BGE Attack on AES White-Boxes: Extending Blue Galaxy Energy for Decryption and Shuffled States (./bge-attack-on-aes-white-boxes-extending-blue-galaxy-energy-for-decryption-and-shuffled-states.html)

Date Thu 29 February 2024 By Nicolas Surbayrole (./author/nicolas-surbayrole.html) Philippe Teuwen (./author/philippe-teuwen.html) Category Cryptography (./category/cryptography.html) Tags cryptography (./tag/cryptography.html) white-box (./tag/white-box.html) tool (./tag/tool.html) release (./tag/release.html) BGE (./tag/bge.html) 2024 (./tag/2024.html)

We announce the release of a new version of *Blue Galaxy Energy*, our white-box cryptanalysis tool implementing the BGE attack against AES. This version addresses the main limitations of the previous version.

Introduction

In a previous blog post (https://blog.quarkslab.com/blue-galaxy-energy-a-new-white-box-cryptanalysis-open-source-tool.html), we introduced Blue Galaxy Energy, a tool for performing the *BGE attack* against white-box implementations of AES. However, the initial version suffered from some limitations, only supporting encryption white-box implementations with unshuffled, 8-bit encoded intermediate states.

This v2.0 release addresses these limitations by introducing support for:

- **Shuffled intermediary states:** The tool can handle implementations that shuffle the order of intermediate states.
- **Decryption white-box implementations:** We can now analyze implementations that perform decryption operations (in case of shuffling as well).

Support for Shuffled Intermediary States

Using Blue Galaxy Energy requires locating the intermediary states of the white-box through reverse engineering. While the states may be easily accessible in some cases (e.g., on the stack or heap), they can also be stored in registers or obfuscated structures. Additionally, implementations may purposely shuffle the state to hinder key extraction.

So we implemented three extra steps to support the shuffled-state case.

- The first hurdle involved finding a permutation that mimics the byte propagation within an AES round. This was crucial to generate optimized inputs for the attack.
- Next, during the affine parasites recovery step, we extracted the MixColumns coefficients. To determine each coefficient, we associate with each characteristic polynomial the involved MixColumns coefficient. While we need to compute only 4 characteristic polynomials by column to perform the BGE attack, the recovery of MixColumns coefficients needs the computation of 16 characteristic polynomials to fully define each coefficient of a column.
- Finally, we tackled the task of finishing the unshuffling, with some optimizations compared to the literature. This allowed us to reduce the possibilities to a mere 16. The actual key schedule then helped pinpoint the unique correct permutation.

Our tool now supports shuffled states by allowing you to set the `shuffle` parameter to `True` in the `run` method. In this mode, the tool automatically detects the correct byte order of each intermediary state. However, enabling shuffled states support increases the minimum required rounds for a unique key recovery:

- AES-128: 4 rounds (compared to 3 previously)
- AES-256: 5 rounds (compared to 4 previously)

The additional round allows identifying the correct key from potential candidates using the AES key scheduling algorithm. Once the key is found, the `getShuffle()` method provides the correct order of each intermediary state.

In cases where providing the additional round is not feasible, the tool will return 16 key candidates instead of a single key. This allows for further analysis to identify the correct key among the candidates.

For implementation details, please refer to the aptly named file implementation_details.md (https://github.com/SideChannelMarvels/BlueGalaxyEnergy/blob/main/implementation_details.md#shuffled-states), which elaborates on the approach partially based on "Phase 4" described in the paper *Revisiting the BGE Attack on a White-Box AES Implementation (https://eprint.iacr.org/2013/450)* by Yoni De Mulder et al.

Support for Decryption White-Boxes

Decrypting with the BGE attack turned out to be much trickier than expected. We had to rearrange the AES steps to achieve a similar structure for decryption and discovered that a key proposition from the original attack no longer holds true.

The main difficulty stemmed from replacing the SBox with its inverse. This seemingly simple change meant we could no longer uniquely identify specific values at a specific step of the attack.

Nevertheless, we were able to narrow down the possibilities and leverage the key extraction equation to identify the correct ones. Although this process involved a moderate brute-force, it only needs to be done once per decryption whitebox. Overall, the complexity of the BGE attack for decryption is even lower than for encryption due to certain optimizations. These details are explained in implementation_details.md (https://github.com/SideChannelMarvels/BlueGalaxyEnergy/blob/main/implementation_details.md#support-of-decryption).

To enable analysis of decryption implementations, we added a mandatory `isEncrypt` method to the `WhiteBoxedAES` template class.

```
from BlueGalaxyEnergy import WhiteBoxedAES
class MyWhitebox(WhiteBoxedAES):

    def isEncrypt(self):
        # return True if the white-box is an encryption white-box, False otherwise
        return False

    # ... other methods (getRoundNumber, applyRound)
```

Conclusion

This new version of Blue Galaxy Energy significantly expands its capabilities, allowing you to analyze both decryption and shuffled state white-box implementations. These improvements address previous limitations and simplify the process of applying BGE attacks. However, reverse engineering and instrumentation remain necessary to isolate and identify individual rounds within the implementation.

For further information, please refer to the project's README (https://github.com/SideChannelMarvels/BlueGalaxyEnergy/blob/main/README.md).

We encourage you to use Blue Galaxy Energy to analyze white-box implementations with external encodings and share your findings whenever possible.

To update an existing installation to the v2.0 release, simply execute `pip install --upgrade bluegalaxyenergy` .

We welcome feedback, suggestions, and contributions to support additional use cases.

Acknowledgments

We reiterate our gratitude to Laurent Grémy for having developed the core functionality of Blue Galaxy Energy for encryption white-box implementations.

If you would like to learn more about our security audits and explore how we can help you, get in touch with us (https://content.quarkslab.com/talk-to-our-experts-blog)!

Back to top