RISC V Implementation of SHA-3

Grant Gurvis

***Abstract*—This is the abstract**

I. Introduction

Cryptographic hash functions are a fundamental primitive in modern cryptography, they allow mapping of messages to a digest with no know way to reverse other than brute-force search. Among the most famous hash functions are the Secure Hash Algorithms (SHA) family of hash functions, the most recent standard being the SHA-3 algorithm. SHA-3 is amoung the most secure hash functions

II. Background of Algorithm

The SHA family of algorithms has been the standard hash functions used in most cryptographic applications since they were standardized. SHA-1 and SHA-2 are both based on the Merkle–Damgård construction, which is also used by other hash functions such as MD5. SHA-1 has been shown to no longer be secure several time, in 2015 practical attacks were shown to be possible,[1] in 2017 an actual collision was found,[2] and in 2019 chosen-prefix attacks were shown to be practical.[3] Since the attacks on SHA-1, many systems have migrated to SHA-2, such as TLS and Git. SHA-2 is currently used in wide use but some holes in its security have been shown since it uses similar underlying mathematics as SHA-1. For example all Merkle–Damgård construction are susceptible to length extension attacks and collision attacks on SHA-2 are an active field of research where attacks aren't practical, but have been getting closer. This is not of immediate concern but a transition to SHA-3 would help mitigate potential weaknesses of SHA-2 that are found in the future.

SHA-3 is based on the Keccak algorithm which was designed by Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. It is the first SHA to not be designed by the National Security Agency (NSA) which has led to some increased confidence in its security since the NSA has potentially previously created compromised algorithms.[4] The algorithm was chosen to be the third SHA after it was chosen in a 2012 public competition run by the National Institute of Standards and Technologies (NIST) who maintain many cryptographic and technological standards (including SHA). SHA-3 is known to be slower in software implementations than previous SHAs but it is much faster when implemented via hardware accelerators.

The current standard for SHA-3 was published in 2015.[5]

III. Code Details

This implement

IV. Conclusion

References

[1] M. Stevens, P. Karpman, and T. Peyrin, "Freestart Collision for Full SHA-1," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, vol. 9665, pp. 459–483, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-662-49890-3_18

[2] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The First Collision for Full SHA-1," in *Advances in Cryptology – CRYPTO 2017*, ser. Lecture Notes in Computer Science, J. Katz and H. Shacham, Eds. Cham: Springer International Publishing, 2017, pp. 570–596.

[3] G. Leurent and T. Peyrin, "From Collisions to Chosen-Prefix Collisions Application to Full SHA-1," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, vol. 11478, pp. 527–555, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-030-17659-4_18

[4] N. Perlroth, "Government Announces Steps to Restore Confidence on Encryption Standards," Sep. 2013, section: Technology. [Online]. Available: https://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/

[5] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Aug. 2015, last Modified: 2018-11-10T10:11-05:00. [Online]. Available: https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions

Appendix

Hello there

```
myfunc:
    addi    sp, sp, -8
    sd      ra, 0(sp)
    la      a0, prompt
    mv      a1, t0
    mv      a2, t1
    call    printf
    ld      ra, 0(sp)
    addi    sp, sp, 8
    ret
```