## NSA Senior Project Proposal

### "Finding Algorithmic Bits in a Binary Haystack"

**Background:** There are many situations in which one would like to determine whether an executable program (e.g., .exe or .dll file) contains an instantiation of a particular algorithm. Sometimes the details of the algorithm itself may be incomplete or unknown – e.g., the algorithm performs matrix multiplication followed by an inversion. In other cases, a complete specification of the algorithm may be available or only a partial specification is provided but with known constants unique to the algorithm – e.g., SHA-1 contains magic values that could be searched for without full knowledge of the hash function.

**Key Research Questions:**

- What types of properties / characteristics of an algorithm are strong indicators of and could be used to identify an instance of an algorithm in executable code?
- How can these properties / characteristics be specified efficiently to an automated tool? What if only partial knowledge of the target algorithm is known?
- What are some methods for finding probable instantiations of a target algorithm in executable code? Evaluate the effectiveness of different approaches when used in isolation and in conjunction with other techniques.
- Can the properties/characteristics be automatically derived if an implementation of the target algorithm is given as input?

**Problem Statement:** Design and develop a solution for finding, with high probability, potential instantiations of a particular algorithm or capability within a binary executable program. The output of the solution should identify possible entry/exit points of the targeted algorithm within the program in a meaningful human readable format. The potential instantiations should be presented in priority order with the most probable displayed first along with a confidence rating for each of the solutions. The user should be able to specify a confidence threshold whereby any potential instantiations of the algorithm found with a confidence rating that falls below the threshold is discarded. The preferred implementation would be as a plugin module for the IDA Pro disassembler, but other disassemblers may be used if desired.

**Project Outline:**

- Design manual (i.e., human involved) processes and methods for identifying intrinsic properties of an algorithm. What are the types of algorithmic characteristics or properties that are likely to be manifested at the binary level, independent of machine format and compiler optimizations? How can a user specify these properties in a reasonable amount of time (i.e., ideally without having to implement the entire algorithm) to an automated tool?

How might an automated tool be constructed that mirrors the manual processes/methods designed above and is capable of identifying the unique properties of an algorithm from source/binary code without human intervention?

- Design automated methods for finding a probable instantiation of an algorithm based upon the intrinsic properties specified (by a user) using the approach designed above. How can the automated tool score the quality or likelihood of the found solution(s) being correct to provide the user with a confidence rating in the results?

- Develop a prototype implementation of your approach (preferably as an IDA Pro plugin module). To scope the solution, assume the target binaries are 32-bit x86 programs. It is recommended to keep all platform specific code in a separate module to make the solution easier to port to other platforms, but not required.

- As a proof-of-concept, select at least two algorithms with differing properties/characteristics and implement them as standalone C/C++ programs. Create additional variants of these programs by compiling with different optimization settings on at least two different compilers.

    - Test your prototype implementation by running the plugin against the disassembly of the standalone program variants. Does the plugin successfully identify the algorithm implementations? Is there any variance in the confidence ratings between different compilers, compiler optimizations, etc? Are there any false positives?

    - Test your plugin by running it against the disassemblies of programs that do not contain the target algorithm. How many false positives are identified? How high is the confidence rating for each wrongly identified algorithm?

    - Test your plugin by running it against the disassemblies of programs that contain both the target algorithm and other unrelated code. Does the plugin correctly identify the target algorithm? How many false positives are identified? What is the difference in confidence rating between the target algorithm and any false positives that are identified?