A distinguisher for reduced-round LUX
By Peter Schmidt-Nielsen

1 Definitions

"Lux" will refer to Lux-256. It will be explained how to extend the attack to Lux-512.

2 Description of Lux

Lux has two portions of internal state. The "core" and the "buffer". The buffer is composed of a 16x4 matrix of bytes. The core is composed of a 8x4 matrix of bytes.

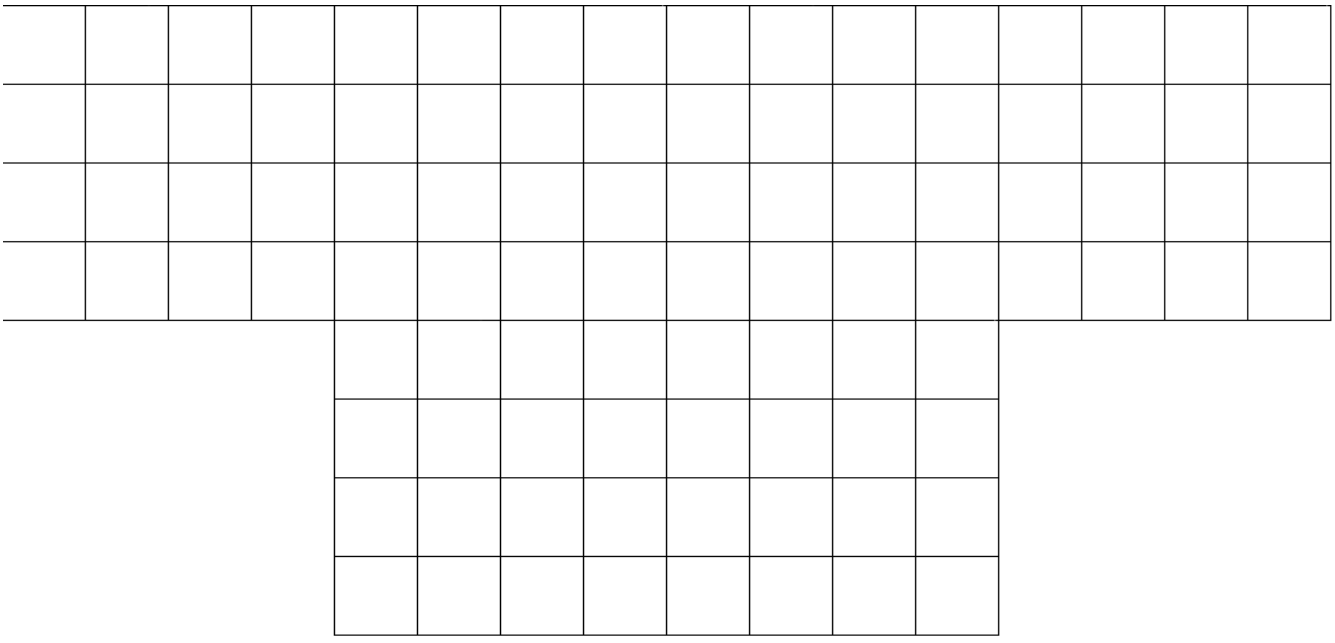
To hash 4 bytes of message *m*:

1) *m* is XORed into the leftmost column of both the core and the buffer.
2) 1 Rijndael round is performed on the core.
3) The core is XORed into the middle of the buffer.
4) The rightmost column of the buffer is XORed into the leftmost column of the core.
5) The buffer is rotated right by one column.

After the message is entirely processed in 4 byte blocks, 3 padding blocks are processed, followed by 16 blank rounds (Rounds in which $m = 0$). Finally, the hash output is extracted from the fourth column of the core, then performing a Rijndael round until the output reaches the desired length.

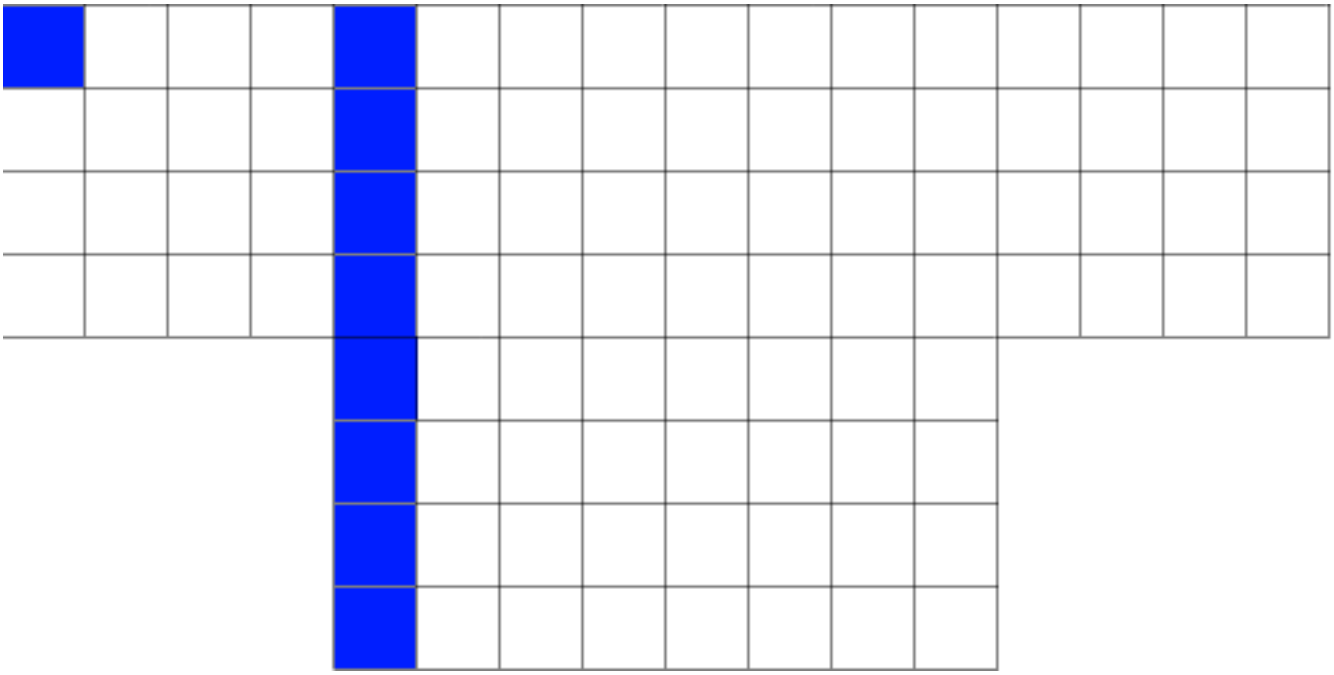
The Rijndael round performed shifts the first row of the core by zero. This means that a change in the upper leftmost byte of the core will only diffuse to the leftmost column of the core after 1 Rijndael round. As only one Rijndael round is performed between message injections, one can set the next message block to exactly cancel out any differential created in the core. This means that one can modify the buffer without affecting the core.(This has also been noted in [1]) This holds with probability one. In addition, any differential put in the buffer will not affect the core until it reaches the rightmost column of the buffer.

We can use this property to distinguish Lux reduced to 8 blank rounds from a random oracle by examining the first 32 bits of the output of Lux. (For brevity's sake, the explanation assumes that you are familiar with Lux.) The following figure will represent the internal state of Lux: The upper portion being the buffer, and the lower portion being the core. It is also crucial to note that the figures here are all snapshots of the state just *before* the buffer is rotated.

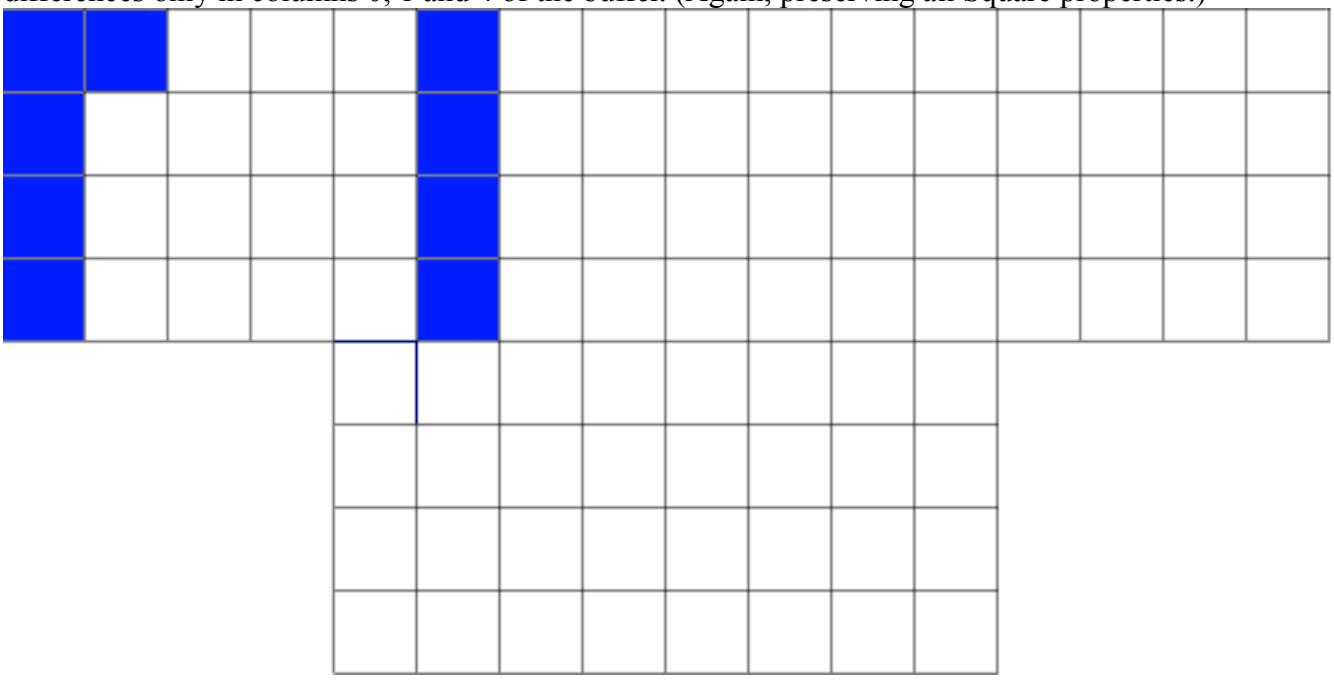


3 Description of distinguisher

1. We produce 256 messages that differ in the first byte of the first message block. After this block is processed, we have differences in column 0 and column 3 of the buffer, and column 0 of the core. This Square property is preserved through the first round, and marked in blue on the figure:

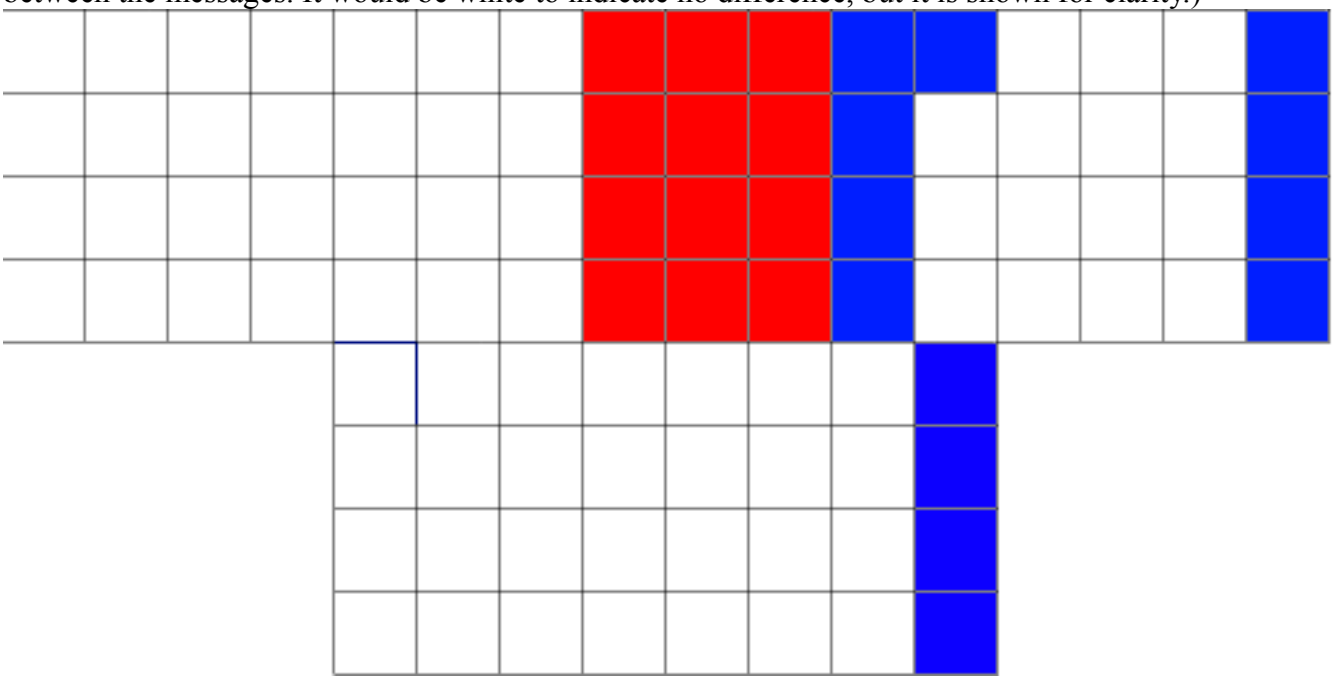


2. We cancel the difference in the core with the second message block. After this round, we have differences only in columns 0, 1 and 4 of the buffer. (Again, preserving all Square properties.)

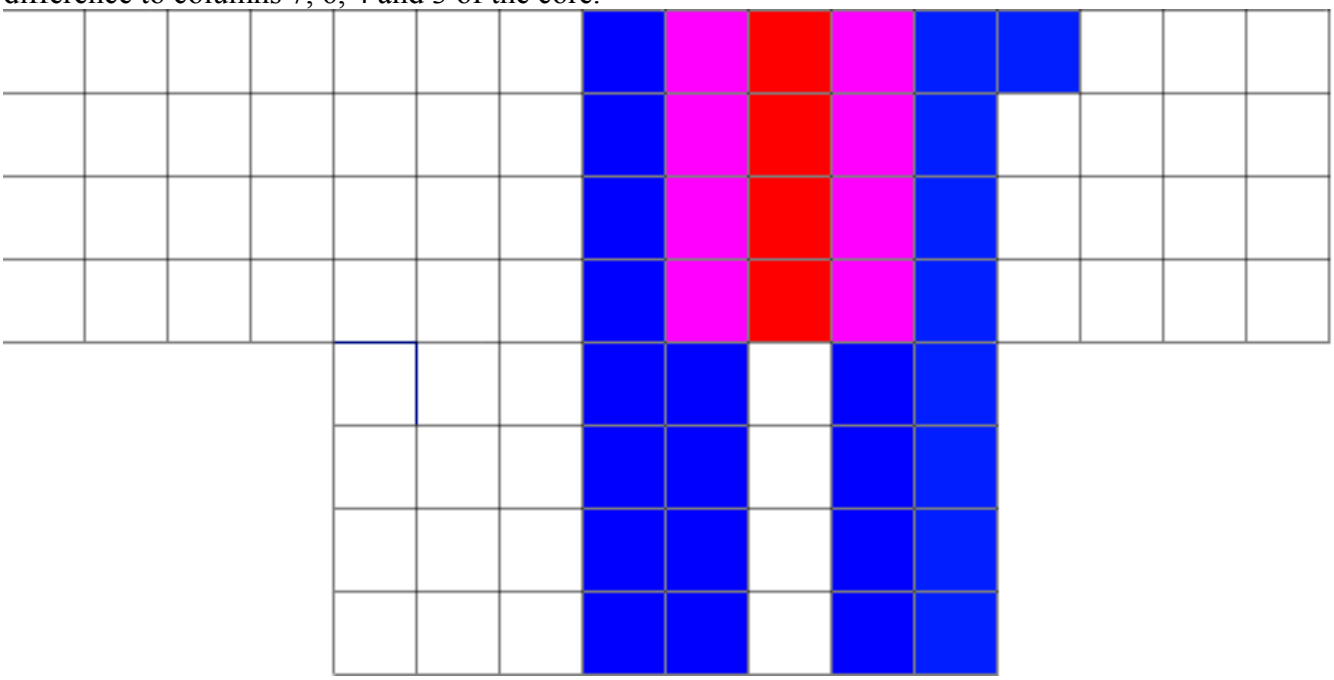


(Continued on next page)

3. Padding is applied: Lux processes 1 block (32 bits) of "10" padding and 2 blocks (64 bits) of length padding. After 7 more blank rounds, we have differences column 7 in the core because of the feedback from the buffer to the core. (Red represents the padding. Note that the padding isn't actually different between the messages. It would be white to indicate no difference, but it is shown for clarity.)

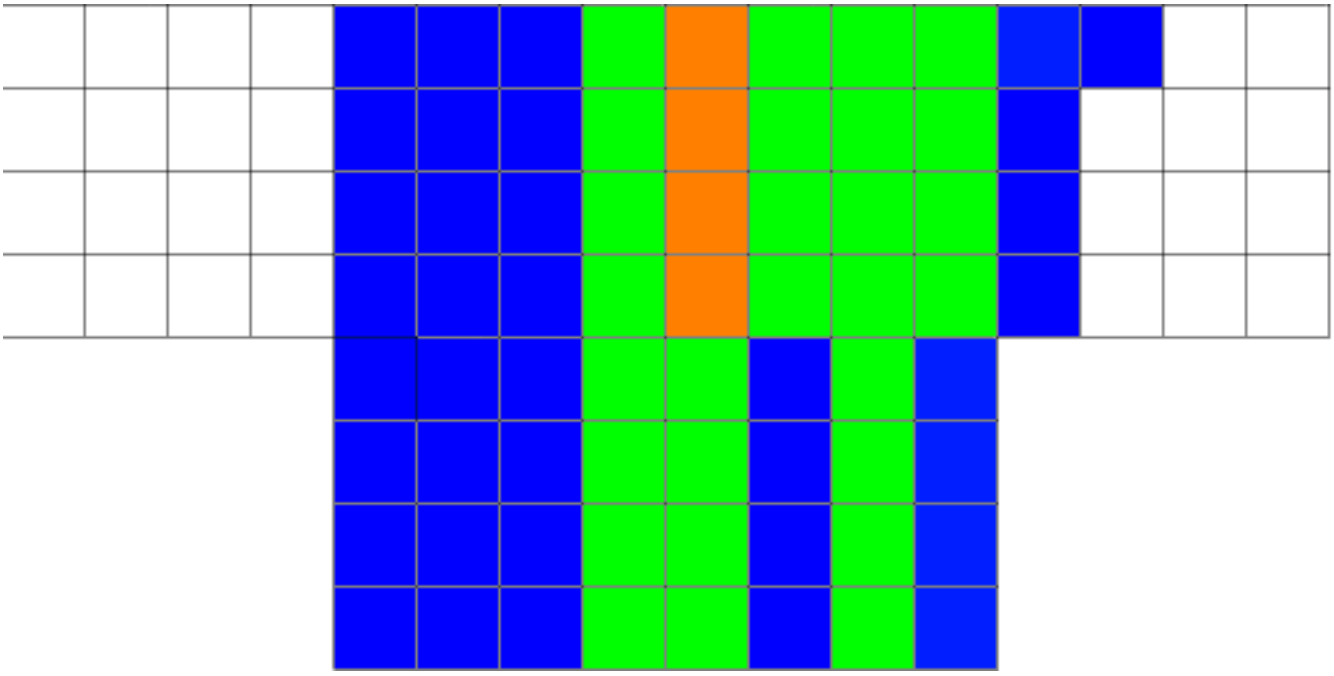


4. The eighth blank round is applied. At this point ShiftRows followed by MixColumns spreads the difference to columns 7, 6, 4 and 3 of the core.



Purple here represents the mix of padding (red) and differences with the Square property (blue). Again, the padding makes no difference, as it is the same for all messages. In is merely shown for clarity. After another few rounds these properties are destroyed, so our attack stops here. Lux now applies the output function to get the hash sum from the internal state. The first output round is processed, after which the first 32 bits of output are taken from column 3 of the core. At this point we have differences all over the place, but the Square property still holds and we have made a distinguisher using 2^8 hash function calls.

Increasing the number of blank rounds from 8 to 9 makes the very simple Square property I am using fail, but more advanced tricks can probably be used to fix it. After that I do not know how many rounds we can manage to get. However, this attack only has a cost so far of 2^8. This leaves a very large "budget" to potentially brute-force the remaining 8 rounds. If, for example, one applies another round, then the state is:



Note: Green here represents a column that is mixed from two blue columns, and orange represents a column that is mixed from three blue columns.

4 Extending to Lux-512

As the Lux-512 state is twice as big, I chose to use Lux-256 for the purposes of keeping the figures to a manageable size. The attack actually works on a greater number of rounds for Lux-512 because there is only 1 block of length padding. Therefore, with Lux-512 one can distinguish up to 9 rounds. This costs no more than the attack on Lux-256, and you still have a large budget to brute-force the remaining 7 rounds.

5 Practical results

Tor E. Bjørstad has verified the attack using the reference implementation of LUX. His program finds a set of input blocks b[1] ... b[255] in the specified way, showing that

8LUX(0x00000000 || 0x00000000) ^ 8LUX(0x01000000 || b[1]) ^ ... ^ 8LUX(0xff000000 || b[255]) = 0x000000001ff64833 8171904ce9def177abc97d30254aac0bc2fb339536202d56

Where ^ represents the XOR operation, and || represents concatenation. Also note that 8LUX represents a weakened Lux-256 with only 8 blank rounds.

The same property holds for LUX-512 reduced to 9 blank rounds:

9LUX(0x0000000000000000 || 0x0000000000000000) ^ ... 9LUX(0xff00000000000000 || *b*[255]) = 0x00000000000000000892678bf6224d2ab546f72371f6f54ab03b4379abc05c31ea5199158790d03e4e10031ba32e3402fa62be68d4a41a7add8aff78b12b56f f8

Note on the figures: These figures have not been produced from actual observation of the internal state, but merely from thinking through the operation of Lux. However, as the attack does actually work by following the instructions as above, the figures are probably correct.

6 References:

1. Wu, Feng, Wu. *Cryptanalysis of the Hash Function LUX-256*