# An Approximation of a Definitive Survey of Notes on the Future State of Hash Functions; Pre-re-visited Redux Encore

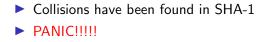
Saqib A. Kakvi

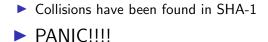
EUROCRYPT 2019 Rump Session

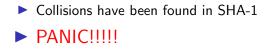


Collisions have been found in SHA-1

Collisions have been found in SHA-1PANIC!!!!









- Collisions have been found in SHA-1
- ► PANHE!!!!!

- Collisions have been found in SHA-1
- ► PANHE!!!!!
- We have other hash functions

- Collisions have been found in SHA-1
- ► PANICIUU
- We have other hash functions
- SHA-2 and SHA-3 are still fine!<sup>1</sup>

<sup>1</sup>I hope! I didn't bother to check but nobody will notice.

- Collisions have been found in SHA-1
- ► PANICIUU
- We have other hash functions
- SHA-2 and SHA-3 are still fine!<sup>1</sup>
- But why should be stick to the SHA family?<sup>2</sup>

<sup>1</sup>I hope! I didn't bother to check but nobody will notice. <sup>2</sup>My SHA-3 candidate was rejected due "lack of rigour"

Create a new family of hash functions<sup>3</sup>.

<sup>3</sup>I'll just make my own hash function family.

- Create a new family of hash functions<sup>3</sup>.
- Encourage the usage of organic primitives.<sup>4</sup>

<sup>3</sup>I'll just make my own hash function family. <sup>4</sup>Are sponges organic? This joke doesn't work if they are.

- Create a new family of hash functions<sup>3</sup>.
- Encourage the usage of organic primitives.<sup>4</sup>
- And range-free<sup>5</sup> primitives.

<sup>3</sup>I'll just make my own hash function family. <sup>4</sup>Are sponges organic? This joke doesn't work if they are. <sup>5</sup>It's like free range, but better! It makes my acronym work!

- Create a new family of hash functions<sup>3</sup>.
- Encourage the usage of organic primitives.<sup>4</sup>
- And range-free<sup>5</sup> primitives.
- Aim for wholesome primitives.<sup>67</sup>

<sup>&</sup>lt;sup>3</sup>I'll just make my own hash function family.

<sup>&</sup>lt;sup>4</sup>Are sponges organic? This joke doesn't work if they are. <sup>5</sup>It's like free range, but better! It makes my acronym work! <sup>6</sup>Again, for the acronym, but hide the details in footnotes! <sup>7</sup>Surely nobody will notice.

- Create a new family of hash functions<sup>3</sup>.
- Encourage the usage of organic primitives.<sup>4</sup>
- ► And range-free<sup>5</sup> primitives.
- Aim for wholesome primitives.<sup>67</sup>
- I think we can all agree these are good things!

<sup>3</sup>I'll just make my own hash function family. <sup>4</sup>Are sponges organic? This joke doesn't work if they are. <sup>5</sup>It's like free range, but better! It makes my acronym work! <sup>6</sup>Again, for the acronym, but hide the details in footnotes! <sup>7</sup>Surely nobody will notice.

W holesome

W holesomeO rganic

- W holesome
- 0 rganic
- **R** ange-free

- W holesome
- 0 rganic
- ${\bf R}$  ange-free
- T rusted

- W holesome
- 0 rganic
- ${\bf R}$  ange-free
- T rusted
- H ash-function

- W holesome
- 0 rganic
- ${\bf R}$  ange-free
- T rusted
- **H** ash-function
  - of

- W holesome
- 0 rganic
- ${\bf R}$  ange-free
- T rusted
- **H** ash-function
  - of
- L arge &

- W holesome
- 0 rganic
- **R** ange-free
- T rusted
- **H** ash-function
  - of
- L arge &
- E xceptionally

- W holesome
- 0 rganic
- **R** ange-free
- T rusted
- H ash-function
  - of
- L arge &
- E xceptionally
- S trong

- W holesome
- 0 rganic
- **R** ange-free
- T rusted
- H ash-function
  - of
- L arge &
- E xceptionally
- S trong
- S ecurity



Block sizes are fixed at 1 million bits for technical reasons.

Block sizes are fixed at 1 million bits for technical reasons.
Hash inputs are padded up to the next million bits.

- Block sizes are fixed at 1 million bits for technical reasons.
- Hash inputs are padded up to the next million bits.
- Security is defined as:

|input| |output|

- Block sizes are fixed at 1 million bits for technical reasons.
- Hash inputs are padded up to the next million bits.
- Security is defined as:

|input| |output|

Security is measured on parts per million (ppm).

- Block sizes are fixed at 1 million bits for technical reasons.
- Hash inputs are padded up to the next million bits.
- Security is defined as:

|input| |output|

- Security is measured on parts per million (ppm).
- ► For high security we need a low ppm, but not too low.

- Block sizes are fixed at 1 million bits for technical reasons.
- Hash inputs are padded up to the next million bits.
- Security is defined as:

#### |input| |output|

- Security is measured on parts per million (ppm).
- ► For high security we need a low ppm, but not too low.
- Optimal ppm = 47.  $21 \le ppm \le 999$  is secure.<sup>8</sup>

#### <sup>8</sup>Detailed explanation & formulae are in the Full Version.

# First candidate: WORTH-LESS-1 ( $\ll_{_1}$ )

• We first measure the Hamming weight of the block  $\omega$ .

# First candidate: WORTH-LESS-1 ( $\ll_1$ )

- We first measure the Hamming weight of the block  $\omega$ .
- ► We compute:

$$\mathtt{WMI} = rac{\omega}{1,000,000}.$$

# First candidate: WORTH-LESS-1 ( $\ll_1$ )

- We first measure the Hamming weight of the block  $\omega$ .
- ► We compute:

$$extsf{WMI} = rac{\omega}{1,000,000}.$$

The output sets are sets of divine numbers D<sub>WMI</sub> chosen using a magical known method from fiction literature.

- We first measure the Hamming weight of the block  $\omega$ .
- ► We compute:

$$\mathtt{WMI} = rac{\omega}{1,000,000}$$

- The output sets are sets of divine numbers D<sub>WMI</sub> chosen using a magical known method from fiction literature.
- ▶ We then pick the *n*<sup>th</sup> element of the set where:

$$n = rac{1}{\omega + e^i} \mod |\mathbb{D}_{\mathtt{WMI}}|.$$

- We first measure the Hamming weight of the block  $\omega$ .
- We compute:

$$\mathtt{WMI} = rac{\omega}{1,000,000}$$

- The output sets are sets of divine numbers D<sub>WMI</sub> chosen using a magical known method from fiction literature.
- We then pick the *n*<sup>th</sup> element of the set where:

$$n = rac{1}{\omega + e^i} \mod |\mathbb{D}_{\mathtt{WMI}}|.$$

The hash of that block is then D<sub>WMI</sub>[n] × π.

- We first measure the Hamming weight of the block  $\omega$ .
- We compute:

$$\mathtt{WMI} = rac{\omega}{1,000,000}$$

- The output sets are sets of divine numbers D<sub>WMI</sub> chosen using a magical known method from fiction literature.
- We then pick the *n*<sup>th</sup> element of the set where:

$$n = rac{1}{\omega + e^i} \mod |\mathbb{D}_{\mathtt{WMI}}|.$$

- The hash of that block is then  $\mathbb{D}_{WMI}[n] \times \pi$ .
- ► The final hash is the concatenation of all the block has values.  $\ll_1 (m) = ||_{i=1}^{\lceil |m|/1,000,000\rceil} \mathbb{D}_{\text{WMI}_i}[n_j] \times \pi_j.$

Security of  $\ll_1$  is between 25ppm and 47ppm.

- Security of  $\ll_1$  is between 25ppm and 47ppm.
- This depends on the input string and choice of divine numbers.

- ► Security of ≪<sub>1</sub> is between 25ppm and 47ppm.
- This depends on the input string and choice of divine numbers.
- This helps to confuse attackers!

- Security of  $\ll_1$  is between 25ppm and 47ppm.
- This depends on the input string and choice of divine numbers.
- This helps to confuse attackers!
- Also weeds out lazy developers!

- Security of  $\ll_1$  is between 25ppm and 47ppm.
- This depends on the input string and choice of divine numbers.
- This helps to confuse attackers!
- Also weeds out lazy developers!
- Only committed devs will implement multiple check algorithms.

- ► Security of ≪<sub>1</sub> is between 25ppm and 47ppm.
- This depends on the input string and choice of divine numbers.
- This helps to confuse attackers!
- Also weeds out lazy developers!
- Only committed devs will implement multiple check algorithms.
- THERE ARE NO COLLISIONS!!! GUARANTEED!!!<sup>9</sup>

<sup>9</sup>Not an actual guarantee. Terms and conditions apply.

Newer, better, cooler, shinier hash function family!

Newer, better, cooler, shinier hash function family!

First candidate, which is super secure!

- Newer, better, cooler, shinier hash function family!
- First candidate, which is super secure!
- More candidates welcome!

- Newer, better, cooler, shinier hash function family!
- First candidate, which is super secure!
- More candidates welcome!
- Full version available at

- Newer, better, cooler, shinier hash function family!
- First candidate, which is super secure!
- More candidates welcome!
- Full version available at

www.NotAFakeWebsite.com/CreditCardBorrower.js.exe.pdf

- Newer, better, cooler, shinier hash function family!
- First candidate, which is super secure!
- More candidates welcome!
- Full version available at

www.NotAFakeWebsite.com/CreditCardBorrower.js.exe.pdf

- Newer, better, cooler, shinier hash function family!
- First candidate, which is super secure!
- More candidates welcome!
- Full version available at

www.NotAFakeWebsite.com/CreditCardBorrower.js.exe.pdf

# THANK YOU!