1/d$ is the compression rate of the compression function S, since S transform n bits into $\alpha n + \log_2(n)$ bits. In the sequel, in order to simplify the analysis, we want to ignore the $\log_2(n)$ term, thus we will work with modular knapsacks instead of usual knapsacks. However, similar results can be obtained in the non-modular case, as will be shown in the full paper. In this paper, we use the same approach as in Coster's analysis of the Lagarias-Odlyzko attack, more precisely, we fix a value for τ we let $m = \lfloor \tau n \rceil$ and choose for the a_i random values lower than 2^m. As n tends toward infinity, this generating process models random knapsacks of compression rate τ. These knapsack are then considered modulo 2^m.

In order to search collisions in such a modular knapsack, we reduce the following lattice:

$$B = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n & K2^m \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Note that this lattice is a modular variation of Lagarias-Odlyzko's lattice for solving knapsack problems (see [5]). Let us consider the various short vectors that can occur. Since K is large, it is clear that the first component of a short vector is 0. Looking at the other components, two things can happen, either they are all 0, 1 or -1, or not. If the shortest vector is of the first type, we clearly get a collision, since having the vector:

$$e = \begin{pmatrix} 0 \\ \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

in the lattice B, with all ϵs 0, 1 or -1 implies:

$$\sum_{i=1}^{n} \epsilon_i a_i = 0$$

and thus:

$$\sum_{\epsilon_i = 1} a_i = \sum_{\epsilon_i = -1} a_i.$$

In general, we cannot show that the shortest vector will be of the proper type, we actually expect that the probability for such a vector to occur, tends exponentially fast towards 0. However, we show in the next section, that using a lattice reduction oracle, we can find collisions in a knapsack compression function, must faster than by exhaustive search or a birthday paradox attack. We also show that in small dimensions, the naïve algorithm works in practice by giving experimental result on the success rate of the non-modular naïve algorithm using LLL, or a blockwise Korkine-Zolotarev reduction algorithm in place of the lattice reduction oracle.

3 Average size of the collision

In this section, we show how to compute the average size of a collision for the kind of knapsacks we are looking at. Let us consider random knapsacks of n elements and fixed compression rate τ, we have:

Lemma 1. *Let ρ be a fixed constant such that*

$$\rho + H_2(\rho) > \tau > \rho$$

With probability tending exponentially to 1 when n tends to infinity, there exists a relation

$$\sum_{i=1}^{n} \epsilon_i a_i = 0$$

where all ϵ_is are 0, 1 or -1 and where

$$\sum |\epsilon_i| \leq \rho n$$

In the above $H_2(\alpha)$ denotes, as usual, $-\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$.
Let us sketch the proof:

Consider the family of all possible vectors with n coordinates, all of them 0, 1 or -1, with size ρn. The number of elements in this family is roughly $2^{\rho n} 2^{H_2(\rho) n}$. A collision is expected for $N > 2^{\tau n}$, thus leading to the above lemma.

This proof can be made precise, and will be presented in [4].

4 Finding collisions using a lattice reduction oracle

Given a random knapsack of size n and compression rate τ, we know that almost surely it contains a collision of size $(L(\tau)+\epsilon)n$. Suppose now, that we can guess αn non zero elements of such a collision, then we can form another random modular knapsack by replacing the αn elements involved in the guess by their ponderated sum modulo 2^m. We thus obtain a modular knapsack containing $(1-\alpha)n+1$ random modular numbers $b_0, b_1, \ldots, b_{(1-\alpha)n}$. We can associate to the knapsack the following lattice:

$$B' = \begin{pmatrix} Kb_0 & Kb_1 & \cdots & Kb_{(1-\alpha)n} & K2^m \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

By construction, this lattice contains a short vector of size $(L(\tau)+\epsilon-\alpha)n+1$. Transposing the main argument from Lagarias-Odlyzko, we can show that with a probability tending exponentially fast towards 1, this vector is the shortest existing in the lattice B', as soon as the density of the new knapsack is smaller than a function of the relative size of the short vector in the new knapsack. This relative size is:

$$\frac{L(\tau)+\epsilon-\alpha}{1-\alpha},$$

and the density is:

$$\frac{1-\alpha}{\tau}.$$

Since the condition from Lagarias-Odlyzko involves complicated functions, we can't give a close form for the solution. However, we have computed the graph of α as a function of τ, see figure 4 for the curve corresponding to the limiting case $\epsilon = 0$.

We can now derive an semi-exhaustive search algorithm, where we try random subsets of size αn, and all partitions of these subsets into 1s and -1s. The probability for a random subset to be part of a fixed collision of size $L(\tau)n$ (we are still considering the case $\epsilon = 0$) is roughly $2^{\mu n}$, where μ is a function of τ (see figure 4). Thus this semi-exhaustive algorithme costs $O(2^{(\alpha+\mu)n})$ steps, where each step is a call to the lattice reduction oracle. In the worst case, when $\tau = 1$, this yields a running time of approximately $O(2^{n/1000})$ steps. This proves that in the general case, searching a collision in a knapsack problem is much more efficient using lattice reduction than using a birthday paradox attack ($O(2^{\tau n/2})$ steps). On the other hand, this is still an exponential time algorithm.

5 Practical results in small size

Looking at the results of the previous section, it is tempting to forget that we are dealing with asymptotic results, and to look what happen if we substitute finite values for n in the formulaes. Moreover, as long as αn stays below 1, we can argue that there is no need to guess the missing bit and hope that a single lattice reduction will find a collision.

In this section we give a table of practical results, using the worst case compression rate 1, and various lattice reduction algorithms, namely LLL and Blockwise Korkine-Zolotarev reduction with blocs of size 10 and 20. These results concern non-modular knapsacks, and thus use the following lattice:

$$B = \begin{pmatrix} Ka_1 & Ka_2 & \cdots & Ka_n \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}.$$

The tables in figures 5, 5 and 5 contain success rate and average user running times on a Sun sparcstation for knapsacks with compression rate 1, i.e. for worst case compression knapsacks. For each choice of dimension and algorithm, the success rate and running are averaged over 10 random knapsacks. These tables show that using LLL, we can find collisions with non-negligible success rate up to dimension ≈ 60, with BKZ10 up to dimension ≈ 90 and with BKZ20 up to dimension ≈ 105.

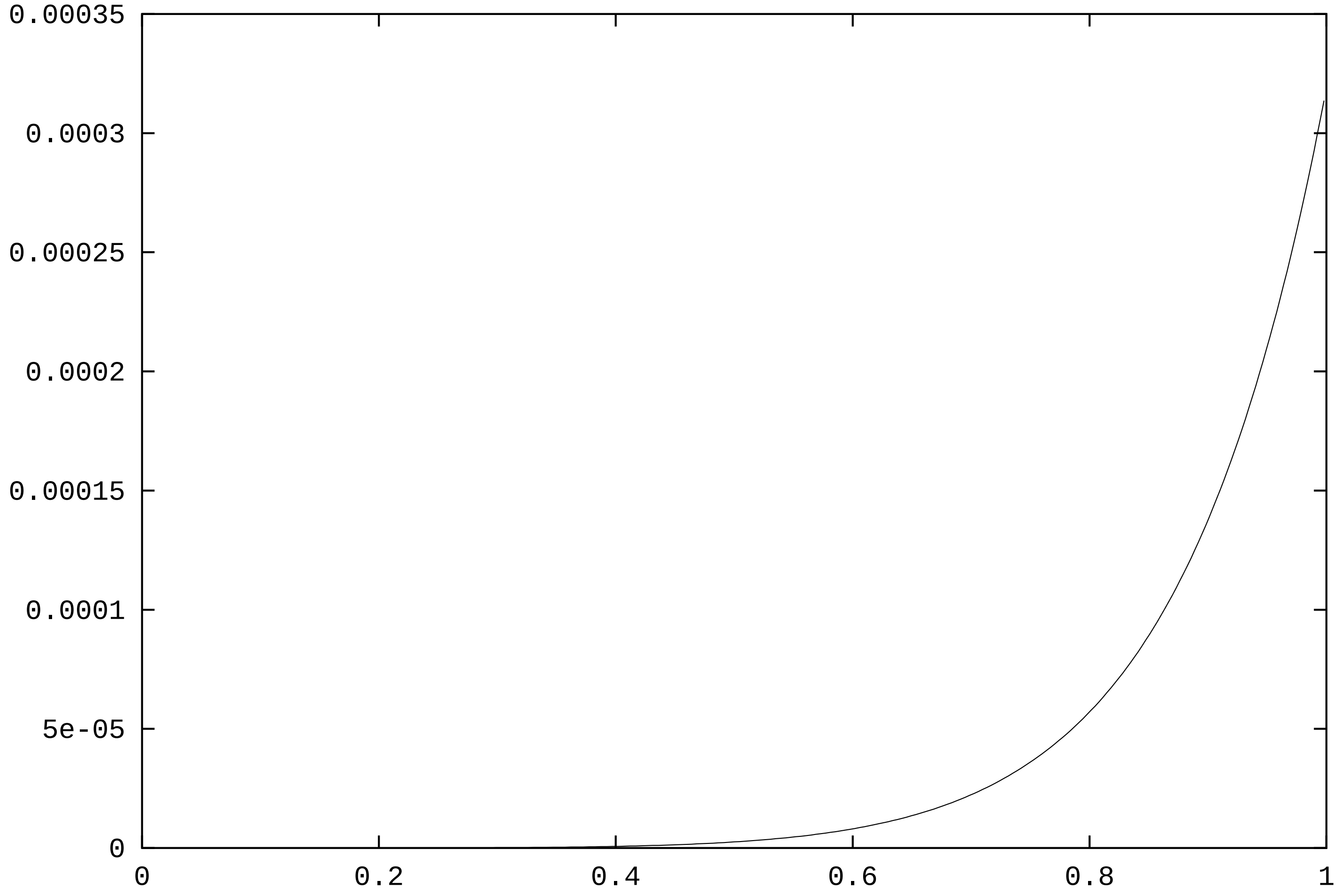

Fig.1. α as a function of τ

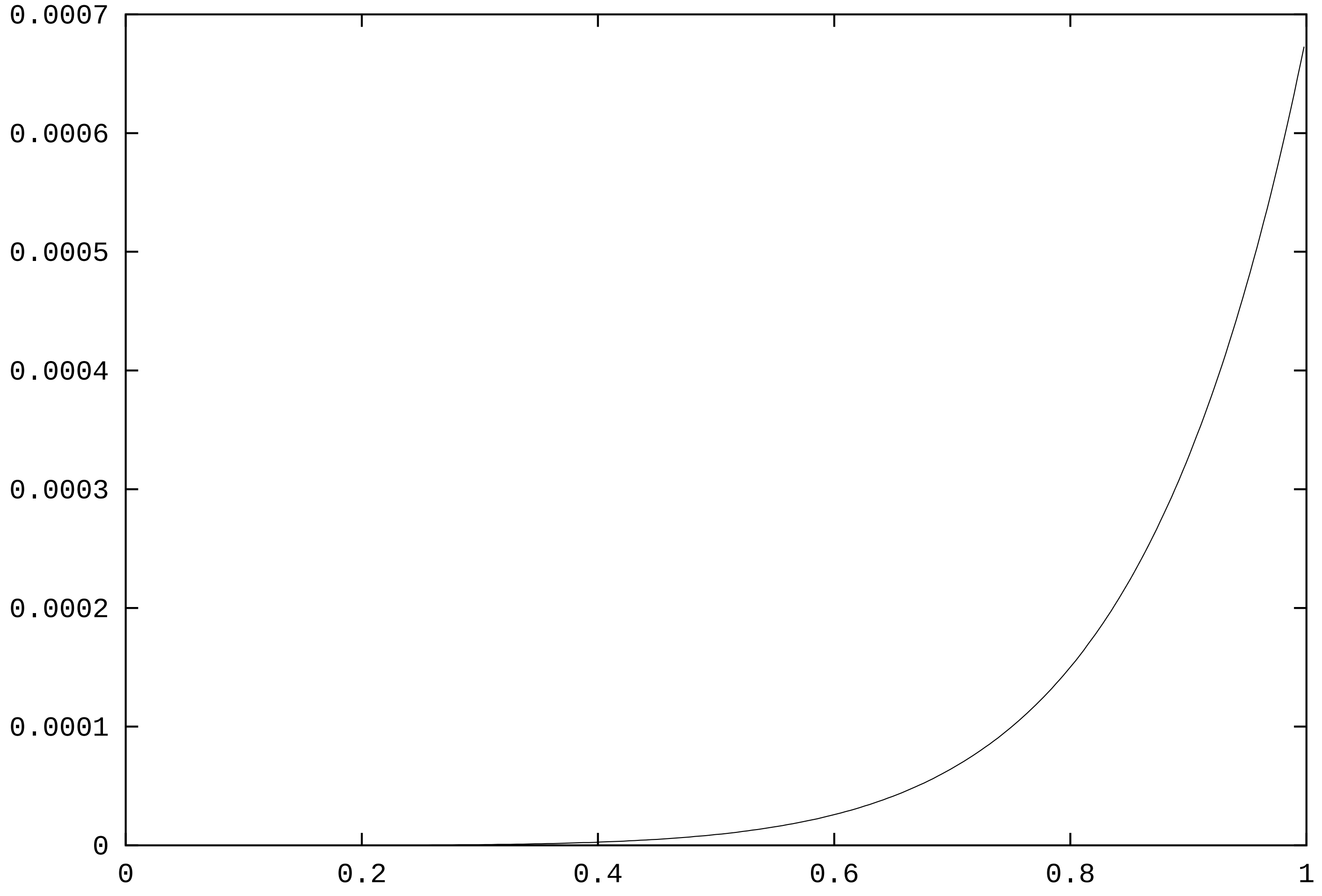

Fig. 2. μ as a function of τ

Dimension	Successes (/10)	User CPU	Dimension	Successes (/10)	User CPU
5	10	0.0s	6	10	0.0s
7	10	0.0s	8	10	0.0s
9	10	0.0s	10	10	0.0s
11	10	0.0s	12	10	0.0s
13	10	0.0s	14	10	0.0s
15	10	0.0s	16	10	0.0s
17	9	0.0s	18	9	0.0s
19	10	0.0s	20	10	0.0s
21	10	0.0s	22	10	0.0s
23	10	0.0s	24	10	0.0s
25	9	0.6s	26	9	0.9s
27	10	1.0s	28	9	1.1s
29	9	1.2s	30	10	1.4s
31	9	2.0s	32	10	1.9s
33	9	2.0s	34	5	2.5s
35	9	2.9s	36	7	2.9s
37	6	3.3s	38	7	3.5s
39	4	3.7s	40	5	4.5s
41	8	4.3s	42	7	5.3s
43	8	5.2s	44	5	5.5s
45	5	5.6s	46	6	6.3s
47	2	7.5s	48	3	7.5s
49	2	7.6s	50	2	8.2s
51	1	8.8s	52	2	8.8s
53	3	9.0s	54	3	10.8s
55	0	11.3s	56	0	10.9s
57	1	12.2s	58	1	12.5s
59	2	13.1s	60	0	14.6s
61	0	14.4s	62	0	16.3s
63	1	15.0s	64	0	17.4s
65	0	17.9s			

Fig. 3. Results using LLL

6 Attacking Damgard hash-function

In [3], Damgard proposed to base an hash function on a knapsack compression function using 256 non modular numbers of size 120 bits. This roughly corresponds to a compression rate of 1/2. However, in general, finding collisions for a hash function is harder than in for the corresponding compression function, because the first half of the data entering the compression function is either a fixed initialisation value or the result of previous rounds of the hash functions. Luckily, here we can get rid of this problem, by removing the first half of the knapsack. We thus get a compression function with compression rate roughly 1. However, according to our analysis, it is still possible to find collision in a compression function involving 128 numbers of 120 bits.

The main problem in order to implement this attack against Damgard hash-

Dimension	Successes (/10)	User CPU	Dimension	Successes (/10)	User CPU
10	10	0.0s	11	10	0.0s
12	10	0.0s	13	10	0.0s
14	10	0.0s	15	10	0.0s
16	10	0.0s	17	9	0.0s
18	9	0.0s	19	10	0.0s
20	10	0.0s	21	10	0.3s
22	10	0.5s	23	10	1.1s
24	9	1.1s	25	10	1.6s
26	10	1.7s	27	10	2.3s
28	10	3.0s	29	10	2.9s
30	9	3.9s	31	10	6.8s
32	10	7.2s	33	10	7.0s
34	10	7.8s	35	10	10.1s
36	10	11.1s	37	10	13.1s
38	10	14.0s	39	10	20.0s
40	10	19.3s	41	9	20.6s
42	9	25.1s	43	10	26.5s
44	10	31.3s	45	9	32.7s
46	10	32.0s	47	10	38.5s
48	10	45.3s	49	10	40.9s
50	9	53.7s	51	10	55.8s
52	7	63.1s	53	9	65.2s
54	10	75.5s	55	10	68.0s
56	10	71.7s	57	6	90.1s
58	7	77.5s	59	9	100.1s
60	8	130.7s	61	7	95.8s
62	6	109.8s	63	6	126.0s
64	6	143.4s	65	8	135.4s
66	9	151.2s	67	5	158.7s
68	4	147.0s	69	5	153.3s
70	3	190.2s	71	4	192.8s
72	1	185.8s	73	2	186.1s
74	0	186.9s	75	3	190.9s
76	4	198.8s	77	2	220.6s
78	4	235.2s	79	2	244.0s
80	4	298.3s	81	1	285.8s
82	2	329.1s	83	3	331.2s
84	2	344.8s	85	0	290.2s
86	2	360.9s	87	0	350.9s
88	1	379.3s	89	0	309.0s
90	0	373.2s	91	0	395.9s
92	0	419.3s	93	0	403.2s

Fig. 4. Results using BKZ10

Dimension	Successes (/10)	User CPU	Dimension	Successes (/10)	User CPU
20	10	0.1s	21	10	0.2s
22	10	0.5s	23	10	1.0s
24	10	1.2s	25	10	1.4s
26	10	2.1s	27	10	2.7s
28	10	3.5s	29	10	3.7s
30	10	5.2s	31	10	6.0s
32	10	9.0s	33	10	11.9s
34	10	13.7s	35	10	16.5s
36	10	13.9s	37	10	19.9s
38	10	27.8s	39	10	26.9s
40	10	37.5s	41	10	41.1s
42	10	43.0s	43	10	55.8s
44	8	56.0s	45	9	51.3s
46	10	84.9s	47	10	93.0s
48	10	91.5s	49	10	108.5s
50	10	134.1s	51	10	184.2s
52	9	163.7s	53	10	223.1s
54	10	303.8s	55	10	243.4s
56	10	205.4s	57	10	255.9s
58	9	320.6s	59	10	285.5s
60	10	424.2s	61	10	378.3s
62	10	404.7s	63	8	463.9s
64	9	404.1s	65	8	464.9s
66	8	445.3s	67	8	656.3s
68	7	518.3s	69	7	545.0s
70	10	783.3s	71	10	597.5s
72	8	706.4s	73	8	513.0s
74	8	685.5s	75	4	744.0s
76	8	740.1s	77	6	747.0s
78	3	924.0s	79	5	979.7s
80	5	945.2s	81	4	993.7s
82	5	1215.9s	83	7	1025.4s
84	6	1239.2s	85	5	1248.8s
86	5	1461.2s	87	2	1434.8s
88	4	1654.3s	89	5	1251.4s
90	2	1693.5s	91	2	1773.3s
92	3	1997.1s	93	2	2074.4s
94	4	1876.6s	95	2	1869.3s
96	1	1921.6s	97	0	2201.8s
98	0	1934.0s	99	0	1788.4s
100	2	2064.2s	101	0	2493.1s
102	0	2490.7s	103	0	2622.5s
104	1	2562.0s	105	0	2750.3s

Fig. 5. Results using BKZ20

function is to find a suitable lattice reduction algorithm. We know from the previous section that BKZ20 is not strong enough in this case, and BKZ reduction with larger blocks is too slow. Luckily, C.P. Schnorr and M. Euchner have presented in their paper [7], a very efficient lattice reduction algorithm called pruned blockwise Korkine-Zolotarev reduction. We have slightly modified the algorithm in order to tune it for the lattices we are dealing with, and we also introduced a limit on the running time of the program. Tests were performed both on the Sun sparcstation 10 and on an IBM RS6000 model 590 which is roughly 1.7 times faster. We used time limit 1h and 4h on the IBM and 24h on the sparcstation (this correspond roughly to 14h on the IBM). We obtained the following success rates:

Time limit	# trials	# success	rate
1h	100	3	0.03
4h	100	10	0.10
$\approx$ 14h	30	8	0.27

This clearly that collisions can be found in Damgard's hash-function.

7 Acknowledgments

We would like to thank Jacques Stern for his helpful comments and for his proof of lemma 1.

References

1. P. Camion and J. Patarin. The knapsack hash-function proposed at crypto'89 can be broken. In D. W. Davies, editor, *Advances in Cryptology, Proceedings of Eurocrypt'91*, volume 547 of *Lecture Notes in Computer Science*, pages 39–53, New York, 1991. Springer-Verlag.
2. M. J. Costerr, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C.-P. Schnorr, and J. Stern. Subset sum algorithms. *Comp. Complexity*, 2:11–28, 1992.
3. I. Damgard. A design principle for hash functions. In *Advances in Cryptology, Proceedings of Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 25–37, New York, 1989. Springer-Verlag.
4. A. Joux and J. Stern. Lattice reduction: a toolbox for the cryptanalyst. submitted to the Journal of Cryptology, 1994.
5. J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. Assoc. Comp. Mach.*, 32(1):229–246, 1985.
6. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.
7. C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In L. Budach, editor, *Proceedings of Fundamentals of Computation Theory 91*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85, New York, 1991. Springer-Verlag.

This article was processed using the LaTeX macro package with LLNCS style