https://doi.org/10.1093/comjnl/bxac092

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

Chitchanok Chuengsatiansup[1,*], Eyal Ronen[2], Gregory G. Rose[3] and Yuval Yarom[1]

[1]*The University of Adelaide, Australia*
[2]*Tel Aviv University, Israel*
[3]*Deckard Technologies Inc., USA*
*****Corresponding author: chitchanok.chuengsatiansup@adelaide.edu.au*

The Pilsung cipher is part of the North Korean Red Star operating system, which was leaked to the West in 2014. Initial analysis by Kryptos Logic reported a possibility of a class of weak keys due to the use of pseudo-random diffusion. Following this lead, we analyzed the cipher and identified a small class of such weak keys. We developed techniques for searching for a key that belongs to the class. After spending thousands of CPU hours, we found a supposedly weak key for a slightly weaker version of Pilsung, but the key did not behave as we expected. On further investigation we found out a crucial misunderstanding in a critical part of the cipher and that no such class of weak keys exists in Pilsung. Thus, this paper makes two main contributions to the art of cryptanalysis. First, it identifies and shows how to investigate a potential weakness in randomizing diffusion, which although does not exist in Pilsung, may affect future designs. Second, it highlights the need for early verification of results in order to identify errors before expending significant resources.

Keywords: Pilsung; AES; weak keys; differential analysis

Received 10 February 2022; Revised 29 April 2022; Editorial Decision 30 May 2022
Handling editor: Tomer Ashur

1. INTRODUCTION

In 2014 it has been reported that the North Korean operating system, Red Star, was leaked to the West [1]. Investigations of the operating system identified that it is based on Linux, but with some added modules, including three new cryptographic modules, Jipsam1, Jipsam2 and Pilsung. These ciphers were reverse engineered and analyzed by Kryptos Logic [2], which found that they are based on Advanced Encryption Standard (AES) [3], but include some modifications.

The most complex of these is Pilsung, which uses key-dependent S-Boxes and permutations. In particular, the ShiftRows operation of AES is replaced with a pseudo-random permutation, which is selected based on the round key.

The ShiftRows operation plays a key role in ensuring sufficient diffusion in AES. Selecting a random permutation may fail to sufficiently diffuse the state of the cipher and lead to vulnerability. This is also noted in the Kryptos Logic report, which states that the ShiftRows operation in Pilsung 'can make weak classes of keys possible, by having permutations that do not change columns at all.'

In this work, we investigate the potential for weak keys in Pilsung. We first explore possible classes of weak keys. We identify several classes that result in weak ciphers and describe how these can be exploited.

We then proceed to analyze the ShiftRows permutations in Pilsung. We determine how weak keys can be constructed and design efficient algorithms for searching for such keys. We then use Phoenix, the University of Adelaide's computing cluster, spending thousand of CPU hours to find weak keys.

We tested the keys, and found that due to a 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. BACKGROUND

2.1. Pilsung

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, SubBytes and AddRoundKey.

These steps are similar but are not exactly the same as in AES. Below, we outline the functionality of the four steps.

- SubBytes: Each byte of the state is replaced by another byte using a substitution table (S-Box).
- ShiftRows: The 16 state bytes are rearranged according to the randomness derived from the corresponding round key.
- SubBytes: Each column of the state is multiplied by a pre-define 4×4 matrix. This operation is the same as AES, also with the same pre-defined matrix.
- AddRoundKey: Each byte of the state is XORed with the corresponding byte of the round key.

While following the same structure of AES, Pilsung also differs from AES in several important aspects. The key schedule is similar but not identical, in particular, Pilsung uses 160 bits for key material. The S-Boxes are based on but are not identical to the AES S-Boxes. Specifically, for each state byte at each round, Pilsung applies a pseudo-random permutation, which depends on the corresponding byte of the round key, to the output bits of the AES S-Box, yielding a pseudo-random S-Box. Last, and most important for this paper, instead of using a fixed permutation in the ShiftRows step, Pilsung uses a key-dependent permutation.

2.2. Rao–Sandelius shuffle

To generate the permutation, Pilsung uses the Rao–Sandelius shuffle [4, 5], which first 'randomly' splits the array into two equal halves, then recursively shuffles each half. To shuffle 16 bytes, this requires four levels of shuffle. Algorithm 1 outlines this permutation as implemented in Pilsung. Figure 1 shows a visualization of the algorithm.

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.

For the first level, the algorithm divides the 16 bytes of the state into four groups of four bytes, splitting each group equally into the two halves. That is, two of the bytes of each group go to the first half and the other two go to the second half. Overall, there are $\binom{4}{2} = 6$ ways to split four bytes equally. To determine which of these six options to use, the algorithm XORs two key bytes and uses the number modulo 6 to select the permutation. For example, to split the first group in round i, the algorithm XORs byte 0 and byte 4, the second group uses bytes 1 and 5, and so on. The selected permutation is specified as a group of four bits that are fed as 'random' input to the Rao-Sandelius shuffle (array s in algorithm 2). By selecting combinations that have two zeros and two ones, the algorithm guarantees local balance, i.e. that each four bytes are split equally between the halves. We note that because 256 is not a multiple of 6, there is a slight bias in the selection, where permutations number 0 to 3 are more likely to be chosen than permutations number 4 and 5.

Algorithm 1 Rao–Sandelius shuffle

Input: A randomness array r to shuffle $\ell = 2^k$ elements
Output: A random permutation of $[0, \ldots, (\ell-1)]$ *out*

for $i \leftarrow 0$ to $k-1$ **do**
  $d \leftarrow 1 \ll i$ ▷ $\ll$ denotes left shift
  $s \leftarrow 1 \ll (k-i)$
  for $j \leftarrow 0$ to $d-1$ **do**
    $\mathsf{sort}(r[i{\cdot}\ell, \ldots, i{\cdot}\ell{+}\ell{-}1], out[j{\cdot}s, \ldots, j{\cdot}s{+}s{-}1])$ ▷ see Algorithm 2
  end for
end for

For the second level of the permutation, the algorithm divides the state into eight pairs of bytes and uses one random bit to select which byte of the pair goes to the first half and which to the second. Specifically, for pair p, the algorithm uses bit p of byte p of RK_i. As in the first level, the selection is implemented as 'random' two-bit input to the Rao-Sandelius shuffle, maintaining local balance of the permutation.

For the third and fourth levels of the shuffle, the algorithm repeats the process of first and second levels, this time using bytes 8 through 15 from RK_i.

Algorithm 2 Distribution sort: $\mathsf{sort}(s, p)$

Input: An array s containing $n/2$ zeros and $n/2$ ones
Output: A sorted array p according to 0-1 array s

$a \leftarrow 0$
$b \leftarrow 0$
$t \leftarrow$ an array of size n
for $i \leftarrow 0$ to $n-1$ **do**
  if $s[i] = 1$ **then**
    $t[n/2 + a] \leftarrow p[i]$
    $a \leftarrow a + 1$
  else
    $t[b] \leftarrow p[i]$
    $b \leftarrow b + 1$
  end if
  $\mathsf{Copy}(p, t)$ ▷ copy content in t to p
end for

Thus, in each of levels 1 and 3, the algorithm chooses one of 6 permutations for each group of four state bytes, to a total

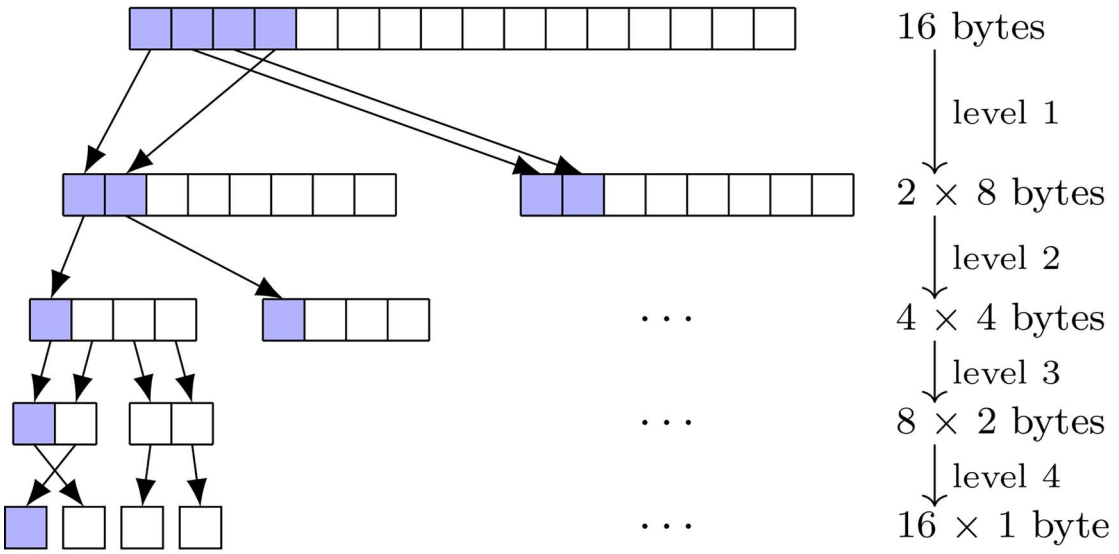

FIGURE 1. Rao–Sandelius shuffle in Pilsung.

of $6^4 = 1296$ possible options. Similarly, in levels 2 and 4, the number of possible permutations is $2^8 = 256$. In total, there are $6^4 \cdot 256 \cdot 6^4 \cdot 256 \approx 2^{36.7}$ possible permutations. This is much fewer than the total number of possible permutations $16! \approx 2^{44}$.

2.3. Related work

Attacks on AES The first attack on reduced round AES has been presented in the original proposal [3]. Since then, a wide variety of attacks have been published. The full AES cipher has been targeted with related-key [6] and Biclique [7] attacks, but they are far from practical. There have been a long line of attacks on 5-round version of AES [8–13], with recent attacks [14, 15] breaking the 2^{32} time complexity barrier. Some practical attacks have been shown on the 6-round reduced variant of AES [10, 16], but the current attacks on 7-round AES [10, 14, 17–19] are not considered practical.

Attacks on Pilsung Genkin et al. [20] investigate the level of protection that using secret S-Boxes and ShiftRows transforms provide against side-channel attacks. Adapting techniques used for attacking AES [21] to Pilsung, they demonstrate that the key can be recovered after monitoring 3.52×10^7 encryptions.

Weak key attacks Weak key classes are rare classes of keys that share a property that makes the cipher more vulnerable to attacks. Arguably the most famous example for weak keys is the 'weak' and 'semi-weak' key classes in DES [22]. The keys in these classes cause the encryption and decryption modes of DES to be identical (potentially between two different keys). Other notable examples of weak keys example in ciphers are IDEA [23–25], the FROG AES candidate [26] and GHOST [27, 28].

3. EXPLOITING WEAK KEYS

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 distinguishing attacks for such keys.

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}.

When a key preserves a column throughout the encryption, i.e. from Round 1 to Round 10, the encryption is clearly distinguishable from a random permutation. Because the ShiftRows is the only operation that diffuses state between columns, such a key basically partitions the state into the preserved column and the rest of the state. Consequently, encrypting two plaintexts that differ by a single byte would yield ciphertexts that have at least one identical column. The probability of this event occurring with a random permutation is extremely small (2^{-32} for each column, which with four columns gives a probability of $\approx 2^{-32} \cdot 4 = 2^{-30}$). Hence we can use such an event to distinguish the cipher from a random permutation.

In practice, we do not even need the key to preserve all of the rounds. We can easily distinguish a key that preserves Rounds 2–9. Suppose we encrypt two plaintexts that only differ in one byte. Figure 2 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 SubBytes 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

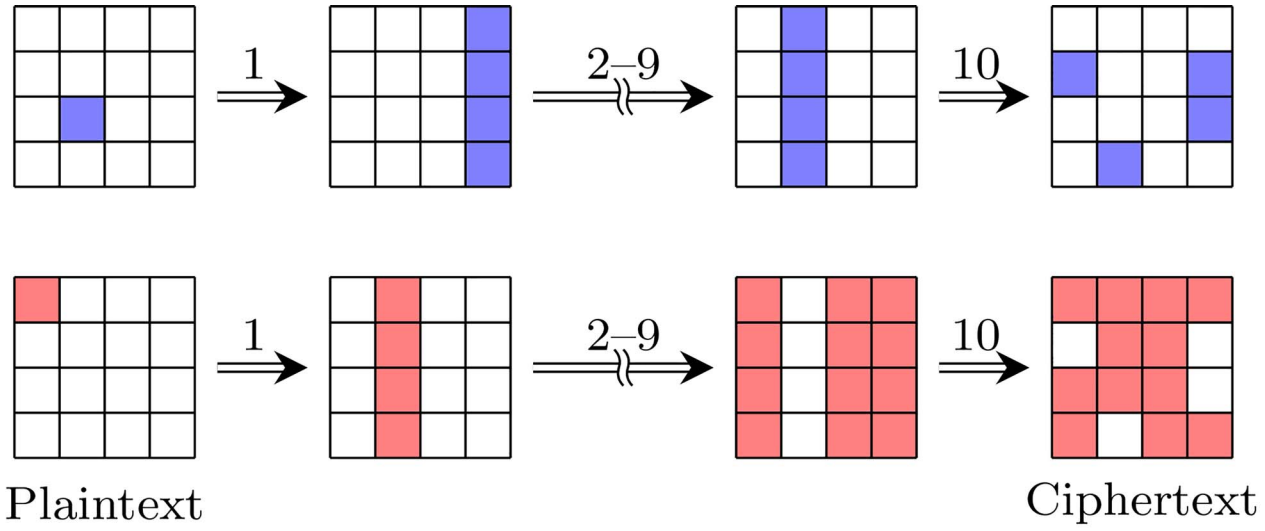

FIGURE 2. Propagation of a difference with a key that preserves rounds 2–9 when hitting the preserved column (top) and when missing it (bottom).

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 SubBytes 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 2 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

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

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 3 shows, we can extend the attack to a key that preserves rounds 3 to 9. With a probability 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 paper.

[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.

Finally, we note that the idea of attacking reduced-round AES by using a plaintext difference to achieve a difference in a single column of an internal round dates back to the original proposal of Rijndael for AES [3]. It would seem that preserving that column may allow extending such attacks to more rounds, and even to the full cipher. We note that it is not clear how such attacks would work on Pilsung, in particular when the S-Boxes are key dependent. We leave investigating this direction to future work.

4. EFFICIENT SEARCH ALGORITHM

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

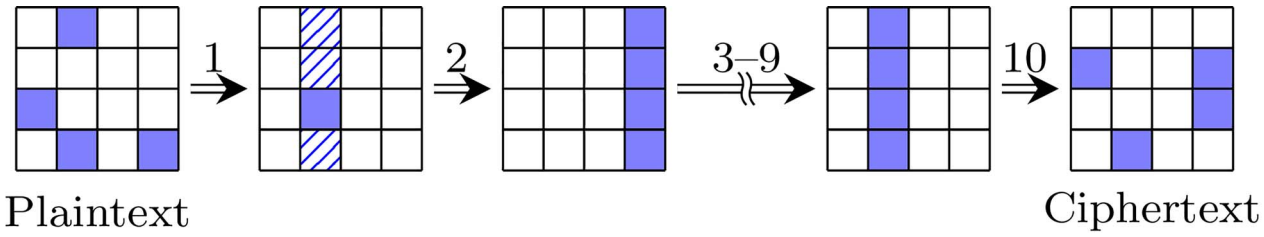

FIGURE 3. Propagation of a difference with a key that preserves rounds 3–9.

distribute the state bytes across the 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.

Algorithm 3 Search for a weak key in Pilsung

1: **while** Key not found **do**
2: **repeat**
3: $\mathrm{RK}_3[0,\ldots,63] \xleftarrow{\$} \{0,1\}^{64}$
4: **until** $\mathrm{RK}_3[0,\ldots,63]$ can preserve a column
5: **repeat**
6: $\mathrm{RK}_3[64,\ldots,127] \xleftarrow{\$} \{0,1\}^{64}$
7: **until** RK_3 preserves a column
8: $\mathrm{RK}_4[0,\ldots,31] \xleftarrow{\$} \{0,1\}^{32}$
9: **repeat**
10: Expand *Key*
11: **if** Key preserves to Round 9 **then**
12: **break**
13: **end if**
14: $\mathrm{RK}_4[0,\ldots,31] \leftarrow \mathrm{RK}_4[0,\ldots,31] + 1$
15: **until** it's time for a new RK_3
16: **end while**

Algorithm 3 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.

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 4. Matrix orderings.

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

5. WEAK KEY SEARCH

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-round cipher. We ran the attack on one of the keys, finding 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 revisiting Pilsung's algorithm 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 created a confusion that the vector representation uses the *row-first*

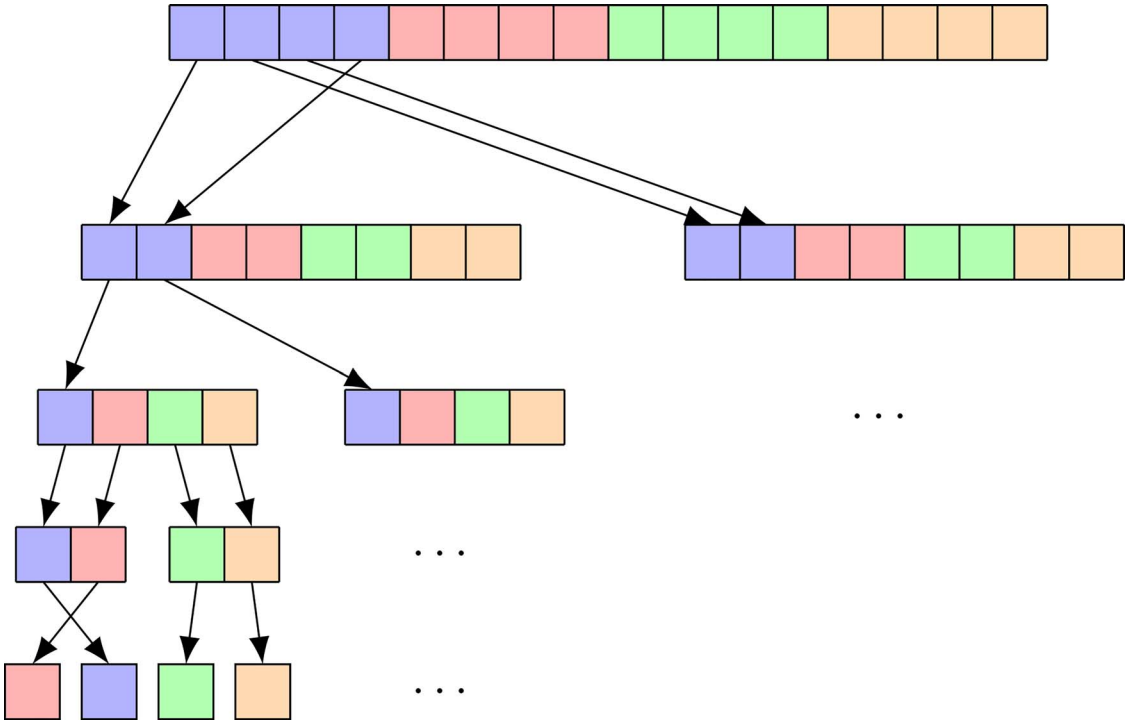

FIGURE 5. The Pilsung ShiftRows permutation ensures no columns are preserved. Four bytes in a column from the previous round (four consecutive blocks with the same color) will spread into one in each column in the following round (four consecutive blocks with four different colors).

order, shown in the left part of Figure 4, 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 4. 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.

Further investigation demonstrated that the randomness chosen for the ShiftRows permutation ensures that columns are not preserved. Figure 5 illustrates how the Pilsung ShiftRows permutation works with column-first order. Before the permutation (top part of Figure 5), the same color represents bytes in the same column. After the first level of shuffle, each of the current first and third column contains exactly two bytes from the previous first (blue) and second (red) columns. At the same time, each of the current second and fourth column contains exactly two bytes from the previous third (green) and fourth (orange) columns. Observe that bytes that came from the same column are placed consecutively next to each other. Therefore, the second level of shuffle, which places consecutive bytes into different groups (i.e. different columns) ensures that columns are shuffled evenly so that after the shuffle, each column contains exactly one byte of each of the original columns.

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.

6. CONCLUSION

In this work, we analyze the North Korean cipher Pilsung. Its deviation from AES by replacing a circular shift with a key-dependent permutation for ShiftRows questions a possibility to have a class of weak keys that preserves columns after the ShiftRows permutation. We design a distinguisher and an efficient algorithm to search for such weak keys. Our analysis reveals that the key-dependent permutation used in ShiftRows guarantees no column-preserve weak keys.

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; gifts from Google, Intel and Robert Bosch Foundation. Eyal Ronen is a member of Checkpoint Institute of Information Security.

REFERENCES

[1] Grunow, F. and Schiess, N. (2015) Lifting the fog on Red Star OS. *Chaos Computer Club*, Hamburg, Hamburg, 27–30 December. CCC, Berlin. https://www.youtube.com/watch?v=8LGDM9exlZw.

[2] Kryptos Logic (2018). *A brief look at North Korean cryptography*. https://www.kryptoslogic.com/blog/2018/07/a-brief-look-at-north-korean-cryptography/.

[3] Daemen, J. and Rijmen, V. (2002) *The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography*. Springer.

[4] Rao, C.R. (1961) Generation of random permutations of given number of elements using random sampling numbers. *Sankhya: Indian J. Stat., Ser. A*, 23, 305–307.

[5] Sandelius, M. (1962) A simple randomization procedure. *J. R. Stat. Soc. Ser. B*, 24, 472–481.

[6] Biryukov, A. and Khovratovich, D. (2009) Related-key cryptanalysis of the full AES-192 and AES-256. In *Proc. Asiacrypt 2009*, Tokyo, Japan, 6–10 December, pp. 1–18. Springer, Heidelberg.

[7] Bogdanov, A., Khovratovich, D. and Rechberger, C. (2011) Biclique cryptanalysis of the full AES. In *Proc. Asiacrypt 2011*, Seoul, South Korea, 4–8 December, pp. 344–371. Springer, Heidelberg.

[8] Derbez, P. (2013) Meet-in-the-middle attacks on AES. PhD thesis, *Ecole Normale SupÃ©rieure de Paris — ENS Paris*.

[9] Tiessen, T. (2016) Polytopic cryptanalysis. In *Proc. Eurocrypt 2016*, Vienna, Austria, 8–12 May, pp. 214–239. Springer, Heidelberg.

[10] Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D.A. and Whiting, D. (2000) Improved cryptanalysis of Rijndael. In *Proc. FSE 2000*, New York, NY, USA, 10–12 April, pp. 213–230. Springer, Heidelberg.

[11] Rønjom, S., Bardeh, N.G. and Helleseth, T. (2017) Yoyo tricks with AES. In *Proc. Asiacrypt 2017*, Hong Kong, China, 3–7 December, pp. 217–243. Springer, Heidelberg.

[12] Biham, E. and Keller, N. (1999) *Cryptanalysis of Reduced Variants of Rijndael*. Unpublished manuscript.

[13] Grassi, L. (2018) Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. *IACR Trans. Symmetric Cryptol.*, 2018, 133–160.

[14] Bar-On, A., Dunkelman, O., Keller, N., Ronen, E. and Shamir, A. (2018) Improved key recovery attacks on reduced-round AES with practical data and memory complexities. In *Roc. Crypto 2018*, Santa Barbara, CA, USA, 19–23 August, pp. 185–212. Springer, Heidelberg.

[15] Dunkelman, O., Keller, N., Ronen, E. and Shamir, A. (2020) The retracing boomerang attack. In *Proc. Eurocrypt 2020*, Virtual, 11–16 May, pp. 280–309. Springer, Heidelberg.

[16] Daemen, J. and Rijmen, V. (2000) Rijndael for AES. In *Proc. AES3*, New York, NY, USA, 13–14 April, pp. 343–348. NIST.

[17] Boura, C., Lallemand, V., Naya-Plasencia, M. and Suder, V. (2018) Making the impossible possible. *J. Cryptology*, 31, 101–133.

[18] Derbez, P., Fouque, P.-A. and Jean, J. (2013) Improved key recovery attacks on reduced-round AES in the single-key setting. In *Proc. Eurocrypt 2013*, Athens, Greece, 26–30 May, pp. 371–387. Springer, Heidelberg.

[19] Gilbert, H. and Minier, M. (2000) A collision attack on 7 rounds of Rijndael. In *Proc. AES3*, New York, NY, USA, 13–14 April, pp. 230–241. NIST.

[20] Genkin, D., Poussier, R., Sim, R.Q., Yarom, Y. and Zhao, Y. (2020) Cache vs. key-dependency: side channeling an implementation of Pilsung. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020, 231–255.

[21] Osvik, D.A., Shamir, A. and Tromer, E. (2006) Cache attacks and countermeasures: the case of AES. In *CT-RSA*, San Jose, CA, USA, 13–17 February, pp. 1–20. Springer, Heidelberg.

[22] Moore, J.H. and Simmons, G.J. (1986) Cycle structures of the DES with weak and semi-weak keys. In *Proc. Crypto 1986*, Santa Barbara, CA, USA, 11–15 August, pp. 9–32. Springer, Heidelberg.

[23] Daemen, J., Govaerts, R. and Vandewalle, J. (1993) Weak keys for IDEA. In *Proc. Crypto 1993*, Santa Barbara, CA, USA, 22–26 August, pp. 224–231. Springer, Heidelberg.

[24] Hawkes, P. (1998) Differential-linear weak key classes of IDEA. In *Proc. Eurocrypt 1998*, Espoo, Finland, 31 May–4 June, pp. 112–126. Springer, Heidelberg.

[25] Biryukov, A. Jr., J, N., Preneel, B. and Vandewalle, J. (2002) New weak-key classes of IDEA. In *Proc. ICICS 2002*, Singapore, 9–12 December, pp. 315–326. Springer, Heidelberg.

[26] Wagner, D., Ferguson, N. and Schneier, B. (1999) Cryptanalysis of FROG. In *Proc. AES2*, Rome, Italy, 22–23 March, pp. 175–181. NIST.

[27] Biham, E., Dunkelman, O. and Keller, N. (2007) Improved slide attacks. In *Proc. FSE 2007*, Luxembourg City, Luxembourg, 26–28 March, pp. 153–166. Springer, Heidelberg.

[28] Kara, O. (2008) Reflection cryptanalysis of some ciphers. In *Proc. Indocrypt 2008*, Kharagpur, India, 14–17 December, pp. 294–307. Springer, Heidelberg.