Cost-Effective Architectures for RC5 Brute Force Cracking

J. Buček, J. Hlaváč, M. Matušková, R. Lórencz

In this paper, we discuss the options for brute-force cracking of the RC5 block cipher, that is, for revealing the unknown secret key, given a sample ciphertext and a portion of the corresponding plaintext. First, we summarize the methods employed by the current cracking efforts. Then, we present two hardware architectures for finding the secret key using the "brute force" method. We implement the hardware in FPGA and ASIC and, based on the results, we discuss the cost and time needed to crack the cipher using today's technology and suggest a minimum key length that can be considered secure.

Keywords: RC5 cipher, decryption, brute-force cracking, FPGA, ASIC.

1 Introduction

RC5 is a fast block cipher designed by R. Rivest [1, 2, 3]. It is a parameterized algorithm in which the block size, key length, and number of rounds are variable. The parameters are often specified in the RC5-w/r/b notation, where w is the word size in bits, $r \in [0, 255]$ is the number of rounds, and $b \in [0, 255]$ is the number of bytes in the secret key. The block size is twice the word size and the allowed values are 32 bits (recommended for experiments and testing only), 64 bits, or 128 bits. This flexibility allows the optimal level of security and efficiency to be chosen. A suggested "nominal" choice is RC5-32/12/16, that is, RC5 with 32-bit words (64-bit blocks), 12 rounds, and a 128-bit (16-byte) key.

Although RC5 is extremely simple and easy to implement, it provides a good level of security, as long as a sufficient key length and enough rounds are employed. Attempts to crack RC5 through various cryptanalysis techniques [4, 5, 6, 7, 8, 9] have been published; [6] concludes that at least 16 rounds are necessary to prevent a partial differential attack. However, all of these works consider a chosen-plaintext attack, which is not always plausible.

In 1997, RSA Security Inc. announced a "Secret-Key Challenge" [10]. The objective of the challenge is to find the secret key, given one ciphertext sample together with a portion of the corresponding plaintext. This corresponds to a real-world scenario, where an attacker could, for instance, capture packets of certain network communication and guess the contents of the encrypted packet header. Unfortunately, chosen-plaintext attack methods are of limited use in this case, and the exhaustive keyspace search method is the only known option.

In this paper, we analyze the feasibility of a brute-force attack, using general-purpose computers as well as dedicated hardware. We present the design of a fast, fully pipelined cracking engine, and compare the cost and speed with software-based solutions used by the currently employed methods. Based on the results, we suggest a minimum key length that can be considered secure against various types of attackers. These numbers are then compared to suggestions published by a group of computer scientists and cryptographers in January 1996 [11].

2 The RC5 algorithm

The RC5 algorithm is fully described and discussed in [1, 2, 3]. We include the relevant routines here for the sake of completeness.

The algorithm consists of three routines: key expansion, encryption, and decryption. It makes heavy use of data-dependent rotations, besides additions and subtractions modulo 2^w and XORs.

In the following description, + and – denote addition and subtraction modulo $2w$ respectively, $\oplus$ denotes bit-wise XOR, $A \lll B$ and $A \ggg B$ denote a rotation of A by B bits to the left or right respectively. The least-significant bit is assumed to be at the rightmost position.

2.1 Constants and variables

Besides the already-defined constants w, r, b, [1] defines the following constants to describe the algorithm: $c = \lceil b/(w/8) \rceil$ is the number of w-bit words necessary to hold the b bytes of the key, and $t = 2r + 2$ is the size of the expanded key table (array S). The key expansion routine uses the magic constants P_w and Q_w, which are defined in [1] and are derived from certain irrational numbers. Their values, for $w = 32$, are equal to (in hex):

$$P_{32} = 0\text{xB7E15163}, \; Q_{32} = 0\text{xE93779B9}.$$

The key is initially stored in the array K of b bytes. The key expansion routine uses two arrays, S and L, containing t or c words respectively, two w-bit variables A and B, and the counters i, j.

2.2 Key expansion

First, the array $L[0 \ldots c-1]$ is filled with the secret key $K[0 \ldots b-1]$. On little-endian machines, this is accomplished by zeroing-out the array L and copying the contents of K directly into the same memory area. Then, the array $S[0 \ldots t-1]$ is initialized and the secret key is mixed in:

$S[0] = P_w$;
for $i = 1$ **to** t **do**
$S[i] = S[i-1] + Q_w$;

$i = j = 0$; $A = B = 0$;
repeat $3\max(t, c)$ **times**
$A = S[i] = (S[i] + A + B) \lll 3$;
$B = L[j] = (L[j] + A + B) \lll (A + B)$;
$i = (i + 1) \bmod t$;
$j = (j + 1) \bmod c$;

2.3 Decryption

The following decryption routine assumes that the array S contains the mixed key and that the ciphertext block is in registers A and B.

for $i = r$ **downto** 1 **do**
$B = ((B - S[2i + 1]) \ggg A) \oplus A$;
$A = ((A - S[2i]) \ggg B) \oplus B$;
$B = B - S[1]$;
$A = A - S[0]$;

2.4 Encryption

Although encryption is not a primary concern of this paper, the appropriate routine is given here for completeness. The plaintext is assumed to be in A and B.

$A = A - S[0]$;
$B = B - S[1]$;
for $i = 1$ **to** r **do**
$A = ((A \oplus B) \lll B) + S[2i]$;
$B = ((B \oplus A) \lll A) + S[2i + 1]$;

3 Current cracking efforts

3.1 Distributed.net

The most well-known project aiming to crack the RC5 challenges is organized by Distributed Computing Technologies, Inc. [12]. On October 19, 1997, after 250 days of searching, distributed.net found the 56-bit secret key for the RC5-32/12/7 challenge. On July 14, 2002, they found the 64-bit secret key for the RC5-32/12/8 challenge (it took 1757 days). At the time of writing this paper (June 2004), the remaining challenges still remain to be solved.

The philosophy of the distributed.net project is to search the keyspace using the idle CPU time of many computers connected to the Internet, creating a virtual massively parallel system. Participating users download and install client software that runs in the background, searching portions of the keyspace and sending the results to the main server.

Table 1 lists some of the results published at the *distributed.net* web site [12], combined with the approximate cost of the respective CPU. The numbers pertain to the currently running RC5-72 project (RC5-32/12/9). (Note that the frequency of the AMD processors is actually their performance rating. This follows from the numbers from *distributed.net*.)

It turns out that older common processors provide a more favorable cost per Mkey/s than recent releases. In addition, Intel Pentium 4 implements the rotate instructions less efficiently than the other CPUs, attaining the least favorable results out of those presented.

Table 1: General-purpose CPU results

CPU	Rate (Mkeys/s)	Cost (USD)	Cost/Rate
AMD Athlon 64 (3.4 GHz)	8.4	245	29.2
AMD Athlon XP (3 GHz)	7.7	163	21.1
Intel Pentium 4 (3.2 GHz)	4.5	237	52.7
PowerPC G4 (1.25 GHz)	13.1	399	30.5

3.2 Distributed reconfigurable hardware

A recent work by Morrison et al [13] attempts to crack RC5 using a cluster of PC compatible computers with FPGA accelerator boards. A rate of 168 kkeys/s for a Pentium 4 and 1.7 Mkeys/s per node of their distributed system is cited. These numbers appear rather low in comparison with the *distributed.net* [12] results; unfortunately, it is not clear what is the cause of the mismatch.

4 Hardware cracking engine

We have designed two different architectures of dedicated hardware (one of them was presented in [14]) for brute-force cracking of the RC5 cipher (any of its variants with $w = 32$). To attain the maximum speed possible, the design is not reconfigurable - the parameters of the RC5 variant, which is to be cracked, must be specified at design (synthesis) time.

4.1 Pipelined architecture

The first design utilizes a long pipeline in order to attain a high clock rate and effectively check one key every clock cycle. The number of pipeline stages is equal to $3\max(t, c) + r$.

The overall block diagram of our hardware design is shown in Figure 1. The inputs are "SECRET KEY" – the

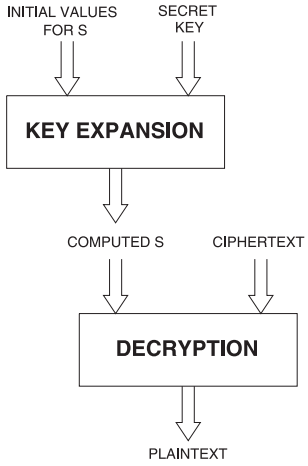

Fig. 1: Block diagram

key which is to be tested, "INITIAL VALUES FOR S" – the contents of the S array detailed in section 2.2, and "CIPHERTEXT" – the one-block sample of the encoded message, to which the corresponding plaintext block is known. Hardware for generating individual keys and for comparing the computed plaintext to the expected value is omitted from the figure.

The implementation of the key expansion pipeline ("KEY EXPANSION" block in Fig. 1) in our design is shown in Fig. 2. Again, its inputs are the initial contents of the array S (constants calculated at design time using the values P_w and Q_w) and the secret key to be tested. The dash-and-dotted line represents registers (fields of the array S) that would still contain original initialization values (which are known at design time) and therefore are not physically implemented.

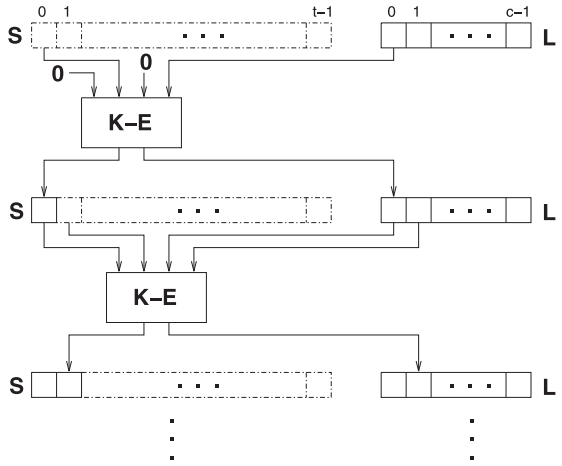

Fig. 2: Key expansion pipeline

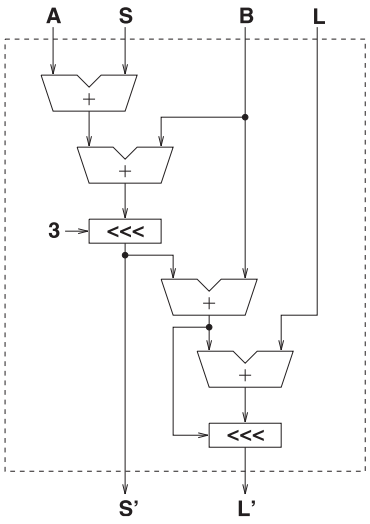

Fig. 3: Key expansion element

The structure of one key expansion element (K-E in Fig.2) is shown in Fig. 3. It performs the operations that are inside the body of the main loop described in section 2.2.

The structure of the decryption pipeline (the "DECRYPTION" block in Fig. 1) is shown in Fig. 4. It performs the functions described by the algorithm in section 2.3. Its inputs are the computed array S (with the secret key mixed in) and the ciphertext block. The dotted lines represent registers (portions of the array S) that are no longer needed for the calculation and thus are not physically implemented. The structure of one decryption round element is shown in Fig. 5. It performs the operations inside the main loop detailed in section 2.3.

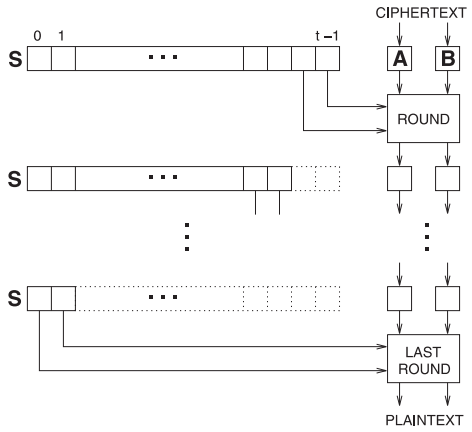

Fig. 4: Decryption pipeline

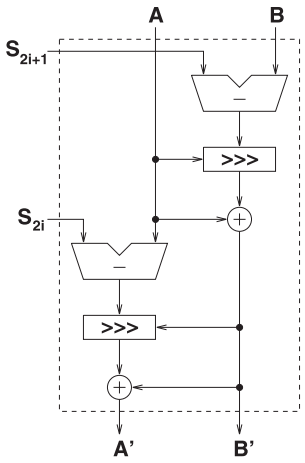

Fig. 5: Decryption round element

The final round (not included in r) differs from the others and its hardware structure is shown in Fig. 6. To give the reader an idea about the size of the entire hardware, let us take the RC5-32/12/9 variant as an example and count the number of hardware elements.

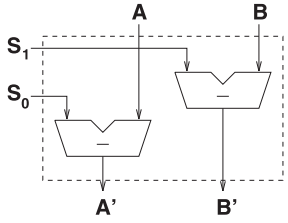

Fig. 6: Final decryption round

Every stage of the key expansion pipeline contains 3 registers (L array), up to 26 registers (S array), 4 adders, 1 barrel shifter. Every stage of the decryption pipeline contains up to

26 registers (S array), 2 registers (A, B), 2 subtractors, 2 XORs, 2 barrel shifters. The last round element contains 2 subtractors. Thus, the entire pipeline would contain 1906 w-bit registers, 312 adders, 26 subtractors, 24 XORs and 102 barrel shifters.

4.2 Sequential (not pipelined) architecture

As stated in section 4.1 above, the pipelined architecture is resource hungry. We anticipate that we would need a large FPGA (or a large ASIC) to implement the full pipeline. However, there is a class of cheap FPGAs (e.g. the Xilinx Spartan-3 series), which could be also utilized, if we can design such a circuit that fits into these small-to-medium sized devices.

We have therefore designed another variant of the cracking engine, which is sequential but not pipelined. (In fact, it still has two stages, but it is not fully pipelined, so we call it sequential here.) The basic block diagram stays the same (see Figure 1). The key expansion and decryption units have been changed so that the computation is performed sequentially, much like the specification of the original algorithm (sections 2.2 and 2.3). The key expansion and decryption units contain only one key expansion element and one decryption round element, respectively.

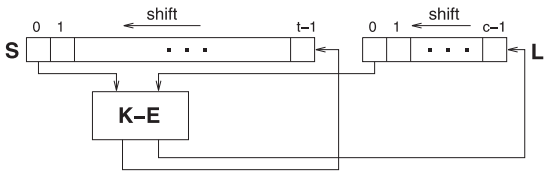

Fig. 7: Sequential key expansion unit

The structure of the sequential key expansion unit is shown in Fig. 7. The main difference from the pipelined unit (Fig. 2) is that the S and L arrays are now implemented as shift registers and the A and B registers are now inside the key expansion element.

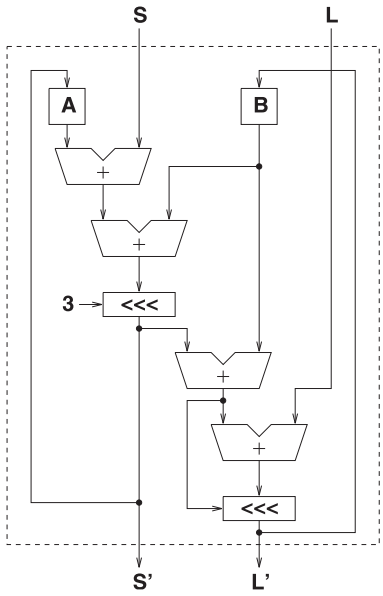

Fig. 8: Sequential key expansion element

The S and L registers shift by one word to the lower indices (to the left in the figure) in order to implement the key expansion sequence. The number of cycles is the same as the number of pipeline stages in the pipelined version, i.e. $3\max(t, c) + r$, and the end of key expansion is determined by a counter (not shown here for simplicity). The unit now contains only one key expansion element, whose structure is shown in Fig. 8. It performs the operations that are inside the body of the main loop described in section 2.2, and contains the A and B registers.

The structure of the sequential decryption unit is shown in Fig. 9. It performs the functions described by the algorithm in section 2.3. The S register is now implemented as a shift register, but this time it shifts by two words toward the higher indices (to the right in the figure). The structure of one decryption round element is shown in Fig. 10. It performs the operations inside the main loop detailed in section 2.3. The last round is now treated differently, and the final decrypted output is now taken directly from the decryption round element (outputs A, and B, in Fig. 10). The end of decryption is determined by a counter (not shown here for simplicity).

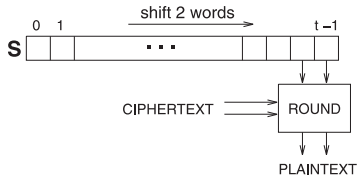

Fig. 9: Sequential decryption unit

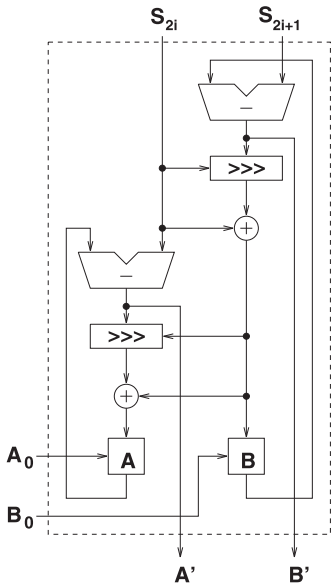

Fig. 10: Sequential decryption round element

5 Results

We described both hardware architectures in VHDL, simulated it and synthesized it for FPGA. We synthesized the pipelined architecture also for ASIC. The following experiments were conducted with the RC5-32/12/9 variant to make the results directly comparable with those in section 3.1 (*distributed.net*).

5.1 FPGA implementation

The pipelined hardware is rather complicated. Furthermore, barrel shifters are not "FPGA-friendly" due to high requirements on routing resources. As a result, a rather large FPGA is necessary to implement the entire design, e.g. a Xilinx xc2vp100 (Virtex II Pro) or xc2v8000 (Virtex II). It is clear that FPGA is not a suitable platform for the pipelined architecture, compared to general-purpose CPUs (Table 1).

The sequential hardware is much smaller than our pipelined implementation. The circuit now fits into cheap, small- to medium-sized FPGAs such as Xilinx xc3s200 and xc3s400 (Spartan-3). The downside is the substantial drop in the decryption rate. However, as we can see, the lower price compensates for the speed decrease. We can see that the sequential FPGA implementation is comparable with general-purpose CPUs, from the Cost/Rate point of view. The results of both variants implemented in FPGA are listed in Table 2.

Table 2: FPGA results

Device	Architecture	Rate [Mkeys/s]	Cost [USD]	Cost/Rate
xc2vp100	Pipelined	39	7900	202
xc2v8000	Pipelined	43	11000	256
xc3s200	Sequential	0.66	13.45	20.4
xc3s400	Sequential	1.32	22	16.7

5.2 ASIC Implementation

We synthesized the pipelined design using a 0.18 μm technology library (unfortunately, we did not have the necessary data for synthesis into the latest 90 nm and 60 nm technologies). Out of the wide range of tradeoffs between speed and area, one of the favorable settings resulted in a design that occupies approximately 15 mm^2 and can run at 215 MHz under typical conditions. Several instances of the design can be placed on one die to further reduce the cost.

For ASICs, it is difficult to obtain an exact price per chip since it greatly depends, among other factors, on the total volume of chips produced. We tried to get a reasonable estimate for a small quantity of chips (based on MOSIS price lists [15] for a lot of 40 chips) and for a large quantity (based on the market price of common ICs with a similar die size). Table 3 lists our results and the estimated prices.

While it would be possible to fine-tune and optimize the synthesis process or implement the design in more advanced technology for even better results, the obtained figures are sufficient for comparison with the other platforms considered in this paper.

Table 3: ASIC Results

Production type	Rate/chip (Mkeys/s)	Cost/chip (USD)	Cost/Rate
Low volume, small die	215	1050	4.9
Low volume, large die	2150	4500	2.1
High volume, large die	2150	180	0.08

5.3 Minimum secure key length

Concluding from the previous sections, it is clear that FPGA is not a reasonable option for the pipelined architecture since it is too expensive. On the other hand, the sequential architecture can be used with smaller and cheaper FPGAs. A low-budget attacker would turn to systems built upon general-purpose CPUs or cheap FPGAs, while a well-funded attacker is likely to use ASICs.

The Cost/Rate ratio of small FPGAs is comparable or even superior to that of general purpose CPUs. However, when considering whether to build a FPGA-based cracking engine or a CPU-based one, we have to take into account additional costs that were not addressed in this paper. These involve the costs of custom built or off-the-shelf mainboards, input/output processing circuits, etc. In this respect, the CPU-based cracking engine would probably be better.

To find the unknown secret key using a brute-force attack, an attacker must search 50% of the keyspace (2^{8b-1} keys) on average, and the entire keyspace (2^{8b} keys) in the worst case. Table 4 details the number of keys an attacker would be able to try every second, based on the amount invested, using the technology discussed in the preceding sections.

Table 4: Speed vs. investment

Investment	Technology	Keys per		
		sec	day	year
$5k	GP CPU or FPGA	2^{28}	2^{44}	2^{53}
$50k	GP CPU or FPGA	2^{31}	2^{47}	2^{56}
	ASIC	2^{33}	2^{49}	2^{58}
$500k	ASIC	2^{38}	2^{54}	2^{63}
$5M	ASIC	2^{46}	2^{62}	2^{70}

From the data in Table 4, we can conclude that 40-bit keys are not secure at all, and 56-bit keys can only protect sensitive data from low-budget attackers for a limited period of time. Should we wish to protect sensitive data against the strongest attacker listed in Table 4, the secret key should be longer than 70 bits. However, considering that more advanced technology is available today than the discussed 0.18 μm process, we would recommend at least 80 to 90 bits. 128-bit keys that are common in today's cryptographic systems satisfy this condition.

Another view is that, for instance, a 54-bit key is inadequate, if an attacker can gain more than $500,000 by recovering it in a day.

It should be noted that while the cost of brute-force attack increases exponentially with the length of the secret key, the cost of encryption and decryption increases negligibly. Therefore, a prudent cryptographic system would use a key that is at least twice or thrice as long as the recommended minimum, to allow a margin for error or technology improvements [11].

6 Conclusion

We have designed two hardware architectures for attacking the RC5 cipher using the exhaustive search (also known as "brute-force") method. We have described the architectures in VHDL, synthesized it for various FPGA and ASIC platforms and compared the speed with common general-purpose processors.

It turns out that FPGAs are comparable to general-purpose CPUs, although the additional costs may favor using CPU-based systems. For a low-budget attacker, today's general-purpose CPUs provide better performance. For an attacker with a generous budget, ASIC is the best option.

Today, an 80 to 90-bit key seems to be secure enough against most attackers. For data that should remain protected for some time to come, at least 128 bits should be used; however, a prudent cryptographer would use a much longer key – the increased cost of encryption and decryption is negligible, while a brute-force attack becomes impossible.

Our future work will include testing the sequental design in a physical implementation in a small or medium-sized FPGA.

References

[1] Rivest, R. L.: The RC5 Encryption Algorithm. *CryptoBytes*, Vol. **1** (1995), No. 1, p. 9–11.

[2] Baldwin, R., Rivest, R.: The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms. RFC 2040, Network Working Group, 1996.

[3] Rivest, R. L.: Block Encryption Algorithm with Data-Dependent Rotations. U.S. Patents No. 5,724,428 and No. 5,835,600, 1998.

[4] Kaliski, B. S. Jr., Yin, Y. L.: "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm." Advances in Cryptology – Crypto'95, Springer-Verlag, 1995, p. 171–183.

[5] Knudsen, L. R., Meier, W.: "Improved Differential Attacks on RC5." Advances in Cryptology – Crypto'96, Springer-Verlag, 1996, p. 216–228.

[6] Biryukov, A., Kushilevitz, E.: "Improved Cryptanalysis of RC5." Advances in Cryptology – Eurocrypt'98, Springer-Verlag, 1998.

[7] Selcuk, A. A.: "New Results in Linear Cryptanalysis of RC5." Proceedings of 5th International Workshop on Fast Software Encryption, Springer-Verlag, 1998, p. 1–16.

[8] Shimoyama, T., Takeuchi K., Hayakawa J.: "Correlation Attack to the Block Cipher RC5 and the Simplified Variants of RC6." Proceedings AES3, New York, 2001.

[9] Yin Y. L.: "The RC5 Encryption Algorithm: Two Years On," *CryptoBytes*, Vol. **2** (1997), No. 3, p. 14–15.

[10] RSA Data Security, Inc.: Secret-Key Challenges. On-line at http://www.rsasecurity.com/rsalabs/chalenges/secretkey/index.html

[11] Blaze, M., Diffie, W., Rivest, R. L. et al.: "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security." A Report by an Ad Hoc Group of Cryptographers and Computer Scientists, 1996. On-line at http://theory.lcs.mit.edu/rivest/bsa-final-report.pdf

[12] Distributed Computing Technologies, Inc.: The "distributed.net" project. On-line at http://www.distributed.net

[13] Morrison, J. P., O'Dowd, P. J., Healy, P. D.: "Searching RC5 Keyspaces with Distributed Reconfigurable Hardware." In: Engineering of Reconfigurable Systems and Algorithms, CSREA Press, 2003, p. 269–272.

[14] Matušková, M., Hlaváč, J., Buček, J., Lórencz, R.: "RC5 Brute Force Cracking Engine." Proceedings of the Sixth International Scientific Conference Electronic Computers and Informatics ECI 2004. Technical University Košice, 2004, p. 259–264. ISBN 80-8073-150-0.

[15] MOSIS IC Fabrication Service. On-line at http://www.mosis.org

Ing. Jiří Buček
e-mail: bucekj@fel.cvut.cz

Ing. Josef Hlaváč
e-mail: hlavacj2@fel.cvut.cz

Ing. Róbert Lórencz, CSc.
e-mail: lorencz@fel.cvut.cz

Department of Computer Science and Engineering

Czech Technical University in Prague
Faculty of Electrical Engineering
Karlovo nám. 13
121 35 Prague, Czech Republic

Ing. Monika Matušková
e-mail: monika.matuskova@vslib.cz

Department of Software Engineering

Technical University of Liberec
Faculty of Mechatronics
Hálkova 6
460 15 Liberec, Czech Republic

