A Practical Attack Against the Use of RC4 in the HIVE Hidden Volume Encryption System*

Kenneth G. Paterson[1] and Mario Strefler[2]

[1]Information Security Group, Royal Holloway, University of London
Kenny.Paterson@rhul.ac.uk
[2]Karlsruhe Institute of Technology
Mario.Strefler@kit.edu

Abstract. The HIVE hidden volume encryption system was proposed by Blass et al. at ACM-CCS 2014. Even though HIVE has a security proof, this paper demonstrates an attack on its implementation that breaks the main security property claimed for the system by its authors, namely plausible hiding against arbitrary-access adversaries. Our attack is possible because of the HIVE implementation's reliance on the RC4 stream cipher to fill unused blocks with pseudorandom data. While the attack can be easily eliminated by using a better pseudorandom generator, it serves as an example of why RC4 should be avoided in all new applications and a reminder that one has to be careful when instantiating primitives.

1 Introduction

At ACM CCS 2014, Blass et al. [2] presented a novel "hidden volume encryption" system called HIVE. Their system splits an encrypted disk into several volumes. The intention of HIVE is to hide the existence of some of these volumes, and to hide the pattern of read and write accesses to the volumes from observers.

To hide the access patterns, reading and writing has to be indistinguishable. This is accomplished by overwriting an empty block with pseudorandom data during a read operation to simulate an encryption. The HIVE implementation makes use of the RC4 algorithm to overwrite blocks with pseudorandom data and AES in CBC mode to perform encryption. This approach can be proved secure if both the ciphertexts produced by the symmetric encryption scheme and the output of the pseudorandom generator are indistinguishable from random. We stress that RC4 is not used to encrypt any data in the scheme, but only to fill unused disk space with pseudorandom bytes. According to [2], RC4 was selected for performance reasons.

We show that RC4 is a poor choice for this task: we demonstrate that its use enables us to break the main security property claimed for HIVE in [2], namely "plausible hiding against arbitrary-access adversaries". In fact, we will break the even weaker notion of plausible hiding against *one-time* adversaries. Our attack does not extract plaintext, but we are able to detect the supposedly hidden volumes by only examining the disk twice.

* The first author was supported by EPSRC Leadership Fellowship EP/H005455/1. The second author did most of the work while working at Aarhus University and was supported by European Research Council Starting Grant 279447 and by the Danish Council for Independent Research via DFF Starting Grant 10-081612.

The supposed infeasability of this kind of attack is the main advantage claimed for HIVE over conventional disk encryption systems and was used to justify its storage overhead [2].

Our attack exploits biases in RC4 keystreams. While it is well-known that RC4 keystreams are biased, and therefore not indistinguishable from random byte streams [5,3,4,1,6], it is not immediately obvious that these existing results can be applied to break any claimed security property of HIVE. This is because previous distinguishers [3,4] require access to either a very large amount of consecutive keystream bytes, or to a corresponding amount of truly random bytes, whereas HIVE acts in a blockwise fashion (with the typical block size being 4096 bytes), presenting a distinguisher with a *mix* of short RC4 blocks and AES-CBC blocks.

Our adversary makes use of the Mantin biases in RC4 key-streams [4] to build a weak blockwise distinguisher for short blocks; in contrast to previous work identifying RC4 biases [3,4] we explicitly present an efficient and near-optimal statistical test that can be used to implement the distinguisher. Our blockwise distinguisher is then applied repeatedly over many blocks to estimate how many blocks on the disk are filled with RC4 and how many are filled with AES-CBC ciphertexts; this approach seems to be novel and not previously exploited in the literature. Finally, the estimate can be used to decide what sequence of read and/or write accesses was performed on the disk by the HIVE system.

Our adversary is efficient, and it needs access only to a moderate number of read/write queries on blocks. For example, it has success rate 0.997 in an attack involving $2^{23.7}$ read/write queries on a disk containing a total of $2^{24.7}$ blocks and 104 GB of data.

Our approach does not involve the derivation of any new properties of RC4. However it does illustrate that RC4 is not only unfit for use as a general purpose pseudorandom generator (PRG), but also that it can result in systems that can be practically attacked in scenarios where RC4's weaknesses are not obviously problematic.

We informed the authors of [2] about our attack at a conceptual level, and, while they updated their FAQ for the HIVE system[1], they did not initially take the opportunity to revise their source code to use a better PRG or amend their research paper describing the system. They instead argued that RC4 is only an exchangeable building block that can easily be replaced. We note that RC4 was the default, and it is unlikely that users would replace it with their own implementation of a secure PRG, leaving them at risk. Given that the authors released HIVE to the public and still claimed that it "provides more security than all existing schemes",[2] we therefore decided to prepare this paper to alert potential users of the HIVE implementation to the shortcomings of the system. We are happy to note that after publication of a pre-print of this paper, RC4 was replaced by AES-CTR in the HIVE implementation.

We again emphasise that we attack only the specific implementation of HIVE that instantiates the PRG with RC4, and do not claim that the HIVE scheme is insecure if

[1] `http://hive.ccs.neu.edu/FAQ`, date of access 26/10/2014
[2] `http://hive.css.neu.edu`, date of access 26/10/2014

instantiated with a different PRG. Indeed we consider HIVE to be a very interesting proposal that is worthy of further (crypt)analysis.

1.1 Paper Organisation

In the next section, we provide more details on HIVE and its use of RC4. We build an RC4 distinguisher for short blocks in Section 3. Section 4 describes how we use this distinguisher to attack the HIVE scheme.

2 Description of the HIVE Scheme

HIVE [2] is a hidden volume encryption scheme that works on a storage device divided into blocks. The user chooses a value l and can then configure up to l logical volumes V_i, where each volume is encrypted with a key derived from a password P_i. Without knowledge of the corresponding password, a volume should be undetectable. The goal is that "a user can plausibly deny the existence of a hidden volume even if the adversary has been able to take several snapshots of their disk and knows the password for the main volume."[3] This makes it necessary to hide the pattern of accesses that a user makes to the disk, which is achieved by building on a write-only oblivious RAM (ORAM) scheme.

The original paper presents several different security notions and two different schemes; to fix the attack target, we choose a weak security notion and one scheme, which we describe below. For simplicity, we assume that all ORAMs have the same size, $1/l$ times the size of the hard disk.

2.1 ORAM

The stash-optimized, write-only ORAM scheme from [2] is a logical overlay of an encrypted disk over a physical disk at least twice the size of the encrypted disk. It makes use of a pseudo-random generator (PRG) and a symmetric encryption scheme with the property that without the key, ciphertexts are indistinguishable from random strings of the same length. The security goal for ORAM is to hide access patterns to the encrypted disk.

The ORAM maintains a map of logical blocks to physical blocks. Initially, all logical blocks are mapped to $\perp$. At each ORAM.Write(logical block, data block) operation, the ORAM picks K physical blocks at random (independently of the logical address) and tries to decrypt each of them. Some may already contain data, while others may be empty. The new data block to be written and any data blocks that were already stored in the selected K blocks are written to memory. Specifically, they are added to a *stash*, which acts as a buffer to store blocks yet to be written to disk. Then as many of the data blocks in the stash as possible are encrypted and written back to the selected K blocks. Because the physical disk is much larger than the logical disk, there is a good chance that one or more of the K selected blocks were still free, so that all the blocks

[3] [2], Introduction

currently in the stash can be written to the disk. Any remaining free blocks from the K selected blocks are overwritten with random data (generated by the PRG). The map from logical to physical blocks is updated accordingly.

If not enough of the selected blocks were free, then some data blocks will remain in the stash, to be written during a later write operation, when free blocks are found. In general, the parameters of the scheme are selected so that there is a good probability that one or more blocks will be free, so that with high probability the stash size remains small throughout. For example, for $K = 3$, the probability of having more than 50 items in the stash is bounded by 2^{-64}.

The security argument is that since for each operation, K randomly selected blocks are overwritten with a pseudo-random string, the operation does not leak any information about the logical address, the data to be written, or the state of the disk (i.e. which blocks are free).

2.2 HIVE

The security target for HIVE is that it should hide the existence of encrypted disk volumes for which the key is not known. This is accomplished by executing a real ORAM.Read and a dummy ORAM.Write for each Read, and a dummy ORAM.Read and a real ORAM.Write for each Write.

It is claimed in [2] that HIVE achieves the notion of plausible hiding against arbitrary-access adversaries. We will break the even weaker notion of plausible hiding against one-time adversaries, which is given in Definition 1 immediately below.

Definition 1 (Plausible hiding against one-time adversaries). *The experiment* $\mathrm{Exp}^{pl\text{-}ot\text{-}b}_{\mathcal{A},\Sigma}(k)$ *for a bit* b *is run between an adversary* $\mathcal{A}$ *and a challenger emulating the scheme* Σ *and consists of the following phases.*

1. *In the setup phase,* $\mathcal{A}$ *sends* l *to the challenger, who chooses* l *passwords* $P_1, \ldots, P_l$*. The challenger initializes* Σ_0 *with* l *volumes and passwords, and* Σ_1 *with* $l-1$ *volumes and passwords* $P_1, \ldots, P_{l-1}$ *and sends* $P_1, \ldots, P_{l-1}$ *and a snapshot* D_0 *of* Σ_b *to* $\mathcal{A}$*.*
2. *In round* i*,* $\mathcal{A}$ *sends two accesses* $o_{i,0}$ *and* $o_{i,1}$ *to the challenger; the challenger executes* $o_{i,b}$ *on* Σ_b*.*
3. *Finally, the adversary requests a snapshot* D_f *of the disk, and outputs a bit* b'*, which is the output of the experiment.*

An access o *is of the form* $o = (op, b, V, d)$*. If* $op = w$*, then data* d *is written to block* b *on volume* V*. If* $op = r$*, then block* b *from volume* V *is read into* d*. Since the adversary knows the passwords* $P_1, \ldots, P_{l-1}$*, if one of the operations in round* i *is a write to one of the volumes* $V_1, \ldots, V_{l-1}$*, then both operations must be identical. Intuitively, this means that any access to* V_l *can be passed off as an access to another volume.*

We define the advantage of the adversary as

$$\mathrm{Adv}^{pl\text{-}ot}_{\mathcal{A},\Sigma}(k) = |\Pr[1 \leftarrow \mathrm{Exp}^{pl\text{-}ot\text{-}0}_{\mathcal{A},\Sigma}(k)] - \Pr[1 \leftarrow \mathrm{Exp}^{pl\text{-}ot\text{-}1}_{\mathcal{A},\Sigma}(k)]|.$$

If we used an asymptotic definition, we would say that Σ is secure if for all PPT $\mathcal{A}$, $\mathrm{Adv}^{\textit{pl-ot}}_{\mathcal{A},\Sigma}(k)$ is a negligible function. Since we want to attack a concrete instance, we use a concrete security definition. We fix a concrete security parameter k and let

$$\mathrm{Adv}^{\textit{pl-ot}}_{\Sigma,k}(\tau, q) = \max_{\mathcal{A}}\{\mathrm{Adv}^{\textit{pl-ot}}_{\mathcal{A},\Sigma}(k)\},$$

where the maximum is taken over all adversaries running[4] *in at most τ steps and making at most q access queries (that is, there are at most q rounds in the game, each round involving either writing to or reading from a single block). We then say that for a concrete parameter k, Σ is (ε, τ, q)-plausible hiding against one-time adversaries, if $\mathrm{Adv}^{\textit{pl-ot}}_{\Sigma,k}(\tau, q) \leq \varepsilon$.*

2.3 HIVE Instantiation

In HIVE, the symmetric encryption scheme is instantiated with AES-CBC using 256-bit, password-derived keys and a fresh IV at each call. The number of blocks written to for each operation, K (k in the original paper), is set to 3 in the implementation.[5] This low value of K necessitates the use of a stash of unwritten blocks to handle the situation where, when K random blocks are selected, all are already in use. From the HIVE source code it appears that a volume is not initially overwritten with random data. This will help our adversary, since it means it will need to consider less additional data, but is not necessary for the attack to work.

The implementation follows the good practice of using a PRG seeded once with true randomness obtained from the OS to generate all the randomness consumed by the cryptographic operations involved in HIVE. Specifically, the output of the PRG is used to:

1. produce a fresh, random 16-byte IV (for use with each call to AES-CBC);
2. fill a 4096-byte sector with random bytes;
3. fill a 32-byte metadata block with random bytes. A metadata block consists of two 8-byte values and a 16-byte IV.
4. select a random sector to write to. This requires a random 8-byte value for the sector id.

The PRG, while being properly seeded, is instantiated with RC4-drop256. According to [2] this is for performance reasons. More specifically, RC4 is keyed using 256 bytes of randomness obtained from the OS, the first 256 bytes of RC4 output are dropped to avoid well-known strong biases (see for example [1] for a complete exposition of these), some further bytes of output are used for other purposes, and then B consecutive bytes of output are used to fill a block with "random" bytes. The HIVE implementation uses $B = 4096$ (so it has 4 KB blocks), though the choice $B = 256$ is also discussed in [2]. Note that we do not (and do not need to) know from precisely where in the RC4 keystream the bytes are selected, but only that they are consecutive.

[4] for an appropriate definition of running time, a problem we do not consider in this paper
[5] Variable HIVE_K in dm-hive.c

3 Blockwise Distinguisher for RC4

In this section, we develop and evaluate a distinguisher $\mathcal{D}$ for short RC4 keystreams as used in HIVE. This will be used in the next section as a component in building an attack against HIVE.

3.1 RC4 Biases

We first recall the main results on biases in RC4 outputs from [4] that we will use. Other biases, notably those in [3] are available, but not so convenient to use in the setting of interest to us. This is because they are position-dependent, and we do not wish to make any assumptions about exactly which positions in the RC4 keystreams the bytes we are targeting are selected from (since this is not readily apparent from the description of HIVE in [2] and the corresponding source code).

The following result is a restatement of Theorem 1 in [4], concerning the probability of occurrence of byte strings of the form $AB\mathcal{S}AB$ in RC4 outputs, where A and B represent bytes and $\mathcal{S}$ denotes an arbitrary byte string of a particular length G.

Result 1 *Let $G \geq 0$ be a small integer. Under the assumption that the RC4 state is a random permutation at step r, then*

$$\Pr\left(Z_r = Z_{r+G+2} \wedge Z_{r+1} = Z_{r+G+3}\right) = 2^{-16}\left(1 + \frac{e^{(-4-8G)/256}}{256}\right).$$

Note that for a truly random byte string $Z_r, \ldots, Z_{r+G_3}$, the probability that $Z_r = Z_{r+G+2}$ and $Z_{r+1} = Z_{r+G+3}$ is equal to 2^{-16}. The relative bias is therefore equal to $e^{(-4-8G)/256}/256$.

The above result was experimentally confirmed in [4] for values of G up to 64, though with quite a small sample size, and not focused on early bytes in RC4 output. We have confirmed that the result holds to a reasonable approximation in the situation in which we are interested, for positions 256 onwards. See Figure 1.

3.2 Discrimination and Statistical Hypothesis Testing

Given two probability distributions p and q on some set $\mathcal{S}$, we define the *discrimination* between p and q, denoted $L(p, q)$, to be $\sum_{s\in\mathcal{S}} p(s) \log p(s)/q(s)$. Note that discrimination is additive: if p_1, p_2, q_1, q_2 are distributions on $\mathcal{S}$, and if p_1p_2, q_1q_2 denote the product distributions on $\mathcal{S} \times \mathcal{S}$, then $L(p_1p_2, q_1q_2) = L(p_1, q_1) + L(p_2, q_2)$. This equality extends in the obvious way to larger products of distributions.

Next consider the distributions p and q arising from a simple "bias presence" test $\mathcal{T}$ based on a Mantin "$AB\mathcal{S}AB$" bias. The test $\mathcal{T}$ receives as input a B-byte string, looks for a particular byte pattern of a fixed type commencing in a fixed position, and outputs 1 if the pattern is detected, and 0 otherwise (here we assume the bias is a positive one, as is the case for all the Mantin biases). For example, the test might examine positions

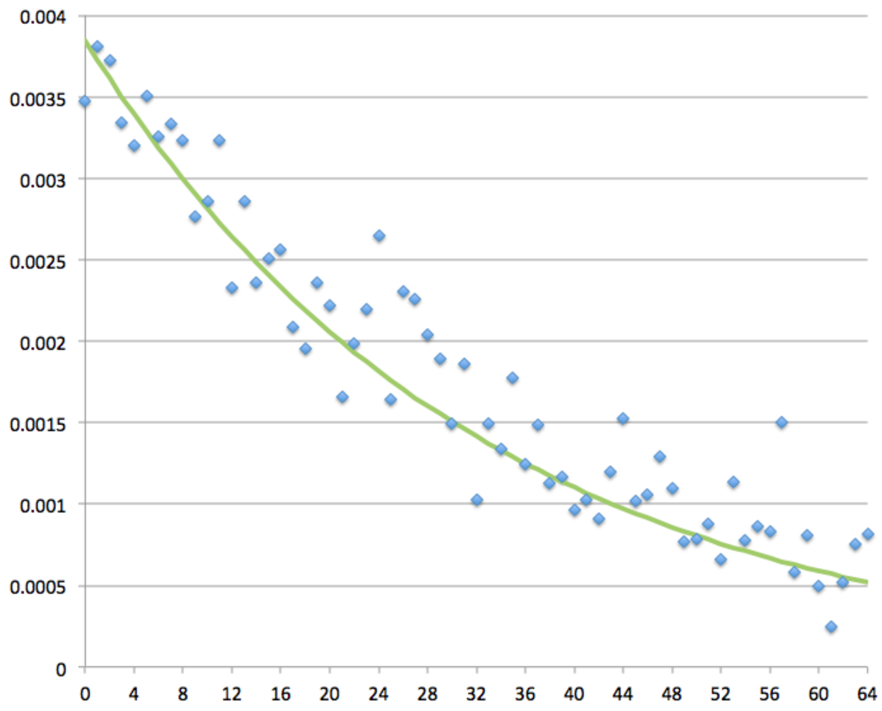

Fig. 1. Experimental validation of Mantin biases based on 2^{28} random 256-byte keys, 4096 bytes of RC4 output per key. The x-axis depicts the value of G, the y-axis scaled biases (γ values). Blue points denote experimentally measured values; green line denotes theoretical values computed according to Result 1.

r, $r+1$, $r+2$, $r+3$ in the string and check whether the quartet of byte values are of the form "$ABAB$" or not, an event which should happen with slightly larger than expected probability if the input string comes from RC4 output rather than being truly random.

Let p be the output probability distribution of $\mathcal{T}$ if the input string comes from RC4 output and q the output probability distribution of $\mathcal{T}$ if the input string is truly random. Then $p(0) = 1-\rho(1+\gamma)$, $p(1) = \rho(1+\gamma)$ and $q(0) = 1-\rho$, $q(1) = \rho$ where $\rho = 2^{-16}$ and $\gamma > 0$ is the relative bias under consideration (so $\gamma = e^{(-4-8G)/256}/256$ for some small integer G).

Then, as shown in Lemma 3 of [4] (using different notation), $L(p,q) \approx \rho\gamma^2$.

The presence of different types of biases in different positions motivates us to consider product distributions $p_1 \cdots p_t$ and $q \cdots q$, where each component in the two products corresponds to a different test $\mathcal{T}_i$ based on a specific bias. Here the second distribution is always a fixed one, q, as defined above (since all the bias presence tests behave the same way in case the input string is truly random), while the first one p_i describes the distribution of the test's $\{0,1\}$-valued outcome O_i. We assume we have t tests in total, and we make the assumption that all the simple bias presence tests that we can perform on our B-byte input are independent. This assumption seems reasonable in view of the different types of test being conducted, despite many of the tests involving overlapping bytes. The same assumption was made in [3,4] when developing distinguishers for RC4. This assumption enables us to write:

$$L(p_1 \cdots p_t, q \cdots q) = \sum_{i=1}^{t} L(p_i, q) \approx \rho \sum_{i=1}^{t} \gamma_i^2$$

where γ_i is the appropriate value for the i-th test performed.

Now let $\mathcal{D}$ denote any distinguisher based on an input that is the concatenation of the outputs O_i of the t individual bias presence tests, and that predicts whether or not the particular string of B bytes used in the tests was generated by RC4 or is truly random. We assume that $\mathcal{D}$ outputs 1 (indicating RC4) with probability $1 - \beta$ when its input is from RC4, and that $\mathcal{D}$ outputs 0 (indicating truly random) with probability $1 - \alpha$ when its input is truly random. In other words, α is the false positive rate for $\mathcal{D}$ and β is the false negative rate for $\mathcal{D}$. Then, following [3], we have that:

$$L(p_1 \cdots p_t, q \cdots q) \geq \beta \log_2 \frac{\beta}{1-\alpha} + (1-\beta) \log_2 \frac{1-\beta}{\alpha}.$$

Moreover equality is achieved by any optimal statistical test, such as a Neyman-Pearson likelihood ratio test.

We next evaluate $L(p_1 \cdots p_t, q \cdots q) \approx \rho \sum_{i=1}^{t} \gamma_i^2$ for the set of tests corresponding to all the possible Mantin biases arising in a B-byte block of consecutive keystream bytes. From the above equation, this will then allow us to establish bounds on (α, β) for optimal distinguishers $\mathcal{D}$. We then provide an efficient and approximately optimal statistical test based on these biases, and compute the parameters of the test in such a way as to maximise the quantity $1 - \alpha - \beta$, which is the usual advantage of the distinguisher $\mathcal{D}$.

3.3 Computation for the Mantin Biases

In a B-byte block, we can perform $B-(G+3)$ bias presence tests with $\gamma = e^{(-4-8G)/256}/256$ for each value of $G \geq 0$. In practice, since the biases decrease in size with increasing G, we work with values G satisfying $0 \leq G \leq G_{\max}$ for some value $G_{\max}$. In our experiments, we take $G_{\max} = 64$.

Combining all of these bias presence tests, we obtain $t = \sum_{G=0}^{G_{\max}} B-(G+3) \approx G_{\max} \cdot B$ tests such that

$$L(p_1 \cdots p_t, q \cdots q) = \rho \sum_{G \geq 0} (B - (G+3)) \cdot e^{(-4-8G)/128}/2^{16}).$$

Direct calculation gives $L(p_1 \cdots p_t, q \cdots q) \approx 8.7276 \approx \times 10^{-7}$ for $B = 256$ and $G_{\max} = 64$. For $B = 4096$ and $G_{\max} = 64$ we obtain $L(p_1 \cdots p_t, q \cdots q) \approx 1.4914 \times 10^{-5}$.

3.4 An Efficient Statistical Test

We start with a likelihood ratio test and develop from this an efficient test which is approximately optimal.

In the likelihood ratio test, the test statistic is computed as a ratio $\Lambda(O) := L(\theta_0|O)/L(\theta_1|O)$ where θ_0 denotes the distribution arising from the hypothesis H_0, θ_1 denotes the distribution arising from the alternative hypothesis H_1, O denotes the observed data, and $L(\theta_i|O) := \Pr(O|\theta_i)$ denotes a likelihood. In the test, the hypothesis H_0 is rejected (and our distinguisher outputs 1 indicating that its input is believed to be RC4) if the ratio of likelihoods $\Lambda(O)$ is less than or equal some value η; otherwise, H_0 is accepted (and our distinguisher outputs 0 indicating that its input is believed to be truly random). In principal η can be calculated from α, the false positive rate for the test. Specifically, η is determined as the value such that $\Pr(\Lambda(O) \leq \eta | H_0) = \alpha$.

In our situation, O is a vector composed of the outcomes O_i of the individual tests $\mathcal{T}_i$ with parameters γ_i. Hypothesis H_0 is that the input sequence of B bytes is random and H_1 is the hypothesis that it is an output of RC4. Keeping in mind that $\gamma_i > 0$ for all i, we can then write

$$L(\theta_0|O) = \prod_{i:O_i=1} \rho \prod_{i:O_i=0} 1 - \rho$$

and

$$L(\theta_1|O) = \prod_{i:O_i=1} \rho(1+\gamma_i) \prod_{i:O_i=0} 1 - \rho(1+\gamma_i).$$

Then, taking logs, using the fact that ρ and all the γ_i are small, manipulating the above expressions using standard log approximations, and simplifying, we finally obtain:

$$\log \Lambda(O) \approx -\sum_{i:O_i=1} \gamma_i + \sum_{i:O_i=0} \rho\gamma_i \,.$$

Let us denote the above quantity by L. Thus a suitable, approximately optimal test is to reject the hypothesis H_0 (that the input string is a truly random string) and output "1" if $L \leq \log \eta$, and to output "0" otherwise. Note that the test statistic L is efficient to compute: it requires on the order of $G_{\max} \cdot B$ floating point operations and byte comparisons.

It remains to compute appropriate values of η for a given target α. This requires us to know the distribution of the test statistic L when O comes from distribution θ_0. Now, in this case, the outcome O_i is Bernoulli distributed with parameter ρ, i.e. $\Pr(O_i = 1) = \rho$. Moreover, we can group the tests into $G_{\max} + 1$ groups, each group corresponding to a set of tests having the same value for γ_i. Since the number of tests in each group is large, the sum of the corresponding terms in L can be approximated by Normal distributions with mean and variance that can be explicitly calculated. This in principal allows the distribution of L to be computed. We omit the details.

3.5 Optimal Choice of Parameters

For reasons that will become clear in the next section, we wish to maximise the value of the quantity $\delta := 1 - \alpha - \beta$ over the choice of distinguisher $\mathcal{D}$ acting on B bytes of input. Let us denote this value by δ_B. (As temporary motivation note that δ is the usual cryptographic definition of the advantage of the distinguisher.) Since we are interested in optimal distinguishers, our choice of parameters (α, β) is subject to the constraint that $L_B = \beta \log_2 \frac{\beta}{1-\alpha} + (1-\beta) \log_2 \frac{1-\beta}{\alpha}$ where the quantity L_B was computed for different values of B in Section 3.3. Direct maximisation using Wolfram alpha[6] yields $\delta_B = 0.00055$ for $B = 256$ (at $(\alpha, \beta) = (0.499459, 0.499991)$) and $\delta_B = 0.002273$ for $B = 4096$ (at $(\alpha, \beta) = (0.498343, 0.499384)$).

3.6 Implementation

We implemented the above statistical test to ensure that its practical performance is in line with the theoretical analysis. As noted above the parameter η could be set by calculating the distribution of the test statistic L when O comes from distribution θ_0. Another approach would be to generate many samples of the test statistic to estimate the distribution of L and set η by computing the α-percentile of the sampled values.

In our experiments, we simply set $\eta = 0$, ran the test on random inputs and on RC4 inputs, and computed the corresponding values of α, β, δ. This gives us sub-optimal values for δ, but is sufficient to validate that the test works and gives us a distinguisher having performance that is reasonably close to that which is theoretically obtainable. For $B = 4096$ and based on 2^{25} samples (2^{24} for each of random input and RC4 inputs), setting $\eta = 0$, we obtained $\alpha = 0.453706$, $\beta = 0.544646$ and $\delta = 0.001648$. Here the value of δ compares favourably to the theoretical maximum of 0.002273. We are confident that a closer match would be obtained by adjusting η. In our evaluation of our attack on HIVE to follow, we present attack costs for both the ideal value of δ and our experimentally obtained value.

4 An Attack on HIVE Based on a Blockwise RC4 Distinguisher

We describe a family of adversaries $\{\mathcal{A}_{B,S,T}\}$, parameterized by the block length B and two other parameters S and T which are defined below. Our adversaries have access to the distinguisher $\mathcal{D}$ acting on B bytes from Section 3. Recall that $\mathcal{D}$ outputs a single bit b; if $\mathcal{D}$'s input is uniformly random bytes, $\mathcal{D}_B$ outputs 1 with probability α; if $\mathcal{D}$'s input consists of B consecutive bytes of RC4 output, $\mathcal{D}$ outputs 1 with probability $1 - \beta$.

For a fixed block length B, $\mathcal{A}_{B,S,T}$ is a one-time adversary against plausible hiding for two volumes, as in Definition 1. In this simple case, there are two encrypted volumes V_1, V_2 protected by passwords P_1, P_2, and a user wants to be able to deny the existence of V_2 to an adversary who knows P_1. We consider two volumes for simplicity and because two

[6] http://www.wolframalpha.com/

volumes are used in the implementation; the attack generalizes directly to any number of volumes. We assume that each volume consists of S blocks of B bytes, so the disk has $4SB$ bytes in total (because it needs to be twice the size of all the volumes combined).

We overload the notation for the access operations in Definition 1 to allow reading and writing of multiple blocks of data in a single query. In what follows, 0^{SB} denotes an all-zero string of SB bytes, equating to S blocks of B bytes each.

Our adversary $\mathcal{A}_{B,S,T}$ then proceeds in five steps:

1. $\mathcal{A}_{B,S,T}$ sends $l = 2$ to the challenger and receives the password P_1 and a snapshot D_0 of the disk.
2. $\mathcal{A}_{B,S,T}$ sends $o_{1,0} = o_{1,1} = (w, 0, V_1, 0^{SB})$.
3. $\mathcal{A}_{B,S,T}$ sends $o_{2,0} = (w, 1, V_2, 0^{SB})$, $o_{2,1} = (r, 0, V_1, d, SB)$.
4. $\mathcal{A}_{B,S,T}$ requests a snapshot D_f, containing $4S$ blocks. $\mathcal{A}_{B,S,T}$ disregards the blocks that are the same as in D_0, runs $\mathcal{D}$ successively on each of the remaining E used blocks of D_f and concatenates the output bits of $\mathcal{D}$ to form a string $R \in \{0,1\}^E$.
5. Finally, $\mathcal{A}_{B,S,T}$ outputs 1 if W, the Hamming weight of R, is at most T; otherwise it outputs 0.

The following theorem is immediate (noting that reading or writing B bytes counts as making 1 query in the security model given previously, so $\mathcal{A}_{B,S,T}$ makes $2S$ queries in total):

Theorem 2. *Let $\tau_{B,S,T}$ denote the running time of $\mathcal{A}_{B,S,T}$ and $\mathrm{Adv}_{B,S,T}$ its advantage. Then HIVE is not $(\mathrm{Adv}_{B,S,T}, \tau_{B,S,T}, 2S)$-plausible hiding against one-time adversaries.*

Now let us determine values for S, B and T for which the theorem gives a meaningful attack. We consider what information $\mathcal{A}_{B,S,T}$ has access to in its attack when $b = 0$ and $b = 1$ in the security experiment. When $b = 0$, the disk is initialised with V_1 and V_2, and $o_{1,0}$ and $o_{2,0}$ are executed. When $b = 1$, the disk is initialised only with V_1, and $o_{1,1}$ and $o_{2,1}$ are executed.

The adversary gets an initial snapshot of the disk before any operations are executed, so identifying unused blocks is trivial; E denotes the number of blocks that differ between D_0 and D_f. All blocks that are written to certainly differ, so we have $2S \leq E \leq 4S$. We call EB the effective disk size, since the other blocks are irrelevant to the attack. Note that the expected value of E is approximately $3.1075S$. (There are $4S$ blocks on the disk, and in the $2S$ read and write operations in our attack, HIVE will choose $K \cdot 2S = 6S$ blocks at random to write to. This means that we expect that, out of the total of $4S$ blocks available, $4S \cdot \left(1 - \frac{1}{4S}\right)^{6S}$ blocks are *not* written to.[7] Using $(1 - 1/n)^m \approx e^{-m/n}$, we get $E[E] \approx (1 - e^{-1.5})4S \approx 3.1075S$.)

Let W_b denote the Hamming weight of the vector R constructed by $\mathrm{Adv}_{B,S,T}$ in case b. We have:

[7] In reality, because of the way the stash operates, slightly fewer than S or 2S blocks of data might be written to the disc. However, this does not materially affect our analysis. The probability of eight or more items remaining on the stash is only 0.05 %.

- When $b = 0$, $2S$ blocks of data are written to the disk, so it contains $2S$ blocks of AES output and $E - 2S$ blocks of RC4 output. Since AES output is indistinguishable from random (otherwise, we would have an attack on AES), the expected value for W_0 is $\alpha \cdot 2S + (1-\beta) \cdot (E - 2S)$.
- When $b = 1$, only S blocks of data are written to the disk, so it contains S blocks of AES output and $E - S$ blocks of RC4 output. The expected value for W_1 is $\alpha \cdot S + (1-\beta) \cdot (E - S)$.

The difference in the expected values of the Hamming weight of R in the two cases is therefore $(1 - \alpha - \beta)S = \delta S$, which corresponds to the S blocks of additional data written to the disk in case 0.

Note that the output of $\mathcal{D}$ on each B-byte block is an independent Bernoulli random variable with one of two possible distributions, depending on whether the block contains output from RC4 or AES (and where we assume the latter is indistinguishable from truly random bytes). Specifically, when a block contains RC4 output, $\mathcal{D}$'s output equals 1 with probability $1 - \beta$, while, when it contains AES output, $\mathcal{D}$'s output equals 1 with probability α. Thus, when $b = 0$, the distribution of W is a Poisson binomial distribution with parameters $p_1 = \ldots = p_{2S} = \alpha$, $p_{2S+1} = \ldots = p_E = 1 - \beta$ (since the trials are independent, the order does not matter). Similarly, when $b = 1$, the distribution of W is a Poisson binomial distribution with parameters $p_1 = \ldots = p_S = \alpha$, $p_{S+1} = \ldots = p_E = 1 - \beta$.

By standard results for the Poisson binomial distribution, the variance of W_0 is equal to $2S\alpha(1-\alpha) + (E-2S)\beta(1-\beta)$, while the variance of W_1 is equal to $S\alpha(1-\alpha) + (E-S)\beta(1-\beta)$. Since $\alpha(1-\alpha), \beta(1-\beta) \leq 1/4$ and $E \leq 4S$, it follows that the two variances are bounded by S. We define $\sigma^2 := S$.

Now, given that W_0 and W_1 are obtained as sums of large numbers of independent Bernoulli random variables, we can consider them to be Normally distributed to a good approximation (here, we assume E is large, as indeed it will need to be to obtain reasonable success rates for our adversary). We already established that their variances are bounded by S, while their means are separated by δS. Note that

$$\begin{aligned}\mathrm{Adv}^{\textit{pl-ot}}_{\mathcal{A}_{B,S,T},\Sigma}(k) &= |\Pr[1 \leftarrow \mathrm{Exp}^{\textit{pl-ot-}0}_{\mathcal{A}_{B,S,T},\Sigma}(k)] - \Pr[1 \leftarrow \mathrm{Exp}^{\textit{pl-ot-}1}_{\mathcal{A}_{B,S,T},\Sigma}(k)]| \\ &= |\Pr[W_0 \leq T] - \Pr[W_1 \leq T]|\end{aligned}$$

where W_0 and W_1 are Normally distributed by assumption. We set T to be half-way between the means; viz $T = E - 1.5S - 1.5S\alpha - (E - 1.5S)\beta$. This finally brings us to a position where we can apply standard tail bounds for Normal distributions in order to estimate the advantage of $\mathcal{A}_{B,S,T}$.

Suppose that we insist that the means of W_0, W_1 are $2n\sigma$ apart for some parameter n; then by our choice of T and under our assumption of Normality (and since σ^2 is an upper bound on the variances of W_0, W_1) we get an advantage for our adversary which is at least

$$1 - \operatorname{erf}\left(\frac{n}{\sqrt{2}}\right)$$

n	$\log_2 S$ (ideal)	$\log_2 S$ (experimental)	$\mathrm{Adv}_{B,S,T}$
1	19.6	20.5	0.683
2	21.6	22.5	0.954
3	22.7	23.7	0.997

Table 1. Number of blocks S required to achieve a given advantage $\mathrm{Adv}_{B,S,T}$ for $B = 4096$.

where $\mathrm{erf}(\cdot)$ is the standard error function for the Normal distribution:

$$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2}\, dt.$$

Hence, under the condition that $\delta S = 2n\sigma = 2nS^{1/2}$, the advantage $\mathrm{Adv}_{B,S,T}$ of our adversary $\mathcal{A}_{B,S,T}$ will be at least the claimed value, namely $1 - \mathrm{erf}(n/\sqrt{2})$. Solving for S, we see that we require

$$S = \frac{4n^2}{\delta^2}$$

to achieve the claimed advantage.

4.1 Concrete Numbers

To launch a concrete attack, we set $B = 4096$, as in [2]. Then, from the results of Section 3 we can take $\delta = 0.002273$ as the optimal setting. Setting $n = 1$ in the above analysis and solving for S, we get $S = 2^{19.6}$ for an advantage of 0.683. Continuing for other values of n, we obtain the second column in Table 1. Thus we see that an adversary with advantage very close to 1 can be achieved by setting $S = 2^{22.7}$, i.e. for an attack involving $2S = 2^{23.7}$ read/write queries on a disk containing a total of $4S = 2^{24.7}$ blocks and $4BS = 2^{36.7}$ bytes of data (i.e. roughly 104 GB of data).

The third column in Table 1 contains estimates for S based on setting $\delta = 0.001648$, the value experimentally observed when setting $\eta = 0$ in the statistical test developed in Section 3.6. The higher numbers in this column reflect the sub-optimal performance of the blockwise distinguisher for this setting of η.

The main computational cost of implementing the attack is that of running the blockwise distinguisher E times; since $E \leq 4S$, the total cost of the attack is on the order of $G_{\max} \cdot B \cdot S$ floating point operations. In our implementation, the running time for a *single* execution of the blockwise distinguisher for $G_{\max} = 64$, $B = 4096$ was 0.000719s (on a Macbook Air with a 1.3 GHz Intel Core i5 processor and 8GB of RAM); the running time of the whole attack with $E = 4S$ and $S = 2^{23.7}$ would then be about 11 hours. This running time could be improved by optimising the choice of η (thus allowing a reduced value of S).

Similar calculations can be carried out with $B = 256$, where we can take $\delta = 0.00055$. We obtain Table 2. We see that S is required to be greater, reflecting the much weaker performance of our blockwise distinguisher when the block size is small. On the other

n	$\log_2 S$ (ideal)	$\mathrm{Adv}_{B,S,T}$
1	23.6	0.683
2	25.6	0.954
3	26.8	0.997

Table 2. Number of blocks S required to achieve a given advantage $\mathrm{Adv}_{B,S,T}$ for $B = 256$.

hand, the total disk size is only moderately increased and the running time of the distinguisher is further reduced (because of the reduced block size).

5 Conclusions

We have shown that the current instantiation of HIVE using RC4 is insecure. Specifically, there is an efficient attack involving $2^{23.7}$ read/write queries on $B = 4096$ byte blocks for which we can construct an adversary having advantage 0.997 in the security model for plausible hiding against one-time adversaries. This violates the main security property claimed for HIVE.

Our work illustrates that if a provably secure scheme is instantiated with an insecure primitive, then all security guarantees may be lost.

HIVE can be repaired by replacing RC4 with a stronger PRG. For example, AES in counter mode (CTR) could be used instead; as well as being more secure, the performance will be better than that of the current RC4-based system on platforms with hardware support for AES. More options, optimized for performance in hard- or software, are offered by the eSTREAM project.[8]

We gave the authors of HIVE the opportunity to make changes to their code and to their paper after notifying them of our initial concerns about their scheme's reliance on RC4. They did not avail themselves of this opportunity. In view of the fact that HIVE had been released to the public and was promoted as "*not rely[ing] on heuristics or obfuscation techniques, but rather strong cryptographic primitives which can be mathematically proven*" and as being able to "*provide very strong security in practice*"[9], we decided to refine and publish our attack to provide a clear demonstration of the shortcomings of instantiating the system with RC4.

Shortly after publication of a pre-print version of this paper, RC4 was replaced by AES-CTR as the default PRG in the HIVE implementation. According to the HIVE developers, I/O performance remains unaffected.[10]

References

1. N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of RC4 in TLS. In *USENIX Security*. USENIX Association, 2013. `https://www.usenix.org/conference/usenixsecurity13/security-rc4-tls`.

[8] `http://www.ecrypt.eu.org/stream/`

[9] Both quotes from `http://hive.ccs.neu.edu/`, date of access 26/10/2014

[10] `http://www.onarlioglu.com/hive/`, section "Change Log"

2. E.-O. Blass, T. Mayberry, G. Noubir, and K. Onarlioglu. Toward robust hidden volumes using write-only oblivious RAM. In *CCS '14*, pages 203–214. ACM, 2014. Full version at `https://eprint.iacr.org/2014/344`.
3. S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In *FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2001.
4. I. Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2005.
5. I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2002.
6. S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. (Non-) random sequences from (non-) random permutations – analysis of RC4 stream cipher. *Journal of Cryptology*, 27(1):67–108, 2014.