

# MV3: A new word based stream cipher using rapid mixing and revolving buffers<sup>\*</sup>

Nathan Keller<sup>1\*\*</sup>, Stephen D. Miller<sup>2\*\*\*</sup>, Ilya Mironov<sup>3</sup>, and Ramarathnam Venkatesan<sup>4</sup>

<sup>1</sup> Einstein Institute of Mathematics, The Hebrew University, Givat Ram, Jerusalem 91904, Israel  
`nkeller@math.huji.ac.il`

<sup>2</sup> Department of Mathematics, Rutgers University, Piscataway, NJ 08854  
`miller@math.rutgers.edu`

<sup>3</sup> Microsoft Research, Silicon Valley Campus, 1065 La Avenida, Mountain View, CA 94043  
`mironov@microsoft.com`

<sup>4</sup> Cryptography and Anti-piracy Group, Microsoft Research, 1 Microsoft Way, Redmond, WA 98052  
and  
Cryptography, Security and Algorithms Research Group, Microsoft Research India  
Scientia - 196/36 2nd Main, Sadashivnagar, Bangalore 560 080, India  
`venkie@microsoft.com`

**Abstract.** MV3 is a new *word based* stream cipher for encrypting long streams of data. A direct adaptation of a byte based cipher such as RC4 into a 32- or 64-bit word version will obviously need vast amounts of memory. This scaling issue necessitates a look for new components and principles, as well as mathematical analysis to justify their use. Our approach, like RC4's, is based on rapidly mixing random walks on directed graphs (that is, walks which reach a random state quickly, from any starting point). We begin with some well understood walks, and then introduce nonlinearity in their steps in order to improve security and show long term statistical correlations are negligible. To minimize the short term correlations, as well as to deter attacks using equations involving successive outputs, we provide a method for sequencing the outputs derived from the walk using three revolving buffers. The cipher is fast — it runs at a speed of 4.8 cycles per byte on a Pentium IV processor. A word based cipher needs to output more bits per step, which exposes more correlations for attacks. Moreover we seek simplicity of construction and transparent analysis. To meet these requirements, we use a larger state and claim security corresponding to only a fraction of it. Our design is for an adequately secure word-based cipher; our very preliminary estimate puts the security close to exhaustive search for keys of size  $\leq 256$  bits.

Keywords: stream cipher, random walks, expander graph, cryptanalysis.

## 1 Introduction

Stream ciphers are widely used and essential in practical cryptography. Most are custom designed, e.g. alleged RC4 [Sch95, Ch. 16], SEAL [RC98], SCREAM [HCJ02], and LFSR-based NESSIE submissions such as LILI-128, SNOW, and SOBER [P+03, Ch. 3]. The VRA cipher [ARV95] has many provable properties, but requires more memory than the rest. We propose some new components and principles for stream cipher design, as well as their mathematical analysis, and present a concrete stream cipher called MV3.

To motivate our construction, we begin by considering RC4 in detail. It is an exceptionally short, byte-based algorithm that uses only 256 bytes of memory. It is based on random walks (card shuffles), and has no serious attacks. Modern personal computers are evolving from 32 to 64 bit words, while a growing number of smaller devices have different constraints on their word and memory sizes. Thus one may desire ciphers better suited to their architectures, and seek designs that scale nicely across these sizes. Here we focus on scaling up such random walk based ciphers. Clearly, a direct adaptation of RC4 would require vast amounts of memory.

---

\* This is a condensed version of the papers [MV3,Full MV3], created for submission to SASC 2007.

\*\* Partially supported by the Adams Fellowship.

\*\*\* Partially supported by NSF grant DMS-0301172 and an Alfred P. Sloan Foundation Fellowship.

The security properties of most stream ciphers are not based on some hard problem (e.g., as RSA is based on factoring). One would expect this to be the case in the foreseeable future. Nevertheless, they use components that – to varying degrees – are analyzable in some idealized sense. This analysis typically involves simple statistical parameters such as cycle length and mixing time. For example, one idealizes each iteration of the main loop of RC4 as a step in a random walk over its state space. This can be modeled by a graph  $G$  with nodes consisting of  $S_{256}$ , the permutations on 256 objects, and edges connecting nodes that differ by a transposition. Thus far no serious deviations from the random walk assumptions are known. Since storing an element of  $S_{232}$  or  $S_{264}$  is out of the question, one may try simulations using smaller permutations; however, this is nontrivial if we desire both competitive speeds and a clear analysis. It therefore is attractive to consider other options for the underlying graph  $G$ .

One of the most important parameters of RC4 is its mixing time. This denotes the number of steps one needs to start from an arbitrary state and achieve uniform distribution over the state space through a sequence of independent random moves. This parameter is typically not easy to determine. Moreover, RC4 keeps a loop counter that is incremented modulo 256, which introduces a memory over 256 steps. Thus its steps are not even Markovian (where a move from the current state is independent of earlier ones). Nevertheless, the independence of moves has been a helpful idealization (perhaps similar to Shannon’s random permutation model for block ciphers), which we will also adhere to.

We identify and focus on the following problems:

- **Problem 1 – Graph Design.** How to design graphs whose random walks are suitable for stream ciphers that work on arbitrary word sizes.
- **Problem 2 – Extraction.** How to extract bits to output from (the labels of) the nodes visited by walk.
- **Problem 3 – Sequencing.** How to sequence the nodes visited by the walk so as to diminish any attacks that use relationships (e.g. equations) between successive outputs.

We now expand on these issues. At the outset, it is important to point out the desirability of simple register operations, such as additions, multiplications, shifts, and XOR’s. These are crucial for fast implementation, and preclude us from using many existing constructions of expander graphs (such as those in [LPS86,HLW06]). Thus part of the cipher design involves new mathematical proofs and constructions. The presentation of the cipher does not require these details, which may be found in the appendices of [Full MV3].

**High level Design Principles:** Clearly, a word based cipher has to output more bits per step of the algorithm. But this exposes more relationships on the output sequence, and to mitigate its effect we increase the state size and aim at security that is only a fraction of the log of the state size. We also tried to keep our analysis as transparent and construction as simple as possible. Our key initialization is a bit bulky and in some applications may require further simplifications, a topic for future research.

## 1.1 Graph Design: Statistical properties and Non-linearities

In the graph design, one wants to keep the mixing time  $\tau$  small as a way to keep the long term correlations negligible. This is because many important properties are guaranteed for walks that are longer than  $\tau$ . For example, such a walk visits any given set  $S$  nearly the expected number of times, with exponentially small deviations (a theorem of [Gi98]). A corollary of this fact is that each output bit is unbiased.

Thus one desires the optimal mixing time, which is on the order of  $\log N$ ,  $N$  being the size of the underlying state space. Graphs with this property have been well studied, but the requirements for stream ciphers are more complicated, and we are not aware of any work that focuses on this issue. For example, the graphs whose nodes are  $\mathbb{Z}/2^n\mathbb{Z}$  (respectively  $(\mathbb{Z}/2^n\mathbb{Z})^*$ ) and edges are  $(x, x + g_i)$  (respectively  $(x, x \cdot g_i)$ ), where  $g_i$  are randomly chosen and  $i = O(n)$ , have this property [AR94]. While these graphs are clearly very efficient to implement, their commutative operations are quite linear and hence the attacks mentioned in Problem 3 above can be effective.

To this end, we introduce some nonlinearities into our graphs. For example, in the graph on  $\mathbb{Z}/2^n\mathbb{Z}$  from the previous paragraph, we can also add edges of the form  $(x, hx)$  or  $(x, x^r)$ . This intuitively allows for more randomness, as well as disrupting relations between successive outputs. However, one still needs to prove that the mixing time of such a modified graph is still small. Typically this type of analysis is hard to come by, and in fact was previously believed to be false. However, we are able to give rigorous proofs in some

cases, and empirically found the numerical evidence to be stronger yet in the other cases. More details can be found in the appendices of [Full MV3].

Mixing up the random walks on multiplicative and additive abelian groups offers a principled way to combine with nonlinearities for an effective defense. As a practical matter, it is necessary to ensure that our (asymptotic) analysis applies when parameters are small, which we have verified experimentally.

We remark here that introduction of nonlinearities was the main motivation behind the construction of the  $T$ -functions of Klimov and Shamir ([KS02]). They showed that the walk generated by a  $T$ -function deterministically visits every  $n$ -bit number once before repeating. A random walk does not go through all the nodes in the graph, but the probability that it returns to a previous node in  $m$  steps tends to the uniform probability at a rate that drops exponentially in  $m$ . It also allows us to analyze the statistical properties as indicated above. (See Appendix A of [Full MV3] for more background.)

## 1.2 Extraction

Obviously, if the nodes are visited truly randomly, one can simply output the LSB's of the node, and extraction is trivial. But when there are correlations among them, one can base an attack on studying equations involving successive outputs. One solution to this problem is to simultaneously hash a number of successive nodes using a suitable hashing function, but this will be expensive since the hash function has to work on very long inputs.

Our solution to the sequencing problem below allows us to instead hash a linear combination of the nodes in a faster way. A new aspect of our construction is that our hash function itself evolves on a random walk principle. We apply suitable rotations on the node labels (to alter the internal states) at the extraction step to ensure the top and bottom half of the words mix well.

## 1.3 Sequencing

As we just mentioned, the sequencing problem becomes significant if we wish to hash more bits to the output (in comparison to RC4). First we ensure that our graph is directed and has no short cycles. But this by itself is insufficient, since nodes visited at steps in an interval  $[t, t + \Delta]$ , where  $\Delta \ll \tau$ , can have strong correlations. Also, we wish to maximize the number of terms required in equations involved in the attacks mentioned in Problem 3. To this end, we store a short sequence of nodes visited by the walk in buffers, and sequence them properly. The buffers ensure that any relation among output bits is translated to a relation involving many nonconsecutive bits of the internal state. Hence, such relations cannot be used to mount efficient attacks on the internal state of the cipher.

The study of such designs appear to be of independent interest. We are able to justify their reduction of correlations via a theorem of [CHJ02] (see Section 4.5).

## 1.4 Analysis and Performance

We do not have a full analysis of the exact cipher that is implemented. However, we have ensured that our idealizations are in line with the ones that allow RC4 be viewed via random walks. Of course some degree of idealization is necessary because random bits are required to implement any random walk; here our design resembles that of alleged RC4 [Sch95, Ch. 16]. Likewise, our cipher involves combining steps from different, independent random walks on the same underlying graph. We are able to separately analyze these processes, but although combining such steps should intuitively only enhance randomness, our exact mathematical models hold only for these separate components and hence we performed numerical tests as well.

Our cipher MV3 is fast on 32 bit processors — it runs at a speed of 4.8 cycles a byte on Pentium IV, while the speed of RC4 is about 10 cycles a byte. Only two of the eSTREAM candidates [DC06] are faster on similar architecture.

We evaluated it against some known attacks and we present the details in Section 4. We note that some of the guess-and-determine attacks against RC4 (e.g. [K+98]) are also applicable against MV3. However, the large size of the internal state of MV3 makes these attacks much slower than exhaustive key search, even for very long keys.

The security claim of MV3 is that no attack faster than exhaustive key search can be mounted for keys of length up to 256 bits.<sup>5</sup>

The paper is organized as follows: In Section 2 we give a description of MV3. Section 3 contains the design rationale of the cipher. In Section 4 we examine the security of MV3 with respect to various methods of cryptanalysis. Finally, Section 5 summarizes the paper.

## 2 The Cipher MV3

In this section we describe the cipher algorithm and its basic ingredients. The letters in its name stand for “multi-vector”, and the number refers to the three revolving buffers that the cipher is based upon.

**Internal state.** The main components of the internal state of MV3 are three revolving buffers  $A$ ,  $B$ , and  $C$  of length 32 double words (unsigned 32-bit integers) each and a table  $T$  that consists of 256 double words. Additionally, there are publicly known indices  $i$  and  $u$  ( $i \in [0 \dots 31]$ ,  $u \in [0 \dots 255]$ ), and secret indices  $j$ ,  $c$ , and  $x$  ( $c, x$  are double words,  $j$  is an unsigned byte).

Every 32 steps the buffers shift to the left:  $A \leftarrow B$ ,  $B \leftarrow C$ , and  $C$  is emptied. In code, only the pointers get reassigned (hence the name “revolving”, since the buffers are circularly rotated).

**Updates.** The internal state of the cipher gets constantly updated by means of pseudo-random walks. Table  $T$  gets refreshed one entry every 32 steps, via application of the following two operations:

$$\begin{aligned} u &\leftarrow u + 1 \\ T[u] &\leftarrow T[u] + (T[j] \ggg 13). \end{aligned}$$

(Symbol  $x \ggg a$  means a circular rotation to the right of the double word  $x$  by  $a$  bits).

In other words, the  $u$ -th element of the table, where  $u$  sweeps through the table in a round-robin fashion, gets updated using  $T[j]$ .

In its turn, index  $j$  walks (in every step, which can be idealized as a random walk) as follows:

$$j \leftarrow j + (B[i] \bmod 256),$$

where  $i$  is the index of the loop. Index  $j$  is also used to update  $x$ :

$$x \leftarrow x + T[j],$$

which is used to fill buffer  $C$  by  $C[i] \leftarrow (x \ggg 8)$ .

Also, every 32 steps the multiplier  $c$  is additively and multiplicatively refreshed as follows:

$$\begin{aligned} c &\leftarrow c + (A[0] \ggg 16) \\ c &\leftarrow c \vee 1 \\ c &\leftarrow c^2 \quad (\text{can be replaced by } c \leftarrow c^3) \end{aligned}$$

**Main loop.** The last ingredient of the cipher (except for the key setup) is the instruction for producing the output. This instruction takes the following form:

$$\text{output:} \quad (x \cdot c) \oplus A[9i + 5] \oplus (B[7i + 18] \ggg 16).$$

The product  $x \cdot c$  of two 32-bit numbers is taken modulo  $2^{32}$ .

Putting it all together, the main loop of the cipher is the following:

```

Input: length  $len$ 
Output: stream of length  $len$ 
  repeat  $len/32$  times
    for  $i = 0$  to  $31$ 
       $j \leftarrow j + (B[i] \bmod 256)$ 

```

<sup>5</sup> Note that MV3 supports various key sizes of up to 8192 bits. However, the security claims are only for keys of size up to 256 bits.

```

     $x \leftarrow x + T[j]$ 
     $C[i] \leftarrow (x \ggg 8)$ 
    output  $(x \cdot c) \oplus A[9i + 5] \oplus (B[7i + 18] \ggg 16)$ 
  end for
   $u \leftarrow u + 1$ 
   $T[u] \leftarrow T[u] + (T[j] \ggg 13)$ 
   $c \leftarrow c + (A[0] \ggg 16)$ 
   $c \leftarrow c \vee 1$ 
   $c \leftarrow c^2$  (can be replaced by  $c \leftarrow c^3$ )
   $A \leftarrow B, B \leftarrow C$ 
end repeat

```

### Key initialization.

The key initialization algorithm accepts as inputs a key  $K$  of length  $keylength$ , which can be any multiple of 32 less than or equal to 8192 (we recommend at least 96 bits), and an initial vector  $IV$  of the same length as the key. The key remains the same throughout the entire encryption session, though the initial vector changes occasionally. The initial vector is publicly known, but should not be easily predictable. For example, it is possible to start with a “random”  $IV$  using a (possibly insecure) pseudo-random number generator known to the attacker, and then increment the  $IV$  by 1 every time (see Section 4.2).

The key initialization algorithm is the following:

```

Input: key  $key$  and initial vector  $IV$ , both of length  $keylength$  double words
Output: internal state that depends on the key and the  $IV$ 
   $j, x, u \leftarrow 0$ 
   $c \leftarrow 1$ 
  fill  $A, B, C, T$  with  $0xEF$ 
  for  $i = 0$  to 3
    for  $l = 0$  to 255
       $T[i + l] \leftarrow T[i + l] + (key[l \bmod keylength] \ggg 8i) + l$ .
    end for
    produce 1024 bytes of MV3 output
    encrypt  $T$  with the resulting key stream
  end for
  for  $i = 4$  to 7
    for  $l = 0$  to 255
       $T[i + l] \leftarrow T[i + l] + (IV[l \bmod keylength] \ggg 8i) + l$ .
    end for
    produce 1024 bytes of MV3 output
    encrypt  $T$  with the resulting key stream
  end for

```

Note that when only the  $IV$  is changed, only the second half of the key initialization is performed.

## 3 Design Rationale

In this section we describe more of the motivating principles behind the new cipher.

**Internal state.** The internal state of the cipher has a huge size of more than 11,000 bits. This makes guess-and-determine attacks on it (like the attack against RC4 in [K+98]) much slower than exhaustive key search, even for very long keys. In addition, it also secures the cipher from time/memory tradeoff attacks trying to invert the function  $f : State \rightarrow Output$ , even for large key sizes. More detail on the security of the cipher with respect to these attacks appears in Section 4.

The buffers  $A, B, C$  and table  $T$ , as well as the indices  $j, c$ , and  $x$  should never be exposed. Since the key stream is available to the attacker and depends on this secret information, the cipher strictly adheres to the following design principles:

**Principle 1.** Output words must depend on as many secret words as possible.

**Principle 2.** Retire information faster than the adversary can exploit it.

As the main vehicle towards these goals, we use random walks (or, more precisely, pseudo-random walks, as the cipher is fully deterministic).

**Updates.** The updates of the internal state are based on several simultaneously applied random walks. On the one hand, these updates are very simple and can be efficiently implemented. On the other hand, as shown in Appendix A of [Full MV3], the update mechanism allows one to mathematically prove some randomness properties of the sequence of internal states. Note that the random walks are interleaved, and the randomness of each one of them relies on the randomness of the others. Note also that the updates use addition in  $\mathbb{Z}/2^n\mathbb{Z}$  and not a bitwise XOR operation. This partially resolves the problem of high-probability short correlations in random walks: In an undirected random walk, there is a high probability that after a short number of steps the state returns to a previous state, while in a directed random walk this phenomenon does not exist. For example, if we would use an update rule  $x \leftarrow x \oplus T[j]$ , then with probability  $2^{-8}$  (rather than the trivial  $2^{-32}$ )  $x$  would return to the same value after two steps. The usage of addition, which unlike XOR is not an involution, prevents this property. However, in the security proof for the idealized model we use the undirected case, since the known proofs of rapid mixing (like the theorem of Alon and Roichman [AR94]) refer to that case.

**Introducing nonlinearity.** In order to introduce some nonlinearity we use a multiplier  $c$  that affects the cipher output in a multiplicative way. The value of  $c$  is updated using an expander graph which involves both addition and multiplication, as explained in Appendix A of [Full MV3]. It is far from clear the squaring or cubing operation still leaves the mixing time small and our theorem addresses this.

Our update of  $c$  involves a step  $c \leftarrow c \vee 1$ . This operation may at a first seem odd, since it leaks  $\text{LSB}(c)$  to attacker, who may use it for a distinguishing attack based only on the LSB of outputs, ignoring  $c$  entirely. However, this operation is essential, since otherwise the attacker can exploit cases where  $c = 0$ , which occur with a relatively high probability of  $2^{-16}$  due to the  $c \leftarrow c^2$  operation (and last for 32 steps at a time). In this situation, she can disregard the term  $x \cdot c$  and devise a guess-and-determine attack with a much lower time complexity than the currently possible one.

**Sequencing rule.** The goals of this step were explained in section 1.3. Our output rule is based on the following general structure: The underlying walk  $x_0, x_1, \dots, x_n, \dots$  is transformed into the output  $y_0, y_1, \dots, y_n, \dots$  via a linear transformation:

$$y_i = x_{n_{i1}} \oplus x_{n_{i2}} \oplus \dots \oplus x_{n_{ik}}.$$

Without loss of generality, we assume that the indices are sorted  $n_{i1} < n_{i2} < \dots < n_{ik}$ . Let  $\mathcal{N} = \{n_{ij}\}$ . The set  $\mathcal{N}$  is chosen to optimize the following parameters:

1. Minimize the latency and the buffer size required to compute  $y_i$ . To this end, we require that there will be two constants  $m$  and  $C$ , between 64 and 256, such that  $i - C \leq n_{ij} \leq i$  for each  $i \geq m$  and  $1 \leq j \leq k$ . We additionally constrain  $n_{ik} = i$  for all  $i > m$ ;
2. Maximize the minimal size of a set of pairs  $x_i, x_{i+1}$  that can be expressed as a linear combination of  $y$ 's. More precisely, we seek to maximize  $a$  such that the following holds for some  $j_1, \dots, j_b > m$  and  $i_1, \dots, i_a$ :

$$(x_{i_1} \oplus x_{i_1+1}) \oplus (x_{i_2} \oplus x_{i_2+1}) \oplus \dots \oplus (x_{i_a} \oplus x_{i_a+1}) = y_{j_1} \oplus y_{j_2} \oplus \dots \oplus y_{j_b}. \quad (3.1)$$

Notice that the value of  $b$  has not been constrained, since usually this value is not too high and the attacker can obtain the required data.

Intuitively speaking, the second constraint ensures that if the smallest feasible  $a$  is large enough, no linear properties of the  $x$  walk propagate to the  $y$  walk. Indeed, any linear function on the  $y$  walk can be expressed as a function on the  $x$  walk. Since the  $x$  walk is memoryless, any linear function on a subset of  $x$ 's can be written as a XOR of linear functions on the intervals of the walk. Each such interval can in turn be broken down as a sum of pairs. If  $a$  is large enough, no linear function can be a good distinguisher. Note that we concentrate on the relation between consecutive values of the state  $x$ , since in a directed random walk such pairs of states seem to be the most correlated ones.

Constructing the set  $\mathcal{N}$  can be greatly simplified if  $\mathcal{N}$  has periodic structure. Experiments demonstrate that for sequences with period 32 and  $k = 3$ ,  $a$  can be as large as 12. Moreover, the best sequences have a highly regular structure, such as  $n_{i1} = i - (5k \bmod 16)$  and  $n_{i2} = i - 16 - (3k \bmod 16)$ , where  $k = i \bmod 16$ . For larger periods  $a$  cannot be computed directly; an analytical approach is desirable.

As soon as the set of indices is fixed,  $y_i$  for  $i > m$  can be output once  $x_i$  becomes available. The size of the buffer should be at least  $i - n_{ij}$  for any  $i > m$  and  $j$ . If  $\mathcal{N}$  is periodic, retiring older elements can be trivially implemented by keeping several buffers and rotating between them. We note that somewhat similar buffers were used recently in the design of the stream cipher PY [BS05].

More precisely, if we choose the period  $P = 32$  and  $k = 3$ , i.e. every output element is an XOR of three elements of the walk, the output rule can be implemented by keeping three  $P$ -word buffers,  $A$ ,  $B$ , and  $C$ . Their content is shifted to the left every  $P$  cycles:  $A$  is discarded,  $B$  moves to  $A$ , and  $C$  moves to  $B$ . The last operation can be efficiently implemented by rotating pointers to the three buffers.

The exact constants chosen for  $n_{ij}$  in the output rule are chosen to maximize the girth and other useful properties of the graph of dependencies between internal variables and the output, which is available to the attacker.

### Rotations.

Another operation used both in the output rule and in the update of the internal state is bit rotation. The motivation behind this is as follows: all the operations used in MV3 except for the rotation (that is, bitwise XOR, modular addition and multiplication) have the property that in order to know the  $k$  least significant bits of the output of the operation, it is sufficient to know the  $k$  least significant bits of the input. An attacker can use this property to devise an attack based on examining only the  $k$  least significant bits of the output words, and disregard all the other bits. This would dramatically reduce the time complexity of guess-and-determine attacks. For example, if no rotations were used in the cipher, then a variant of the standard guess-and-determine attack presented in Section 4 would apply. This variant examines only the least significant byte of every word, and reduces the time complexity of the attack to the fourth root of the original time complexity.

One possible way to overcome this problem is to use additional operations that do not have this problematic property, like multiplication in some other modular group. However, such operations slow the cipher significantly. The rotations used in MV3 can be efficiently implemented and prevent the attacker from tracing only the several least significant bits of the words. We note that similar techniques were used in the stream cipher SOSEMANUK [B+05] and in other ciphers as well.

**Key setup.** Since the bulk of the internal state is the table  $T$ , we concentrate on intermingling  $T$  and the pair  $(key, IV)$ . Once  $T$  is fully dependent on the  $key$  and the  $IV$ , the revolving buffers and other internal variables will necessarily follow suit.

We have specified that the  $IV$  be as long as the  $key$  in order to prevent time/memory tradeoff attacks that try to invert the function  $g : (key, IV) \rightarrow Output$ . The  $IV$  is known to the attacker but should not be easily predictable. One should avoid initializing the  $IV$  to zero at the beginning of every encryption session (as is frequently done in other applications), since this reduces the effective size of the  $IV$  and allows for better time/memory tradeoff attacks. A more comprehensive study of the security of MV3 with respect to time/memory tradeoff attacks is presented in Section 4.

We note that the key initialization phase is relatively slow. However, as the cipher is intended for encrypting long streams of data, the fast speed of the output stream generation compensates for it.

An open question for further research is whether the key initialization phase can be replaced by a faster one without reducing the security. It seems that due to the large size of the internal state of the cipher, a simple key initialization may lead to simple relations between parts of the internal state, that can be used by an attacker. This problem is demonstrated by the recent attacks [WP1, WP2] on the key initialization of the stream cipher Py, utilizing the same problem to mount devastating distinguishing and key-recovery attacks on the cipher. Hence, it seems that designing a fast and secure key initialization for a stream cipher with a large internal state may be a hard task.

## 4 Security

MV3 is designed to be a fast and very secure cipher. We are not aware of any attacks on MV3 faster than exhaustive key search even for huge key sizes of more than 1000 bits (except for the related key attacks in Section 4.6), but have only made security claims up to a 256-bit key size. In this section we analyze the security of MV3 against various kinds of cryptanalytic attacks.

### 4.1 Tests

We ran the cipher through several tests. First, we used two well-known batteries of general tests. One is Marsaglia’s time-tested DIEHARD collection [Mar97], and the other is the NIST set of tests used to assess AES candidates [R+01] (with corrections as per [KUH04]). Both test suites were easily cleared by MV3.

In light of attacks on the first few output bytes of RC4 [MS01,Mir02], the most popular stream cipher, we tested the distribution of the initial double words of MV3 (by choosing a random 160-bit key and generating the first double word of the output). No anomalies were found.

RC4’s key stream is also known to have correlations between the least significant bits of bytes one step away from each other [Gol97]. Neither of the two collections of tests specifically targets bits in similar positions of the output’s double words. To compensate for that, we ran both DIEHARD and NIST’s tests on the most and the least significant bits of 32-bit words of the key stream. Again, none of the tests raised a flag.

### 4.2 Time/Memory/Data Tradeoff Attacks

There are two main types of TMDTO (time/memory/data tradeoff) attacks on stream ciphers.

The first type consists of attacks that try to invert the function  $f : State \rightarrow Output$  (see, for example, [BS00]). In order to prevent attacks of this type, the size of the internal state should be at least twice larger than the key length. In MV3, the size of the internal state is more than 11,000 bits, and hence there are no TMDTO attacks of this type faster than exhaustive key search for keys of less than 5,500 bits length. Our table sizes are larger than what one may expect to be necessary to make adequate security claims, but we have chosen our designs so that we can keep our analysis of the components transparent, and computational overhead per word of output minimal. We intend to return to this in a future paper and propose an algorithm where the memory is premium, based on different principles for light weight applications.

The second type consists of attacks that try to invert the function  $g : (Key, IV) \rightarrow Output$  (see, for example, [HS05]). The *IV* should be at least as long as the key – as we have mandated in our key initialization – in order to prevent such attacks faster than exhaustive key search. We note again that if the *IV*’s are used in some predictable way (for example, initialized to zero at the beginning of the encryption session and then incremented sequentially), then the effective size of the *IV* is much smaller, and this may enable a faster TMDTO attack. However, in order to overcome this problem the *IV* does not have to be “very random”. The only thing needed is that the attacker will not be able to know which *IV* will be used in every encryption session. This can be achieved by initializing the *IV* in the beginning of the session using some (possibly insecure) publicly known pseudo-random number generator and then incrementing it sequentially.

### 4.3 Guess-and-Determine Attacks

A guess-and-determine attack against RC4 appears in [K+98]. The attack, adapted to MV3, has the following form:

1. The attacker guesses the values of all the 32 words in buffers *A* and *B* in some loop of MV3, and the values of  $j, c, x$  in the beginning of the loop.
2. Using the guessed values of the words in *B*, the attacker traces the value of  $j$  during the whole loop.
3. Using the output stream and the guessed values, the attacker traces the value of  $x$  during the whole loop.

4. Using the update rule of  $x$  and the knowledge of  $j$ , the attacker gets the values of 32 words in the  $T$  array. If the attacker encounters a word whose value is already known to her, she checks whether the values match, and if not, discards the initial guess.
5. The attacker moves on to the next loop. Note that due to the knowledge of buffer  $A$  and some of the words  $T[j]$ , the attacker can trace the update of  $c$  and of the  $T$  register.
6. Each “collision” in the  $T$  array supplies the attacker with a 32-bit filtering condition. Since the attacker started by guessing 66 32-bit words, finding 70 collisions should be sufficient to discard all the wrong guesses and find the right one. In 10 loops we expect to find more than 70 such collisions, and hence  $2^{14}$  bits of key stream will be sufficient for the attacker to find the internal state of the cipher.
7. Once the attacker knows the internal state, she can compute the entire output stream without knowing the key.

However, the time complexity of this attack is quite large – more than  $2^{2000}$ , since the attacker starts with guessing more than 2000 bits of the state. Hence, this attack is slower than exhaustive key search for keys of less than 2000 bits length.

#### 4.4 Guess-and-Determine Attacks Using the Several Least Significant Bits of the Words

Most of the operations in MV3 allow the attacker to focus the attack on the  $k$  least significant bits, thus dramatically reducing the number of bits guessed in the beginning of the attack. We consider two reasonable attacks along these lines.

The first attack concentrates on the least significant bit of the output words. In this case, since the least significant bit of  $c$  is fixed to 1, the attacker can disregard  $c$  at all. However, in this case the attacker cannot trace the values of  $j$ , and guessing them all the time will require a too high time complexity. Hence, it seems that this attack is not applicable to MV3.

The second attack concentrates on the eight least significant bits of every output word. If there were no rotations in the update and output rules, the attacker would indeed be able to use her guess to trace the values of  $j$  and the eight least significant bits in all the words of the internal state. This would result in an attack with time complexity of about  $2^{600}$ . However, the rotations cause several difficulties for such an attack:

1. Due to the rotations, the values the attacker knows after her initial guess are bits 24 – 31 of the words in buffer  $C$ , bits 16 – 23 of the words in buffer  $B$ , and bits 0 – 7 of the words in buffer  $A$  (these are the bits that affect the eight LSB’s of the output words). Yet the attacker still does not know the eight LSB’s of the words in buffer  $B$  and hence cannot find the value of  $j$ .
2. If the attacker rolls the arrays to the previous loop, she can find the eight LSB’s of the words in buffer  $B$ . However, the attacker cannot use her guess to get information from the previous loop. In that loop she knows bits 0 – 7 of the words in buffer  $B$  and bits 16 – 23 of the words in buffer  $C$ , but in order to compare the information with the eight LSB’s of the output stream, she needs bits 16 – 23 of the words in buffer  $B$  and bits 24 – 31 of the words in buffer  $C$ . Therefore, the guesses in consecutive loops cannot be combined together.

Hence, it seems that both of the attacks cannot be applied, unless the attacker guesses the full values of all the words in two buffers, which leads to the attack described in subsection 4.3 (with a time complexity of more than  $2^{2000}$ ).

#### 4.5 Linear Distinguishing Attacks

Linear distinguishing attacks aim at distinguishing the cipher output from random streams, using linear approximations of the non-linear function used in the cipher – in our case, the random walk.

In [CHJ02], Coppersmith et al. developed a general framework to evaluate the security of several types of stream ciphers with respect to these attacks. It appears that the structure of MV3 falls into this framework, to which [CHJ02, Theorem 6] directly applies:

**Theorem 1.** *Let  $\epsilon$  be the bias of the best linear approximation one can find for pairs  $x_i, x_{i+1}$ , and let  $A_N(a)$  be the number of equations of type (3.1) that hold for the sequence  $y_m, y_{m+1}, \dots$ . Then the statistical distance between the cipher and the random string is bounded from above by*

$$\sqrt{\sum_{a=1}^N A_N(a) \epsilon^{2a}}. \quad (4.1)$$

Note that for  $\epsilon \ll 1/2$ , the bound (4.1) is dominated by the term with the smallest  $a$ , which equals to 12 in our case. Since the relation between  $x_i$  and  $x_{i+1}$  is based on a random walk,  $\epsilon$  is expected to be very small. Since the statistical distance is of order  $\epsilon^{24}$ , we expect that the cipher cannot be distinguished from a random string using a linear attack, even if the attacker uses a very long output stream for the analysis.

#### 4.6 Related-Key Attacks and Key Schedule Considerations

Related key attacks study the relation between the key streams derived from two unknown, but related, secret keys. These attacks can be classified into distinguishing attacks, that merely try to distinguish between the key stream and a random stream, and key recovery attacks, that try to find the actual values of the secret keys.

One of the main difficulties in designing the key schedule of a stream cipher with a very large state is the vulnerability to related-key distinguishing attacks. Indeed, if the key schedule is not very complicated and time consuming, an attacker may be able to find a relation between two keys that propagates to a very small difference in the generated states. Such small differences can be easily detected by observing the first few words of the output stream.

It appears that this difficulty applies to the current key schedule of MV3. For long keys, an attacker can mount a simple related-key distinguishing attack on the cipher. Assume that  $keylength = 8192/t$ . Then in any step of the key initialization phase, every word of the key affects exactly  $t$  words in the  $T$  array, after which the main loop of the cipher is run eight times and the output stream is XORed (bit-wise) to the content of the  $T$  array. The same is repeated with the  $IV$  replacing the key in the  $IV$  initialization phase.

The attacker considers encryption under the same key with two  $IV$ s that differ only in one word. Since the key is the same in the two encryptions, the entire key initialization phase is also the same. After the first step of the  $IV$  initialization, the intermediate values differ in exactly  $t$  words in the  $T$  array. Then, the main loop is run eight times. Using the random walk assumption, we estimate that, with probability  $(1 - t/256)^{256}$ , each of the corresponding words in the respective  $T$  arrays used in these eight loops are equal, making the output stream equal in both encryptions. Hence, with probability  $(1 - t/256)^{256}$ , after the first step of the  $IV$  initialization the arrays  $A$ ,  $B$ , and  $C$  are equal in both encryptions and the respective  $T$  arrays differ only in  $t$  words.

The same situation occurs in the following three steps of the  $IV$  initialization. Therefore, with probability

$$(1 - t/256)^{256} \cdot (1 - 2t/256)^{256} \cdot (1 - 3t/256)^{256} \cdot (1 - 4t/256)^{256} \quad (4.2)$$

all of the corresponding words used during the entire initialization phase are equal in the two encryptions. Then with probability  $(1 - 4t/256)^{32}$  all of the corresponding words used in the first loop of the key stream generation are also equal in the two encryptions, resulting in two equal key streams. Surely this can be easily recognized by the attacker after observing the key stream generated in the first loop.

In order to distinguish between MV3 and a random cipher, the attacker has to observe about

$$M = (1 - t/256)^{-256} \cdot (1 - 2t/256)^{-256} \cdot (1 - 3t/256)^{-256} \cdot (1 - 4t/256)^{-256} \cdot (1 - 4t/256)^{-32} \quad (4.3)$$

pairs of related  $IV$ s, and for each pair she has to check whether there is equality in the first 32 key stream words. Hence, the data and time complexities of the attack are about  $2^{10}M$ . For keys of length at least 384 bits, this attack is faster than exhaustive key search. Note that (somewhat counter intuitively) the attack becomes more efficient as the length of the key is increased. The attack is most efficient for 8192-bit keys, where the data complexity is about  $2^{10}$  bits of key stream encrypted under the same key and  $2^{15}$  pairs of related  $IV$ s, and the time complexity is less than  $2^{32}$  cycles. For keys of length at most 256 bits, the data

and time complexities of the attack are at least  $2^{618}$  and hence the related-key attack is much slower than exhaustive key search.

If we try to speed up the key schedule by reducing the number of loops performed at each step of the key schedule, the complexity of the related-key attack is reduced considerably. For example, if the number of loops is reduced to four (instead of eight), the complexity of the related-key attack becomes

$$M' = (1 - t/256)^{-128} \cdot (1 - 2t/256)^{-128} \cdot (1 - 3t/256)^{-128} \cdot (1 - 4t/256)^{-128} \cdot (1 - 4t/256)^{-32} \quad (4.4)$$

In this case, the attack is faster than exhaustive key search for keys of length at least 320 bits. If the number of loops is further reduced to two, the complexity of the attack becomes

$$M'' = (1 - t/256)^{-64} \cdot (1 - 2t/256)^{-64} \cdot (1 - 3t/256)^{-64} \cdot (1 - 4t/256)^{-64} \cdot (1 - 4t/256)^{-32} \quad (4.5)$$

and then the attack is faster than exhaustive search for keys of length at least 224 bits.

If the key schedule is speed up by inserting the output of the eight loops into the  $T$  array, instead of XORing it bit-wise to the content of the  $T$  array (as was proposed in a previous variant of the cipher), the complexity of the related-key attack drops to

$$M''' = ((1 - t/256)^{-256})^4 \quad (4.6)$$

In this case, the attack is faster than exhaustive key search even for 256-bit keys.

Hence, the related-key attack described above is a serious obstacle to speeding up the key schedule. However, we note that the related-key model in general, and in particular its requirement of obtaining a huge number of encryptions under different related- $IV$  pairs, is quite unrealistic.

#### 4.7 Other Kinds of Attacks

We subjected the cipher to other kinds of attacks, including algebraic attacks and attacks exploiting classes of weak keys. We did not find any discrepancies in these cases.

## 5 Summary

We have proposed a new fast and secure stream cipher, MV3. The main attributes of the cipher are efficiency in software, high security, and its basis upon clearly analyzable components.

The cipher makes use of new rapidly mixing random walks, to ensure the randomness in the long run. The randomness in the short run is achieved by revolving buffers that are easily implemented in software, and break short correlations between the words of the internal state.

The cipher is word-based, and hence is most efficient on 32-bit processors. On a Pentium IV, the cipher runs with a speed of 4.8 clocks a byte.

#### Acknowledgements

We thank Adi Shamir for his generous discussions. We are grateful to Nir Avni and Uzi Vishne for their careful reading and comments on an earlier version, and Rebecca Landy for providing numerics on expander graphs.

#### References

- [ARV95] W. Aiello, S. Rajagopalan, and R. Venkatesan, *Design of Practical and Provably Good Random Number Generators*, Proc. of SODA'95, pp. 1–9, 1995.
- [AR94] N. Alon and Y. Roichman, *Random Cayley Graphs and Expanders*, Rand. Str. Alg. vol. 5(2), pp. 271–284, 1994.
- [B+05] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, and H. Sibert, *Sosemanuk, a Fast Software-Oriented Stream Cipher*, submitted to *Ecrypt*, 2005.

- [BS05] E. Biham and J. Seberry, *Py (Roo): A fast and secure stream cipher using rolling arrays*, submitted to Ecrypt, 2005.
- [BS00] A. Biryukov and A. Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Proc. of Asiacrypt'00, pp. 1–13, 2000.
- [CHJ02] D. Coppersmith, S. Halevi, and C. S. Jutla, *Cryptanalysis of stream ciphers with linear masking*, Proc. of CRYPTO'02, pp. 515–532, 2002.
- [DC06] C. De Canniere, *eSTREAM testing framework*, 2006, available on-line at <http://www.ecrypt.eu.org/stream>.
- [Gi98] D. Gillman, *A Chernoff bound for random walks on expander graphs*, SIAM J. Comput. 27 (4), pp.1203–1220 (1998).
- [Gol97] J. Golić, *Linear statistical weakness of alleged RC4 keystream generator*, Proc. of Eurocrypt'97, pp. 226–238, 1997.
- [HCJ02] S. Halevi, D. Coppersmith, and C. Jutla, *Scream: a Software-Efficient Stream Cipher*, Proc. of FSE'02, pp. 195–209, 2002.
- [HS05] J. Hong, P. Sarkar, *Rediscovery of Time Memory Tradeoffs*, 2005. Available online at <http://eprint.iacr.org/2005/090>.
- [HLW06] S. Hoory, N. Linial, and A. Wigderson, *Expander graphs and their applications*, Bull. Amer. Math. Soc. 43 (2006), pp. 439–561.
- [MV3] N. Keller, S.D. Miller, I. Mironov, and R. Venkatesan, *MV3: A new stream cipher based on random walks and revolving buffers*, Topics in Cryptology – Proceedings of CT-RSA 2007, Springer Verlag Lecture Notes in Computer Science, 4377 (2007), pp. 1-19.
- [Full MV3] Full version of [MV3], posted at <http://www.arxiv.org/abs/cs.CR/0610048>
- [KUH04] S. Kim, K. Umeno, and A. Hasegawa, *Corrections of the NIST statistical test suite for randomness*, Cryptology ePrint Archive, Report 2004/018, 2004.
- [KS02] A. Klimov and A. Shamir, *A New Class of Invertible Mappings*, Proc. of CHES'02, pp. 470–483, 2002.
- [KS04] A. Klimov and A. Shamir, *New Cryptographic Primitives Based on Multiword T-Functions*, Proc. of FSE'04, pp. 1–15, 2004.
- [K+98] L. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, *Analysis Methods for (Alleged) RC4*, Proc. of ASIACRYPT'98, pp.327-341, 1998.
- [LPS86] A. Lubotzky, R. Phillips, and P. Sarnak, *Explicit expanders and the Ramanujan conjectures*, Proc. of STOC'86, pp. 240–246, 1986.
- [MS01] I. Mantin and A. Shamir, *A practical attack on broadcast RC4*, Proc. of FSE'01, pp. 152–164, 2001.
- [Mar97] G. Marsaglia, *DIEHARD battery of tests*, available from <http://stat.fsu.edu/~geo/>.
- [Mir02] I. Mironov, *(Not so) random shuffles of RC4*, Proc. of CRYPTO'02, pp. 304–319, 2002.
- [P+03] B. Preneel et al., *NESSIE Security Report, version 2.0*, 2003.
- [RC98] P. Rogaway and D. Coppersmith, *A Software-Optimized Encryption Algorithm*, J. of Cryptology 11(4), pp. 273–287, 1998.
- [R+01] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, NIST Special Publication 800-22, <http://www.nist.gov>, 2001.
- [Sch95] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Ed., John Wiley & Sons, 1995.
- [WP1] H. Wu, B. Preneel, *Attacking the IV Setup of Py and Pypy*, available at <http://www.ecrypt.eu.org/stream/papersdir/2006/050.pdf>
- [WP2] H. Wu, B. Preneel, *Key Recovery Attack on Py and Pypy with Chosen IVs*, available at <http://www.ecrypt.eu.org/stream/papersdir/2006/052.pdf>