Formal Review of CTAES for Bitcoin Core

INTRODUCTION: ctaes is a constant-time implementation for AES encryption and decryption. It features pure C code and does not depend on any outside libraries, avoids use of precomputed lookup tables or data-dependent branches, and is based on a purely bit sliced approach. The code is easy to build, use and results in a very small object code.

OVERVIEW: This document describes my approach and findings in formally reviewing the ctaes implementation. The goal was to check that ctaes is not only correct but also resistant to timing-attacks by doing a manual review of the code as well as using applicable automated verification tools. **The entire review includes writing this report and took about 7 hours**.

MANUAL REVIEW

My objective here was to do a manual review of the major components of the AES block cipher based on previous works [1, 2]. From reading the comments/code, I learned that the S-box implementation is based on a depth-16 circuit using XOR, AND and XNOR gates only [1]. The SubBytes function matches the description in the Boyar et al paper (Figures 5-9) [1] where the main difference is that ctaes uses the stack instead of registers used by Kasper et al [2]. Moreover, the ShiftRows/MixColumn (plus Inv*) and AddRoundKey functions on bit-sliced representations indeed follow from [2]. In each layer, the execution time of the logical instructions for SubBytes, ShiftRows, MixColumn, etc are independent of the AES state (or secret key). Consequently, AES key scheduling/expansion, encryption and decryption do not yield any timing variations either. As a result, the ctaes bit-sliced implementation of the AES S-box is resistant to timing attacks.

AUTOMATED REVIEW

My objective here was to apply known formal verification tools to prove that ctaes was correct and secure against known timing attacks.

(1) **Correctness**. Although the AES test vectors included in the test code passed, this alone does not guarantee that the ctaes implementation is correct on all inputs. To prove this, I leveraged Cryptol [3] and the software analysis workbench (or SAW) [4] developed by Galois Inc.

Cryptol is a domain-specific language for formally specifying cryptographic algorithms (e.g., AES, SHA, etc). SAW provides the capability to extract a formal models from C code (via LLVM) and automatically analyze them using SMT/SAT solvers. SAW allows one to verify that the formal model (from C code) matches a specification written in Cryptol. In other words, SAW is capable of showing that a piece of code works on all possible inputs and finding counterexamples when code does not agree with a formal specification.

Using the SAW tool, I was able to prove that the (bit-sliced) ctaes implementation is equivalent to the formal specification of AES (S-box using lookup tables) in Cryptol on all possible inputs.

Therefore, this provides a verifiable proof that ctaes is indeed correct. I have included the SAW script and output as part of this report in Appendix A.

(2) **Resistance to timing-attacks**. The ctaes implementation relies on a bit-slicing approach which by design eliminates cache timing attacks. To verify this, I analyzed ctaes using the FlowTracker tool [5]. FlowTracker detects timing attack vulnerabilities in C/C++ implementations of cryptographic algorithms. FlowTracker is implemented as a LLVM compiler pass and is a flow-sensitive, interprocedural static analyzer. It tracks secret keys in a dependency graph representation of the program. From the graph, it identifies when secret keys determine which parts of the program are executed and identifies code in which memory is indexed by sensitive information. *If interested, please see their paper for the underlying methodology and some examples [5].*

Notably, FlowTracker has been used to evaluate popular cryptographic libraries (including OpenSSL and NaCL). For example, the tool identified potential timing attacks in several OpenSSL functions which led to fixes by the OpenSSL dev team [5]. On the positive side, the tool could not find any vulnerabilities in NaCL's constant-time AES implementation which uses the same bit-slicing techniques described in [2]. Given that ctaes also is based on the same techniques, I thought it would be worthwhile to test FlowTracker on ctaes as well. My goal here was to see what the tool would uncover.

I am glad to report that FlowTracker confirmed that no traces in ctaes (i.e., AES_setup, AES_encrypt and AES_decrypt) would lead to a timing attack. The results are included in Appendix A.

REMARKS: I also re-ran the verification tools on ctaes with MixColumn optimized (https://github.com/sipa/ctaes/pull/3). Note that correctness still holds and no vulnerabilities were found by FlowTracker.

CONCLUSION:
The ctaes implementation is provably correct with respect to all possible inputs for AES_128/AES_192/AES_256 and secure against timing-based attacks. From the manual and automated review, I can conclude that ctaes is verifiably a constant-time implementation of AES.

REFERENCES:
[1] Joan Boyar and Rene Peralta. A depth-16 circuit for the AES S-box. eprint 2011/332.
[2] Emilia Kasper and Peter Schwabe. Faster and Timing-Attack Resistant AES-GCM. eprint 2009/129.
[3] Cryptol: The language of Cryptography. http://www.cryptol.net/.
[4] **SAW**: The Software Analysis Workbench v0.5. By Aaron Tomb et al. http://saw.galois.com/. Galois. 2016. **Background**: SAW leverages symbolic verification to translate code into formal models. During this process it executes code on symbolic inputs, effectively unrolling loops, and

translating code into a circuit representation. Symbolic execution is well suited to verifying code with bounded loops such as cryptographic verification.

[5] **FlowTracker**. Bruno Rodrigues, Fernando Pereira and Diego Aranha. *Sparse Representation of Implicit Flows with Applications to Side-Channel Detection.* In Proceedings of the 25th International Conference on Compiler Construction (CC 2016). ACM, New York, NY, USA, 110-120. DOI=http://dx.doi.org/10.1145/2892208.2892230.

APPENDIX A:

[1] For SAW script, see **ctaes_saw.zip**.
Output in ctaes_saw.txt:

```
Loading module Cryptol
Loading file "ctaes.saw"
Loading module AES
...
Loading sipa/ctaes implementation
Time: 0.078187s
Bitblasting Cryptol implementation
Time: 2.468104s
Checking equivalence (may take about an hour)
Time: 53.080528s
Valid
Writing reference AIG
Time: 2.410691s
```

[2] For FlowTracker output, see **ctaes_flowtracker.zip**.
Output in ctaes_flowtracker.txt:

```
********* Flow Tracking Summary ************
Secrets 1
Branch instructions  or memory indexes 503
Vulnerable Subgraphs: 0
```