Solving Weakened Cryptanalysis Problems for the Bivium Keystream Generator in the Volunteer Computing Project SAT@home

Oleg Zaikin, Alexander Semenov, and Ilya Otpuschennikov

Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
zaikin.icc@gmail.com, biclop.rambler@yandex.ru, otilya@yandex.ru

Abstract. In this paper, a cryptanalysis of the Bivium keystream generator in the SAT form is considered. For encoding the initial cryptanalysis problem into SAT a special program system TRANSALG was used. For an obtained SAT instance we use Monte Carlo method to search for a partitioning with good time estimation. Several weakened cryptanalysis instances of the Bivium generator were successfully solved in the volunteer computing project SAT@home using corresponding partitionings found on a computing cluster.

Keywords: SAT, logical cryptanalysis, Monte Carlo method, volunteer computing, Bivium, SAT@home, Transalg

1 Introduction

Volunteer computing [19] is a type of distributed computing which uses computational resources of PCs of private persons called volunteers. Each volunteer computing project is designed to solve one or several hard problems. When PC is connected to the project, all the calculations are performed automatically and do not inconvenience user since only idle resources of PC are used. Nowadays the most popular platform for organizing volunteer computing projects is BOINC [1] that was developed in Berkeley in 2002. Today there are about 70 active volunteer projects, the majority of them based on BOINC.

A volunteer computing project consists of the following basic parts: server daemons, database, web site and client application. The daemons include work generator (generates tasks to be processed), validator (checks the correctness of the results received from volunteer's PCs) and assimilator (processes correct results). Client application should have versions for the widespread computing platforms.

One of the attractive features of volunteer computing is its low cost — to maintatin a project one needs only a dedicated server working 24/7. Main difficulty here lies in the development of software and in database administration. In addition, it is crucial to provide the feedback to volunteers using the web site of the project and special forums. Another attractive feature of this type of computing is that volunteer project can solve some particular hard problem for months or even years with good average performance.

Wide class of problems from modern computer science can be effectively reduced to Boolean satisfiability problem (SAT) [2]. SAT problems are usually considered as

the problems of search for solutions of Boolean equations in the form of CNF=1, where CNF is a conjunctive normal form. There are many works in which various combinatorial problems are reduced to SAT and solved in this form. For example, such problems can be found in areas of verification, cryptography, combinatorics and bioinformatics. Usually if the cryptanalysis is considered as a SAT problem then it is called a *logical cryptanalysis* [15]. In this case to find a secret key it is sufficient to find a solution of corresponding satisfiable SAT instance. For solving corresponding SAT instance one usually needs large computational resources. A volunteer computing project can provide such resources.

Below we present a brief outline of our paper. In the second section, we describe how we utilize the TRANSALG tool for reduction of the cryptanalysis problem of the Bivium generator to SAT. In the third section various decomposition techniques for SAT are discussed. In the fourth section experimental results on solving weakened cryptanalysis problems for the Bivium generator in the volunteer computing project SAT@home are presented.

2 Propositional Encoding of Cryptanalysis Problem for Bivium

The Bivium generator [5] uses two shift registers of a special kind (see Fig. 1). The first register contains 93 cells and the second contains 84 cells. To initialize the generator, a secret key of length 80 bit is put to the first register, and a fixed (known) initialization vector of length 80 bit is put to the second register. All remaining cells are filled with zeros. An initialization phase consists of 708 rounds during which keystream output is not released.

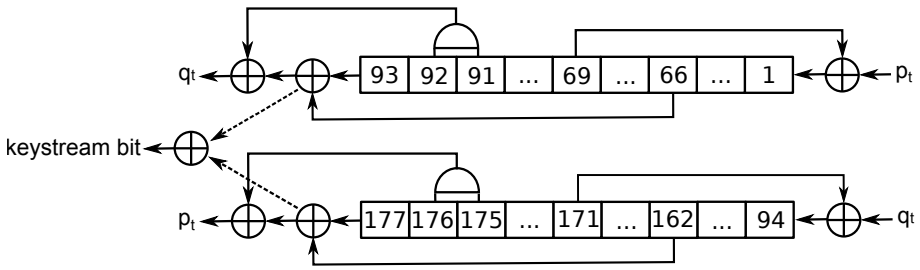

Fig. 1. Scheme of the Bivium generator

In accordance with [16, 25] we considered cryptanalysis problem for Bivium in the following formulation. Based on the known fragment of keystream we search for the values of all registers cells at the end of the initialization phase. Therefore, in our experiments we used CNF encodings where initialization phase was omitted. Usually it is believed that to uniquely identify the secret key it is sufficient to consider keystream fragment of length comparable to the total length of shift registers. Here we followed [8, 25] and set the keystream fragment length for Bivium cryptanalysis to 200 bits.

To encode algorithm of the Bivium generator into CNF we used TRANSALG tool [20]. As an input TRANSALG takes a source code of a program in the domain specific language named *TA language*. TA language is a procedural programming language with C-like syntax. This source code must describe an algorithm of computing of a discrete function. Output of TRANSALG is a CNF which encodes the considered algorithm.

Translation of a TA-program consists of two main stages. On the first stage a source code of a TA-program is parsed, and concrete syntax tree is constructed. On the second stage of translation, the symbolic execution of TA-program is performed. According to the concept of the symbolic execution, interpreter operates not with concrete data, but with symbols which encode this data — Boolean variables and Boolean vectors (arrays). The symbolic execution of a TA-program results in a set of Boolean formulas, which is called *propositional encoding* of the algorithm. We will consider constructing of propositional encoding by TRANSALG on the example of TA-program which describes algorithm of the Bivium keystream generator [5].

The TA-program for the Bivium generator starts from the declaration of integer constants (see Listing 1.1). Here constant *len* is equal to total length of the shift registers (177 bits); constants *lenA* and *lenB* define the lengths of each register (93 and 84 bits respectively); *stream_len* defines the number of bits of a keystream produced by the generator (200 bits). After this, arrays of variables of type `bit` are declared. Main data type in the TA-language is the `bit` type. TRANSALG uses this type to establish links between variables used in a TA-program and Boolean variables included into corresponding propositional encoding. It is important to distinguish between these two sets of variables. Below we will refer to variables that appear in a TA-program as *program variables*. All variables included in a propositional encoding are called *encoding variables*.

Global variables of type `bit` can be declared with attribute `__in` or `__out`. Attribute `__in` marks program variables associated with Boolean variables that encode input of an algorithm. In fact, in such a way we declare a set of variables with the following feature: any assignment of these variables derives values of all other variables in the propositional encoding. It should be noted that in every TA-program that describes some algorithm there must be declared variables which encode input of the considered algorithm. Attribute `__out` marks variables which, after TA-program is executed, contain an output of the algorithm.

Listing 1.1. Declarations of global variables and constants

```
define len 177;
define lenA 93;
define lenB 84;
define stream_len 200;
__in  bit reg[len];
__out bit stream[stream_len];
```

In the Listing 1.1 the bit array `reg` with attribute `__in` is declared. This array contains a state of the shift registers of the generator. Attribute `__in` reports to the translator that initial state of the registers is unknown. Encoding variables, linked with program variables marked with attribute `__in`, get first `len` numbers in a propositional encoding.

Also in Listing 1.1 the bit array `stream` with attribute `__out` is declared. This array contains the keystream produced by the generator. Encoding variables, linked with program variables marked with attribute `__out`, get last `stream_len` numbers in a propositional encoding. Thus by using attributes `__in` and `__out` we uniquely identify the sets of Boolean variables in a propositional encoding, which encode input and output of an algorithm respectively.

In the considered TA-program shifting of the registers is implemented in the form of the procedure `shift_regs` (see Listing 1.2). This procedure updates the state of the global bit array `reg`, according to the Bivium algorithm [5]. Note that as a result of interpretation of the procedure `shift_regs`, new encoding variables are created only to represent the result of feedback functions calculation. Operations of copying data between cells add nothing to a propositional encoding — only links between program variables and corresponding Boolean encoding variables are changed.

Listing 1.2. Function that implements one shift of the Bivium registers

```
void shift_regs()
{
  bit t1 = reg[65]^reg[90]&reg[91]^reg[92]^reg[170];
  bit t2 = reg[161]^reg[174]&reg[175]^reg[176]^reg[68];

  int i;
  for(i = lenA - 1; i > 0; i = i - 1){
    reg[i] = reg[i-1];
  }
  reg[0] = t2;

  for(i = lenA + lenB - 1; i > lenA; i = i - 1){
    reg[i] = reg[i-1];
  }
  reg[lenA] = t1;
}
```

Interpretation of a TA-program starts from executing the function `main`. This function is the entry of any TA-program. Presented in the Listing 1.3 function `main` implements the base loop of the Bivium generator. One iteration of this loop outputs one bit of keystream. For this purpose the current state of the four registers cells are taken — their sum mod 2 gives next keystream bit. After this, the shifting of the registers is performed by calling function `shift_regs`.

Listing 1.3. Implementation of the base work loop of the Bivium generator

```
void main()
{
  for(int i = 0; i < stream_len; i = i + 1){
    bit t1 = reg[65] ^ reg[92];
    bit t2 = reg[161] ^ reg[176];
    stream[i] = t1 ^ t2;
```

```
    shift_regs();
  }
}
```

The considered TA-program describes the generation of 200 keystream bits by the Bivium generator. The translation of this program results in the system of 465 Boolean equations. The transition from this system to the CNF is performed using the Tseitin transformations [27] and procedures of Boolean minimization. During this transition to minimize Boolean formulas, the ESPRESSO library[1], which is integrated to TRANSALG in the form of separate program module, is used. Procedures of Boolean minimization, implemented in Espresso, operate with truth tables. If the size of a constructed formula does not allow to effectively work with its truth table, then to decompose such formulas into subformulas the Tseitin transformations are used. The final CNF which encodes generating of 200 keystream bits by Bivium is constructed over the set of 642 Boolean variables, and consists of 9560 clauses and 49920 literals.

3 SAT partitioning

In [11–14] various approaches to partitioning SAT were studied. Below we will use the terminology from [11]. Consider a satisfiability problem for an arbitrary CNF C. Partitioning of C is a set of formulas

$$C \wedge G_i, i \in \{1, \ldots, S\}$$

such that for any $i, j : i \neq j$ formula $C \wedge G_i \wedge G_j$ is unsatisfiable and

$$C \equiv C \wedge G_1 \vee \ldots \vee C \wedge G_S.$$

When one has a partitioning of an original SAT instance, satisfiability problems for $C \wedge G_j, j \in \{1, \ldots, S\}$ can be solved independently in parallel.

There exist various partitioning techniques. For example one can construct $\{G_j\}_{j=1}^{S}$ using a scattering procedure [12], a guiding path solver [28] or a lookahead solver [10]. Unfortunately, for these partitioning methods it is hard in general case to estimate the time required to solve an original problem. From the other hand in a number of papers about logical cryptanalysis of several keystream generators there was used a partitioning method that makes it possible to construct such estimations in quite a natural way. In particular, in [17, 8, 26, 25] for this purpose the information about the time to solve small number of subproblems randomly chosen from the partitioning of an original problem was used. In the paper [23] strict formal description of this idea within the borders of the Monte Carlo method in its classical form [18] was given.

Consider a satisfiability problem for an arbitrary CNF C over a set of Boolean variables $X = \{x_1, \ldots, x_n\}$. We call an arbitrary set $\tilde{X} = \{x_{i_1}, \ldots, x_{i_d}\}$, $\tilde{X} \subseteq X$ a decomposition set. Consider a partitioning of C that consists of a set of 2^d formulae

$$C \wedge G_j, j \in \{1, \ldots, 2^d\}$$

[1] http://embedded.eecs.berkeley.edu/pubs/downloads/Espresso/index.htm

where G_j, $j \in \{1,\ldots,2^d\}$ are all possible minterms over $\tilde{X}$. Note that an arbitrary formula G_j takes a value of true on a single truth assignment $\left(\alpha_1^j,\ldots,\alpha_d^j\right) \in \{0,1\}^d$. Therefore, an arbitrary formula $C \wedge G_j$ is satisfiable if and only if $C\left[\tilde{X}/\left(\alpha_1^j,\ldots,\alpha_d^j\right)\right]$ is satisfiable. Here $C\left[\tilde{X}/\left(\alpha_1^j,\ldots,\alpha_d^j\right)\right]$ is produced by setting values of variables x_{i_k} to corresponding α_k^j, $k \in \{1,\ldots,d\} : x_{i_1} = \alpha_1^j,\ldots,x_{i_d} = \alpha_d^j$. A set of CNFs

$$\Delta(C,\tilde{X}) = \left\{C\left[\tilde{X}/\left(\alpha_1^j,\ldots,\alpha_d^j\right)\right]\right\}_{\left(\alpha_1^j,\ldots,\alpha_d^j\right)\in\{0,1\}^d}$$

is called a decomposition family produced by $\tilde{X}$.

The number of CNFs in $\Delta(C,\tilde{X})$ is 2^d. It is possible to estimate the total time required for processing this family based on the time of solving of SAT problems for k, $k << 2^d$ randomly chosen CNFs from the family. This estimation can be used to plan a computational experiment.

In [23] the Monte Carlo algorithm based on tabu search heuristics to search for decomposition sets with good time estimations was suggested. This algorithm was implemented as the parallel MPI program PDSAT. We used PDSAT running on a computing cluster to obtain decomposition sets for logical cryptanalysis instances for the Bivium generator. As a result we found the set of of 47 variables for Bivium (see Fig. 2). On this figure cells corresponding to variables from decompositions sets are marked with gray color. Time estimation for this set is 1.3391e+11 seconds for 1 processor core.

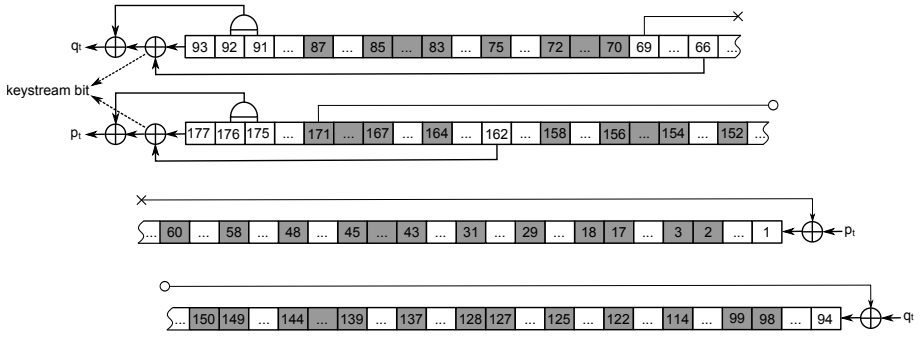

Fig. 2. Decomposition set for logical cryptanalysis of Bivium

4 Using SAT@home project for solving weakened cryptanalysis problems for Bivium

SAT@home[2] [21] is a volunteer computing BOINC-based project aimed at solving hard combinatorial problems that can be effectively reduced to SAT. It was launched

[2] http://sat.isa.ru/pdsat/

on September 29, 2011 by ISDCT SB RAS and IITP RAS. On February 7, 2012 SAT@home was added to the official list of BOINC projects[3] with alpha status. Recently its status was improved to beta.

The SAT@home server uses a number of standard BOINC daemons responsible for sending and processing tasks (transitioner, feeder, scheduler, etc.). Such daemons as work generator, validator and assimilator were implemented taking into account the specificity of the project. The work generator decomposes the original SAT problem to subproblems according to partitioning found by PDSAT (see section 3). The validator checks the correctness of the results, and the assimilator processes correct results. The client application is based on the CDCL SAT solver MINISAT [7].

The characteristics of the SAT@home project as of 22 of February 2015 are (according to BOINCstats[4]):

- 2948 active PCs (active PC in volunteer computing is a PC that sent at least one result in last 30 days) about 80% of them use Microsoft Windows OS;
- 1432 active users (active user is a user that has at least one active PC);
- versions of the client application for CPU: Windows x86, Windows x86-64, Linux x86, Linux x86-64;
- average real performance: 6 teraflops, maximal performance: 10 teraflops.

The dynamics of the real performance of SAT@home can be seen at the SAT@home performance page[5].

With respect to the estimation obtained by PDSAT the solving of one instance of cryptanalysis of the Bivium ciper in the SAT@home project with its current performance would take about 3 years (using decomposition set of 47 variables [23]). That is why we decided to solve weakened cryptanalysis problems for this generator. Below we use the notation $BiviumK$ to denote a weakened cryptanalysis problem for Bivium with known values of K variables (in corresponding SAT instance) encoding last K cells of the second shift register. In particular we considered the $Bivium9$ problem. We used PDSAT to find a decomposition set with good time estimation for $Bivium9$. As a result we obtained the decomposition set of 43 variables with time estimation 4.2873e+09 seconds. From September 2014 to December 2014 with the help of this decomposition set, 5 $Bivium9$ instances were solved in SAT@home. During the corresponding experiment for each cryptanalysis instance the search space was divided into 146602 equal tasks, each containing 60 millions subproblems. For processing one such task about 2 hours of modern CPU core is needed. The client application was based on a modified version of MINISAT 2.2. Time estimation by PDSAT shows good consistency with real solving time of these cryptanalysis instances. As far as we know there are no other available experimental results on solving such problems.

5 Related Work

The first work that used SAT-solvers for cryptanalysis was [15]. The authors of [8, 17, 26] presented some estimations of the time required for logical cryptanalysis of the

[3] http://boinc.berkeley.edu/projects.php
[4] http://boincstats.com/
[5] http://sat.isa.ru/pdsat/performance.php

Bivium cipher. Topics related to organization of SAT solving in distributed environments were considered in many papers, for example in [4, 6, 10–12, 28]. In [22] a desktop grid for solving SAT which used conflict clauses exchange via peer-to-peer protocol was described. Apparently, [3] became the first paper about the use of a desktop grid based on the BOINC platform for solving SAT, but it did not evolve into a full-fledged volunteer computing project. The predecessor of the SAT@home was the BNB-Grid system [9, 24], that was used to solve first large scale cryptographic problems in 2009.

6 Conclusions

Obtained results show that SAT@home can be successfully used for solving hard logical cryptanalysis problems. We plan to use SAT@home for solving non-weakened cryptanalysis problem of the Bivium generator. Also we plan to solve in SAT@home cryptanalysis problems for other keystream generators.

7 Acknowledgments

We thank Mikhail Posypkin and Nickolay Khrapov for their help in maintaining the SAT@home project, Stepan Kochemazov for valuable comments and discussions, and all the SAT@home volunteers for their participation. This work was partly supported by Russian Foundation for Basic Research (grants 14-07-00403-a, 14-07-31172-mol-a, 15-07-07891-a) and by the President of Russian Federation grants for young scientists (grant SP-3667.2013.5).

References

1. D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), 16-19 May 2006, Singapore*, pages 73–80. IEEE Computer Society, 2006.
2. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
3. M. Black and G. Bard. SAT Over BOINC: An Application-Independent Volunteer Grid Project. In S. Jha, N. gentschen Felde, R. Buyya, and G. Fedak, editors, *GRID*, pages 226–227. IEEE, 2011.
4. W. Blochinger, C. Sinz, and W. Küchlin. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969–994, 2003.
5. C. D. Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, editors, *ISC*, volume 4176 of *LNCS*, pages 171–186. Springer, 2006.
6. W. Chrabakh and R. Wolski. GridSAT: a system for solving satisfiability problems using a computational grid. *Parallel Computing*, 32(9):660–687, 2006.
7. N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
8. T. Eibach, E. Pilz, and G. Völkel. Attacking Bivium Using SAT Solvers. In H. K. Büning and X. Zhao, editors, *SAT*, volume 4996 of *LNCS*, pages 63–76. Springer, 2008.

9. Y. Evtushenko, M. Posypkin, and I. Sigal. A framework for parallel large-scale global optimization. *Computer Science - R&D*, 23(3-4):211–215, 2009.
10. M. Heule, O. Kullmann, S. Wieringa, and A. Biere. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In K. Eder, J. Lourenço, and O. Shehory, editors, *Haifa Verification Conference*, volume 7261 of *LNCS*, pages 50–65. Springer, 2011.
11. A. E. J. Hyvärinen. *Grid Based Propositional Satisfiability Solving*. PhD thesis, Aalto University, 2011.
12. A. E. J. Hyvärinen, T. A. Junttila, and I. Niemelä. A Distribution Method for Solving SAT in Grids. In A. Biere and C. P. Gomes, editors, *SAT*, volume 4121 of *LNCS*, pages 430–435. Springer, 2006.
13. A. E. J. Hyvärinen, T. A. Junttila, and I. Niemelä. Partitioning SAT Instances for Distributed Solving. In C. G. Fermüller and A. Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *LNCS*, pages 372–386. Springer, 2010.
14. A. E. J. Hyvärinen, T. A. Junttila, and I. Niemelä. Grid-Based SAT Solving with Iterative Partitioning and Clause Learning. In J. H.-M. Lee, editor, *CP*, volume 6876 of *LNCS*, pages 385–399. Springer, 2011.
15. F. Massacci and L. Marraro. Logical Cryptanalysis as a SAT Problem. *J. Autom. Reasoning*, 24(1/2):165–203, 2000.
16. A. Maximov and A. Biryukov. Two trivial attacks on trivium. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 36–55. Springer, 2007.
17. C. Mcdonald, C. Charnes, and J. Pieprzyk. Attacking Bivium with MiniSat. Technical Report 2007/040, ECRYPT Stream Cipher Project, 2007.
18. N. Metropolis and S. Ulam. The Monte Carlo Method. *J. Amer. statistical assoc.*, 44(247):335–341, 1949.
19. M. Nouman Durrani and J. A. Shamsi. Review: Volunteer computing: Requirements, challenges, and solutions. *J. Netw. Comput. Appl.*, 39:369–380, Mar. 2014.
20. I. Otpuschennikov, A. Semenov, and S. Kochemazov. Transalg: a tool for translating procedural descriptions of discrete functions to sat (tool paper). *CoRR*, abs/1405.1544, 2014.
21. M. Posypkin, A. Semenov, and O. Zaikin. Using BOINC desktop grid to solve large scale SAT problems. *Computer Science Journal*, 13(1):25–34, 2012.
22. S. Schulz and W. Blochinger. Parallel SAT Solving on Peer-to-Peer Desktop Grids. *J. Grid Comput.*, 8(3):443–471, 2010.
23. A. Semenov and O. Zaikin. On estimating total time to solve SAT in distributed computing environments: Application to the SAT@home project. *CoRR*, abs/1308.0761, 2013.
24. A. Semenov, O. Zaikin, D. Bespalov, and M. Posypkin. Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System. In V. Malyshkin, editor, *PaCT*, volume 6873 of *LNCS*, pages 473–483. Springer, 2011.
25. M. Soos. Grain of Salt - an Automated Way to Test Stream Ciphers through SAT Solvers. In *Tools'10: Proceedings of the Workshop on Tools for Cryptanalysis*, pages 131–144, 2010.
26. M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In O. Kullmann, editor, *SAT*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009.
27. G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.
28. H. Zhang, M. P. Bonacina, and J. Hsiang. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems. *J. Symb. Comput.*, 21(4):543–560, 1996.