Dark Phoenix: a new White-box Cryptanalysis Open Source Tool

Date Tue 28 February 2023 By Nicolas Surbayrole Philippe Teuwen Category Cryptography .
Tags cryptography white-box tool release DFA

We are releasing a new cryptanalysis tool based on a known paper but without known open source public implementation so far.

Introduction

For years, we have been maintaining a few white-box cryptanalysis tools in the well-known Side-Channel Marvels set of repositories.

Besides a few very specific attack scripts, the most important tools are the implementations of the *Differential Computation Analysis* (DCA) attack and the *Differential Fault Analysis* (DFA) attack against white-box implementations of AES. The latter was extensively covered in a previous blogpost a few years ago. These tools have the big advantage that they require very few working hypotheses and work *blindly* against white-box implementations, without requiring reverse-engineering. The main hypothesis is to have access to the input or the output of the AES block in clear, as it is for a regular AES.

Even before the existence of these automated attacks, it is well known that a white-box implementation is hard to protect when input or output is not protected. The typical answer is to add so-called *external encodings* on the input and output, which is an extra layer of obfuscation applied on the data before being sent to the AES and removed afterwards. When these external encodings are applied in the same application, it is a matter of reverse-engineering to get to the point where the data are not yet encoded or already decoded.

However, there are a few situations where *external encodings* are not applied locally. For example, in the case of a local secure storage, one might have the data encrypted and decrypted with a local white-box AES, whose input and output are already considered encoded. Since the AES is used with external encodings, it is not the standard AES encryption algorithm anymore. But, as this modified AES is used in isolation, it will not induce any interoperability problem. Nevertheless, in such situations, regular DCA and DFA attacks fail. In this blogpost, we explore a new approach to thwart AES white-box implementations with external encodings applied on their input and output.

Dark Phoenix

The Phoenix became Dark Phoenix due to allowing human emotions to cloud its judgment. In this state, Phoenix was the strongest, but also an evil entity that thirsted for power and destruction. Totally uncontrollable, Dark Phoenix was a force to be reckoned with as it was not bound by a human conscience.

Dark Phoenix is a tool to perform differential fault analysis attacks (DFA) against AES white-boxes with external encodings, as described in *A DFA Attack on White-Box Implementations of AES with External Encodings* by Alessandro Amadori, Wil Michiels and Peter Roelse.

Contrarily to the classical DFA where, in the best conditions, you can break the AES key with just 2 faults, this attack requires more than a million faults! But in a white-box setting, it is not much of a problem and we see hereafter an example where the full attack takes about two minutes.

We first install the tool.

```
$ pip install darkphoenixAES
```

In order to solve some equations, the tool written in Python requires the availability of SageMath on your computer.

To use this tool against a given white-box AES implementation, you need to provide an implementation of your own class inheriting from the provided `WhiteBoxedAES` class. This class is the interface between the white-box and the attack script and it must be able to either introduce a fault at a given position (round and byte) in the white-box or to perform a single round at once and return the intermediate state.

An example is given in the Deadpool repository against the NoSuchCon 2013 white-box. This white-box has the particularity to have external encodings and could not be attacked with classical DCA or DFA. As the NoSuchCon 2013 white-box structure is well understood, it is possible to provide a method that performs a single round at once. Dark Phoenix will then take care of the fault injection by itself.

The corresponding class for NoSuchCon's white-box looks as follows.

```python
from darkphoenixAES import WhiteBoxedAES

class NSCWhiteBoxedAES(WhiteBoxedAES):
    def __init__(self):
        with open("../RE/result/wbt_nsc", "rb") as f:
            # initialize tables based on the white-box file
            self.initSub_sub = ...

    def getRoundNumber(self):
        return 10

    def isEncrypt(self):
        return True

    def hasReverse(self):
        return False

    def apply(self, data):
        for round in range(10):
            data = self.applyRound(data, round)
        return data

    def applyRound(self, data, roundN):
        output=[None]*16
        if roundN < 9:
            for i in range(16):
                b = [0, 0, 0, 0]
                for j in range(4):
                    b[j] = self.roundTables[roundN][i][j][data[j*4+((i+j)%4)]];
                    output[i] = self.xorTables2[(self.xorTables0[(b[0]<<8)|b[1]] << 8) | self.xo
rTables1[(b[2]<<8)|b[3]]]
        else:
            for i in range(16):
                output[i//4 + (i%4)*4] = self.finalTable[i][data[(i&(~3)) +((i+i//4)%4)]]
        return output
```

And running the attack is as simple as this.

```
from darkphoenixAES import Attack
from nosuchcon_2013_whitebox import NSCWhiteBoxedAES

a = Attack(NSCWhiteBoxedAES())
a.run('backup.json')
print("key:", a.getKey().hex())
```

The `backup.json` allows to store intermediate results, which can be handy to avoid running previous steps again when fine-tuning the attack script.

```
$ ./runme.py
key: 4e5343234f707069646123b8dce442d0
```

Faults are first injected one MixColumn before the output, then two MixColumn before it, etc. While the position of the first faults can be found by looking at the output, similarly to the classical DFA, this is not the case for the ones in earlier rounds. If you cannot provide ahead of time an implementation that can inject faults in arbitrary rounds and you need to automate the finding of the right position during the attack itself, a first solution is the following. You can derive your class from another base class `WhiteBoxedAESDynamic`, with an extra method `prepareFaultPosition` that gets two helper functions to check the fault diffusion in the next two rounds. The helper functions allow to check that one faulty byte diffuses to 4 bytes after the next MixColumn and to all 16 bytes after one more MixColumn.

A second mechanism to identify the fault positions is available by using the base class `WhiteBoxedAESAuto` and providing a method `changeFaultPosition` to select a random fault position and associates a tuple (fround, fbytes) to this position. When a fault is asked with `applyFault` with the same tuple, this position should be used. If Dark Phoenix detects that the position is not valid, `changeFaultPosition` is called again, until a valid position is found.

Dark Phoenix supports multiprocessing by default but if this becomes an issue for your class implementation, you might need to disable multiprocessing. See the project README for more information.

Conclusion

Dark Phoenix is provided under the Apache 2.0 license. The source code is available in the Dark Phoenix repository. Have fun using it against other white-box implementations with external encodings, and share your results, whenever it is possible. Feedback and improvements are welcome.

Note that the tool only supports 8-bit wide encodings.

Acknowledgments

Many thanks to Alessandro Amadori for having shared his simulation scripts, which greatly helped us verify our own DFA implementation during its development.

Comments