Row, Row, Row Your Boat: How to Not Find Weak Keys in Pilsung

Chitchanok Chuengsatiansup
The University of Adelaide
chitchanok.chuengsatiansup@adelaide.edu.au

Eyal Ronen
Tel Aviv University
eyalronen@tauex.tau.ac.il

Gregory G. Rose
Deckard Technologies Inc.
ggr@seer-grog.net

Yuval Yarom
The University of Adelaide
yval@cs.adelaide.edu.au

1 Introduction

The Pilsung cipher is part of the North Korean Red Star operating system, which was leaked to the West in 2014 [1]. The cipher was reverse engineered and analyzed by Kryptos Logic [2], which found that it is based on AES, albeit it uses key-dependent S-Boxes and permutations. In particular, Kryptos Logic reports that the ShiftRows operation in Pilsung "can make weak classes of keys possible, by having permutations that do not change columns at all."

To identify and explore this class of weak keys, we analyzed the cipher and got a better understanding of the ShiftRows permutation in Pilsung. Based on this understanding, we designed highly-efficient code for searching for weak keys. We then used Phoenix, the University of Adelaide's compute cluster, spending thousand of CPU hours to find weak keys. Finally, we tested the keys, and found that due to our confusion about some details of the algorithm, all of our efforts were in vain and no similar class of weak keys exists in Pilsung.

The contributions of this work are:

- We demonstrate how AES-like ciphers that have weak ShiftRows permutations can be attacked. (Section 3.)
- We develop techniques for efficient search of weak keys in such vulnerable ciphers. (Section 4.)
- We highlight the benefits of early verification of results. (Section 5.)

2 Row Your Boat

Pilsung is a block cipher with a substitution permutation network design, based closely on AES. Specifically, the Pilsung state is a 4×4 matrix, represented either as a two-dimensional array or as a 16-byte vector. For encryption, the state is initialized with the plaintext and then it undergoes ten rounds of transformations. Following the Kryptos Logic report, we name these steps after their AES counterparts: SubBytes, ShiftRows, MixColumns, and AddRoundKey. These steps are similar but are not exactly the same as in AES. The most important difference from our perspective is that instead of using a fixed permutation in the ShiftRows step, Pilsung uses a key-dependent permutation.

To generate the permutation, Pilsung uses the Rao-Sandelius shuffle [3, 4], which first "randomly" splits the array into two halves, then recursively shuffles each half. To shuffle 16 bytes we require four levels of shuffle. The randomness for the four levels of shuffle used to generate the permutation in round i is drawn from the corresponding round key RK_i. The randomness for the first and second levels shuffle is taken from the first half of the round key, and the randomness for the third and fourth levels shuffle is taken from the second half of the round key.

Although we use 64 bits of randomness for the first and second (also third and fourth) levels shuffle, we only get $6^4 = 1296$ possible permutations in the first (and third) level, and 256 options in the second (and fourth) level. In total we get a total of $6^4 \cdot 256 \cdot 6^4 \cdot 256 \approx 2^{36.7}$ possible permutations. This is much fewer than the total number of possible permutations $16! \approx 2^{44}$.

3 By the Stream

The Kryptos Logic report notices that replacing the AES ShiftRows with a random permutation may result in a class of weak keys that do not change columns. In this section, we explore the risk and develop distinguishers for such keys.

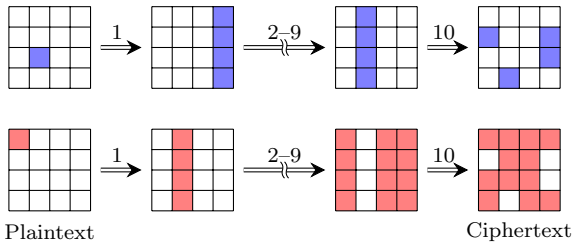

Figure 1: Propagation of a difference with a key that preserves Rounds 2–9 when hitting the preserved column (top) and when missing it (bottom).

We say that a round *preserves* a column i if the ShiftRows permutation moves all of the bytes of column i to a single column j. We further say that a key preserves rounds i to j if there exist $c_i, c_{i+1}, \ldots, c_j$ such that for all $i \leq k < j$, Round k preserves column c_k, moving it to column c_{k+1}.

We now observe that we can easily distinguish a key that preserves rounds 2–9. Suppose we encrypt two plaintexts that only differ in one byte. Figure 1 shows the two possible ways that this difference propagates throughout the encryption. The first round's ShiftRows transformation moves the difference to a new (unknown) location. In the MixColumns the column containing the byte is mixed, resulting in a difference across the whole column. The top half of the figure shows the case that this column is the one that the key preserves. In this case, if the column is the preserved, the difference does not propagate beyond the column, achieving a difference of one column at Round 9. Because Round 10 does not perform the MixColumns transformation, the bytes of the preserved column are permuted, resulting in ciphertexts that differ in at most four bytes. Alternatively, the bottom half of Figure 1 shows the case where the difference is at a column that is not preserved, the difference diffuses across the three non-preserved columns, but does not affect the preserved column.

Either way, after the last round, we get two ciphertexts that have at least four identical bytes. The probability that two random ciphertexts have four identical bytes is

$$2^{-128} \sum_{n=4}^{16} \binom{16}{n} 255^{16-n} \approx 2^{-21.2}$$

Thus such a difference can distinguish between a random permutation and one created by Pilsung with a key that preserves Rounds 2–9.

As Figure 2 shows, we can extend the attack to a key that preserves Rounds 3 to 9. With a probability

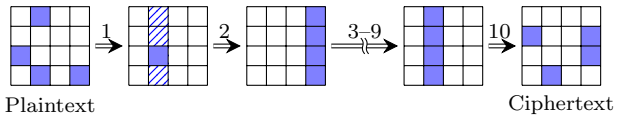

Figure 2: Propagation of a difference with a key that preserves Rounds 3–9.

of 2^{-24}, changing four bytes that all map to a single column in the first round results in a change of a single byte in the second round. If Round 2 shifts the byte to the preserved column, only four bytes of the ciphertext will differ. The probability of selecting four bytes that all go to the same column is one in $\binom{16}{4}/4$. Thus, if we randomly change four bytes, we can expect that approximately one in $\binom{16}{4} \cdot 2^{24}/4 \approx 2^{32.8}$ will result in 12 unmodified ciphertext bytes.[1] We note that better distinguishers exist, but these are outside the scope of this extended abstract

4 Merrily, Merrily

Having established how to exploit weak keys, we now turn our attention to finding them. A quick test with random round keys demonstrates that about one in 682 preserves a specific column. Thus, roughly one in 2^{66} preserves a specific column over rounds 3 to 9, or about one in 2^{64} preserves an arbitrary column. Hence, while not negligible, the class of weak keys is quite small and rare.

Searching 2^{64} keys for a weak key is beyond our modest computational capabilities. However, we note that several properties of the cipher allow us to reduce the search space. First, instead of trying keys at random, we can exploit the structure of the ShiftRows permutation to efficiently find column preserving Round 3 key. Secondly, as Pilsung uses the AES key schedule with five 32-bit words, we can search the space of 2^{32} possible values for the first word of the Round 4 key for a key that preserves Rounds 3–9. Moreover, we find that suitable Round 4 keys are not uniformly distributed. We exploit this by applying a simple heuristic to decide how many combinations of the first word of Round 4 to test.

We now explain how to efficiently find a column preserving Round 3 key. As discussed in Section 2, when generating the ShiftRows permutation, the first two levels of the shuffle distribute the state bytes across the

[1] The probability of choosing appropriate four bytes will be slightly higher if the Round 2 permutation maps more than one byte of the same column to the target column. However, in this case less than three bytes need to remain unchanged.

quarters, whereas the last two levels only move bytes within each quarter. Thus, for the key to preserve a column, the first two levels need to spread the bytes of the preserved column across different rows. By observing the first 64 bits of a key, which determine the two first shuffles, we can rule out candidates guaranteed not to preserve the column.

```
1  while Key not found do
2  |  repeat
3  |  |  RK_3[0,...,63] <-$- {0,1}^64
4  |  until RK_3[0,...,63] can preserve a
   |   column;
5  |  repeat
6  |  |  RK_3[64,...,127] <-$- {0,1}^64
7  |  until RK_3 preserves a column;
8  |  RK_4[0,...,31] <-$- {0,1}^32
9  |  repeat
10 |  |  Expand Key
11 |  |  if Key preserves to Round 9 then
12 |  |  |  break
13 |  |  RK_4[0,...,31] ← RK_4[0,...,31] + 1
14 |  until it's time for a new RK_3;
15 end
```

Algorithm 1: Search for a weak key in Pilsung

Algorithm 1 shows how we search for a key. We first choose the first half of the Round 3 key (Line 3). If the ShiftRows operation with this first half can preserve a column, i.e. it places each of the bytes of a column in a different row, we proceed to select a random second half (Line 6) until we find a round key RK_3 that preserves a column. We then randomly choose the first word of the key of Round 4 (Line 8) and proceed to scan for a key that preserves Rounds 3–9.

The structure of the ShiftRows permutation allows a further optimization. Instead of calculating the ShiftRows permutation, we perform a meet-in-the-middle search. Specifically, for each of the possible $1296 \cdot 256$ permutations in levels 1 and 2 of the shuffle, we record the positions of the bytes of each of the columns it preserves. Similarly, for each of the possible $1296 \cdot 256$ permutations in levels 3 and 4 of the shuffle, we record the positions of the bytes that end up in each of the columns. By matching the positions for the two halves of the shuffle, we can determine whether the source column is preserved and what the destination column is.

The source code for our key search software is available at https://github.com/0xADE1A1DE/PilsungKeySearch.

5 Dream

With an efficient search algorithm, we utilized the Phoenix high-performance cluster at the University of Adelaide to search for a key that preserves rounds 3–9. Because we reuse RK_3 for multiple candidates, the amortized effort for finding a key that preserves Round 3 is negligible, reducing the search space to $682^6 \approx 2^{56.5}$. Our highly efficient search algorithm can explore roughly 2^{25} keys per core per second. Thus the expected search time is about 100 CPU years, which is above our budget. However, we did spend over 10,000 CPU hours and found multiple keys that preserve rounds 3–8.

To test the keys, we modified Pilsung, reducing it to a 9 rounds cipher. We ran the attack on one of the keys, finding to our utter surprise that the attack *fails*. Other keys produced similar results — the attack does not work. We modified Pilsung to output the ShiftRows permutations and found that they do seem to preserve the required columns.

After much head scratching and frustration we found the cause of the failure. The Pilsung code repeatedly shifts between two representations of the internal state. One representation is as a vector of 16 bytes. The other is a square implemented as a two-dimensional array. Unfortunately, the repeated shifts confused us to think that the vector representation uses the *row-first* order, shown in the left part of Figure 3, for storing the state matrix in an array. However, in practice the representation uses the *column-first* order shown in the right part of Figure 3. Consequently, our key search algorithm in Section 4 searches for ShiftRows permutations that preserve rows, rather than columns. While the algorithm is efficient, the security impact of preserving rows is rather dubious – the AES ShiftRows permutation preserves all rows.

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

Row-First

0	4	8	C
1	5	9	D
2	6	A	E
3	7	B	F

Column-First

Figure 3: Matrix Orderings

Further investigation demonstrated that the randomness chosen for the ShiftRows permutation ensures that columns are not preserved. Thus, while we do not claim that there is no class of weak keys in Pilsung, we are

quite certain that the approach in this paper is unlikely to find one.

In retrospect, we should have verified that the attack works much earlier. Had we tried a key that preserves one round on a round-reduced Pilsung, we would have identified the error before spending time and CPU resources on a what in hindsight is a clearly wrong direction. Instead we could have invested the CPU resources into a more profitable target. For example, adding the 10,000 hours to a Bitcoin mining pool would have raised an estimated $7.91, or a whopping $1.97 for each of the authors with three cents to spare.

Acknowledgments

This work was supported by an ARC Discovery Early Career Researcher Award number DE200101577; an ARC Discovery Project number DP210102670; The Blavatnik ICRC at Tel-Aviv University; The Phoenix HPC service at the University of Adelaide; and gifts from Google, Intel, and Robert Bosch Foundation.

Eyal Ronen is a member of Checkpoint Institute of Information Security.

References

[1] Florian Grunow and Niklaus Schiess. "Lifting the Fog on Red Star OS". In: *Chaos Computer Club*. https://www.youtube.com/watch?v=8LGDM9exlZw. 2015.

[2] Kryptos Logic. *A Brief Look At North Korean Cryptography*. https://www.kryptoslogic.com/blog/2018/07/a-brief-look-at-north-korean-cryptography/. July 2018.

[3] C. Randhakrishna Rao. "Generation of Random Permutations of Given Number of Elements Using Random Sampling Numbers". In: *Sankhya: The Indian Journal of Statistics, Series A* 23.3 (Aug. 1961), pp. 305–307.

[4] Martin Sandelius. "A Simple Randomization Procedure". In: *Journal of the Royal Statistical Society. Series B* 24.2 (1962), pp. 472–481.