FIPS 202, the SHA-3 Standard

Overview and Recommendations

Michael Powers and Jason Tseng
Cryptographic & Security Testing Laboratory (CSTL)
6841 Benjamin Franklin Drive
Columbia, MD 21046
NVLAP Lab Code: 200427-0

leidos

Intro to Hashing Algorithms

leidos

What is a Hashing Algorithm?

- Wikipedia defines a hash function as *“any function that can be used to map data of arbitrary size to data of fixed size”*
- One-way function
- Small change in message = big change in hash

```
tsengjk@4MS2JQ1 ~
$ echo -n "SHA3 is the best" | openssl md5
(stdin)= cc7318ba200a2c528b046289873ee155

tsengjk@4MS2JQ1 ~
$ echo -n "SHA3 is the best" | openssl sha1
(stdin)= a771684b34873b78d06e687f5ad413aeaeb835ec

tsengjk@4MS2JQ1 ~
$ echo -n "SHA3 is the best" | openssl sha256
(stdin)= cbb4b81c926e6e4c43fde5648cd86aa3de5ae06e9cbb176d2594d6e2a3952b2e
```

Example Usage of Hash Algorithms

- Message integrity and authentication (HMAC)
- Message fingerprinting (plain SHA)
- Data corruption detection (plain SHA)
- Digital signatures (RSA, DSA, ECDSA w/ SHA)
- Conditioning entropy data
- Primitive for random number generation (Hash_DRBG, HMAC_DRBG)
- Primitive for key derivation (SP800-108, SP800-135, SP800-132)

Desirable Security Properties

- **Collision Resistance**
  - Low probability that H(m_1) = H(m_2) (assuming m_1 != m_2)
- **Pre-image Resistance**
  - Given a value h, it is difficult to find a value m such that H(m) = h
- **Second Pre-image Resistance**
  - Given a value m_1, it is difficult to find a value m_2 (assuming m_1 != m_2) such that H(m_1) = H(m_2)
- Note that collision resistance implies second pre-image resistance

History of the Secure Hash Algorithm (SHA)

leidos

“SHA-0”

- The original publication of FIPS PUB 180 in 1993 is often referred to as “SHA-0”
- Designed by the NSA
- Shown to have a complexity of finding a collision in the range of $\mathbf{2^{33.6}}$ operations
- On an “average PC”, would take approximately **1 hour** to find a collision
  - ~~Collision Resistance~~
  - ~~Second Pre-image Resistance~~
- This algorithm is no longer approved/allowed for usage in any security relevant context

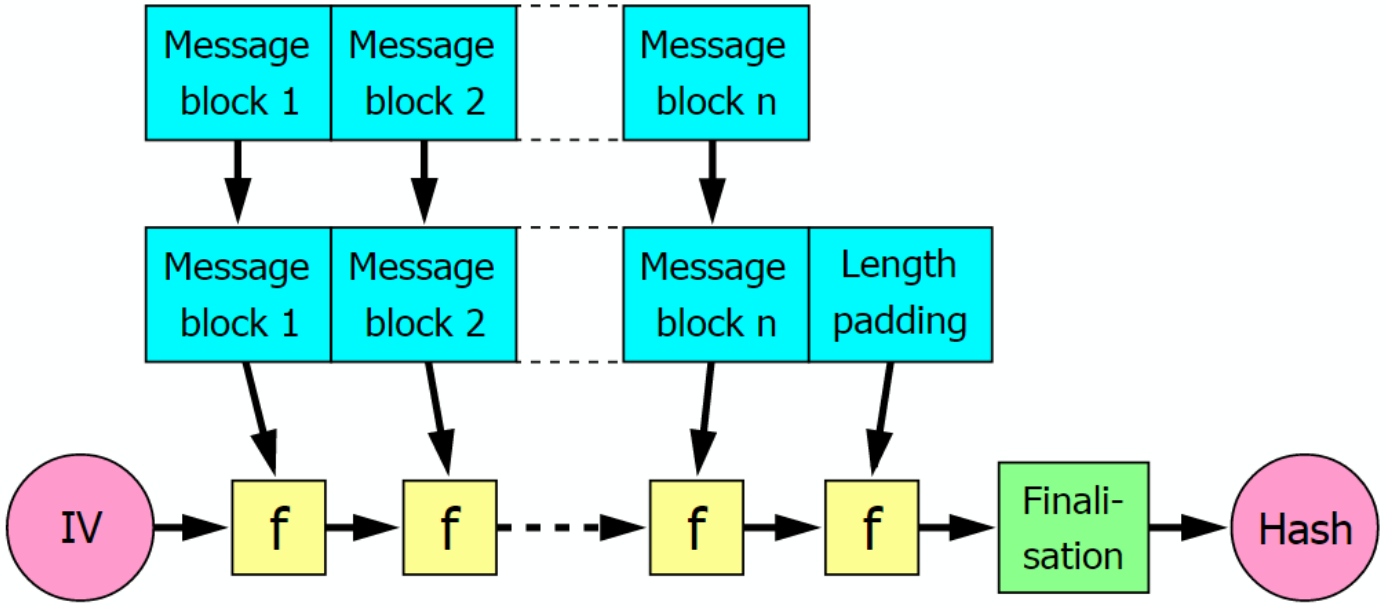
SHA-1

- Introduced in revised publication FIPS PUB 180-1 in 1995
- NSA introduced a “fix” to the original SHA standard, in which a single bitwise rotation was added to SHA-0’s compression function
- Shown to have a complexity of finding a collision in the range of $\mathbf{2^{61}}$ operations. Implies that the SHA-1 algorithm is still providing much less than the ideal **80** bits of security strength.
  - ~~Collision Resistance~~
  - ~~Second Pre-image Resistance~~
- RSA 2016, Adi Shamir mentioned that Marc Stevens working on SHA-1 collision, and that they should be able to find it in the “next few months”
- Marked as **disallowed** or “**legacy**” for usage in digital signatures, as it is not able to provide an adequate amount of security

SHA-2

- Introduced in revised publication FIPS PUB 180-2 in 2001
- Three new hashing algorithms – SHA-256, SHA-384, and SHA-512 (SHA-224, SHA-512/224, and SHA-512/256 would be introduced later)
- Designed by NSA
- No currently-known attacks on **collision resistance** or **pre-image resistance** for the the full-round SHA-2 functions
  - Only on reduced-round variants
- However, vulnerable to **length extension** attacks
  - IF you use the algorithm “incorrectly” and produce an authentication code like:
    - **SHA-2(key || message) = Digest**
  - An attacker, **without knowing the value of ‘key’,** can still calculate valid digests of the form:
    - **SHA-2(key || message || additional_data)**
  - This type of attack is mitigated by the FIPS PUB 198-1 HMAC construction:
    - **HMAC(key, message) = SHA(key xor opad || SHA((key xor ipad) || message))**

Merkle-Damgård Construction



Problem???

- Used in SHA-0, SHA-1 and SHA-2
- Attack against SHA-1 -> possible attack against SHA-2
- Need new hashing algorithm dissimilar from existing SHA algorithms

Description of SHA-3

leidos

The SHA-3 standard

- The publication of FIPS 202 was released in August 2015. This algorithm was originally called Keccak (pronounced “Ketchak”) and was chosen in an open fashion via the **NIST hash function competition**, which included **51** round-1 entries.
- Designed by Guido Bertoni, Joan Daemen (co-designer of Rijndael/ AES), Michael Peeters, and Gilles Van Assche.
- Chosen over:
  - BLAKE
  - Grøstl
  - JH
  - Skein

Possible extensions to the SHA-3 standard

- "**KMAC**" – keyed hash construction, like HMAC, but more efficient and with variable-length output.
  - Will be usable as both a message authentication code (MAC), and as a pseudo random function (PRF)
- "**XMAC**" – Similar to KMAC, but is based on the SHAKE XOFs and cannot be used as a PRF.
- "**XKDF**" – A key derivation function that utilizes XMAC, that could be used in a similar context to the SP800-108 KDFs
- "**TupleHash**" – Hashing a list of strings together to produce a variable length hash.
- "**Fast Parallel Hashing (FPH)**" – Creating a SHA3-based, variable-length hashing algorithm that can be parallelized to benefit from multi-core systems.
- **"Authenticated Encryption (AE)"** – The Keccak specification includes a construction for authenticated encryption, which would provide both confidentiality and integrity. This would be used in a similar context to AES-GCM or AES-CCM modes.

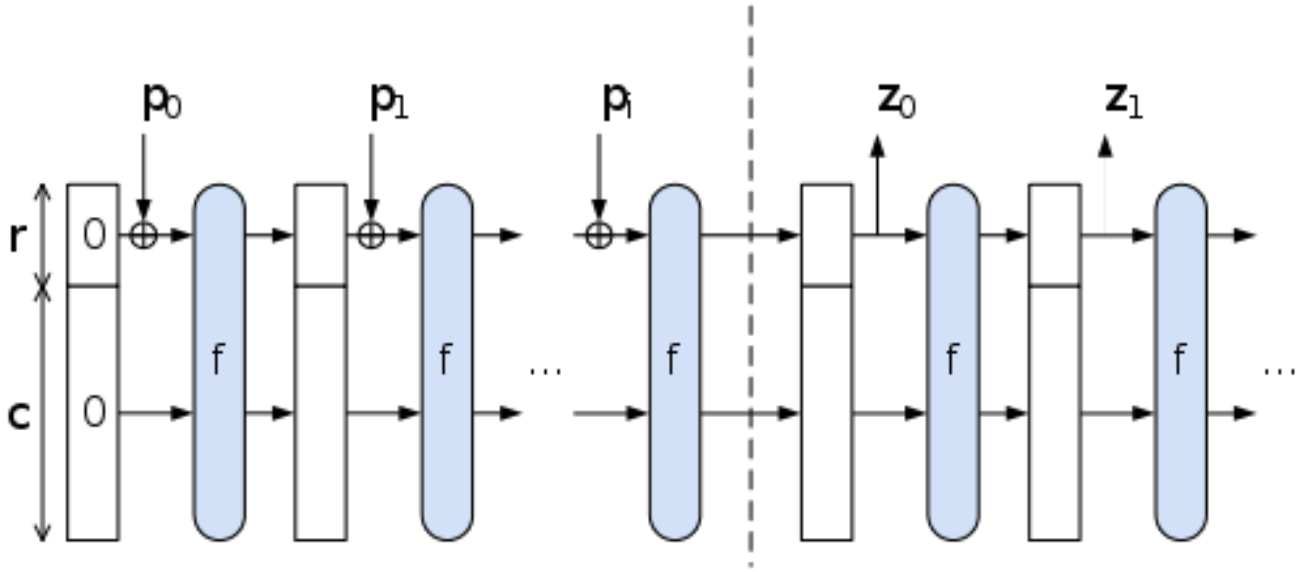
leidos

The “Pros” of SHA-3

- Unlike the other SHA algorithms – SHA-3 does not utilize the Merkle–Damgård construction, but utilizes a Sponge construction. SHA-3 was chosen, in part, due to it being so **vastly different from the other SHA variants**.
  - If a cryptographic break for SHA-2 algorithms is found, it wouldn’t likely apply to SHA-3 (and vice versa)
- Not vulnerable to **length extension attacks**.
- Improved **Second Preimage Resistance** when comparing SHA-256 vs SHA3-256 and SHA-512 vs SHA3-512 (see FIPS 202, section A. 1), especially for longer messages.
- Unlike the other SHA algorithms – SHA-3 was **chosen in an open competition fashion**.

leidos

The Sponge Construction, Illustrated



leidos

Additional “Pros” of SHA-3

- All based on a single “Keccak” function primitive, just with different values provided for **capacity** and **output length**.
  - Trivial to extend the implementation of any one of the SHA-3 algorithms to the others by adjusting a few parameters. **The underlying code does not have to change**.
  - Allows for quick, simple addition of future SHA-3 primitives (e.g.: SHA-3 1024)
  - Speculation: May reduce the number of required power-on self-tests (e.g.: a single self-test for SHA-3 224 could negate any need for having a self-test for SHA-3 256/384/512 due to their similarity)
- Introduces new “extendable output functions” (XOFs), which can be used to produce variable-length outputs. These are called **SHAKE128** and **SHAKE256**.
- Faster in hardware (Anticipated to be **~4X faster** than SHA-2)

leidos

The “Cons” of SHA-3

- Slower in software (~**2X slower** than equivalent SHA-2 algorithms)
- Has not gone through as much **public vetting** as the SHA-2 algorithms
- Unclear right now what the timelines are for the CAVP incorporating SHA3 into current “higher-level” algorithms:
  - For example: HMAC_DRBG, **KAS, RSA, DSA, ECDSA**, SP800-108 KDFs, SP800-135 KDFs
- Unclear right now what the timelines are for the release of the new SHA3 related special publications.
  - For example: KMAC, XMAC, XKDF, TupleHash, FPH, AE

leidos

The “Cons” of the SHAKE algorithms

- “Related outputs” seem to pose a potential security risk
  - SHAKE128(keymaterial, 128) = **ab**
  - Implies SHAKE128(keymaterial, 256) = **abcd**
- Unclear what the timelines are for releasing (potential) standards that incorporate the SHAKE algorithms
  - RSA Full Domain Hash (**FDH**), RSA **OAEP, XKDF**, **Stream Cipher**
- Due to the slower speeds (e.g.: SHAKE128 is about **2 to 3X slower** than other popular stream ciphers), the SHAKE algorithms do not appear to be suitable as stream ciphers.
  - ChaCha20 (from OpenSSL, LibreSSL, BoringSSL, the Linux Kernel) gets approximately **3.91 cycles per byte** in software
  - SHAKE128 gets approximately **9.5 cycles per byte** in software.

leidos

Recommendations

SHA-3 Recommendations for Implementers

- **If your module utilizes SHA-1 or SHA-2 primitives currently**:
  - Implement the SHA-3 and SHAKE algorithms as primitives as well (for redundancy) and get your implementations tested by the CAVP
- **If your module utilizes SHA-1 and SHA-2 in any "higher-level" algorithms:**
  - Implement the SHA-3 variants as well where CAVP testing is available
  - Right now, only testing for HMAC with SHA-3 appears to be available
  - See: http://csrc.nist.gov/groups/ST/toolkit/examples.html
- Please ensure that you **implement self-tests** for each of the SHA-3 variants that you implement.
- Wait for further guidance regarding the SHAKE128 and SHAKE256 algorithms and their intended usage. We recommend against using SHAKE in any security-relevant fashion until it is made clear what their intended use case is.

leidos

SHA-3 Recommendations for Federal Customers

- If choosing between a module that only supports **just SHA-2** and a module that supports **both SHA-2 and SHA-3**, give preference to the one that supports both (for redundancy)
- Don’t give preference to modules that support **SHA-3** but **not SHA-2.** The “3” in SHA-3 is not indicative of an improvement over SHA-1 and SHA-2, and the possibility exists that SHA-3 could be broken before SHA-2.

leidos

Leidos CSTL Contact Information

- Jason Tseng – CSTL Laboratory Manager
  - Jason.K.Tseng@leidos.com
  - +1 (443) 367-7808
- Michael Powers – CSTL Technical Director
  - Michael.C.Powers@leidos.com
  - +1 (443) 367-7422
- www.leidos.com/infosec/testing-accreditation

Sources

- **What Should Be In A Parallel Hashing Standard?**
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/kelsey_sha3_2014_panel.pdf
- **SHA3-based MACs**
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/perlner_kmac.pdf
- **Special Publication on Authenticated Encryption**
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/sonmez-turan_sha3_2014_workshop.pdf
- **Extendable-Output Functions (XOFs)**
  - http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/documents/perlner_XOFs.pdf
- **FIPS 180-4**
  - http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
- **FIPS 202**
  - http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
- **Wikipedia**
  - https://en.wikipedia.org/wiki/Main_Page

Questions?