A Fast Single-Key Two-Level Universal Hash Function

Debrup Chakraborty   Sebati Ghosh   Palash Sarkar

Indian Statistical Institute, Kolkata

7^{th} March, 2017

Outline

1. Introduction
2. Our Contribution
3. Implementation Results
4. Other Contributions

Outline

1. Introduction
2. Our Contribution
3. Implementation Results
4. Other Contributions

Universal Hash Function

- Was introduced by Carter and Wegman in 1979.
- It is an important primitive in cryptography.
- Two main objectives:
    - Reducing the computation time (specially multiplication count)
    - Reducing the key size

scheme	# mult	# sqr	key size
Horner	$\ell - 1$	–	single field element
Bernstein-Rabin-Winograd (BRW)	$\lfloor \ell/2 \rfloor$	$\lfloor \lg \ell \rfloor$	single field element

Table : Univariate polynomial based hashing for message consisting of ℓ blocks for $\ell \geq 3$.

Observation

- BRW polynomials based hash function is advantageous over Horner in terms of operation (field mult.) count.
- Problem is BRW polynomials are inherently recursive; significant implementation overhead for variable length messages.
- If applied on fixed length messages, this difficulty disappear and we can get the benefit of speed.
- Horner can handle arbitrary length messages quite easily.

Objective

- Two-level Hash Function: to combine BRW and Horner to enjoy the benefits of both; apply BRW on fixed length components of the input message and combine the outputs using Horner.
- Use a single field element as the key.
- Propose two-level hash for handling a single binary string (Hash2L) and a vector of binary strings (vecHash2L).
- Optimised implementations of Hash2L over the fields $\mathbb{F}_{2^{128}}$ and $\mathbb{F}_{2^{256}}$.

Outline

1 Introduction

2 Our Contribution

3 Implementation Results

4 Other Contributions

Outline

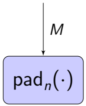
1. Introduction

2. Our Contribution
   - Design
   - Implementation

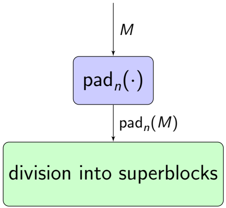
3. Implementation Results

4. Other Contributions

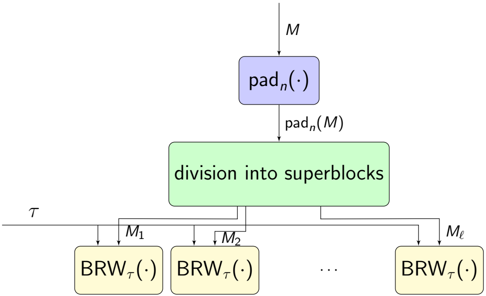
Hash2L: flowchart



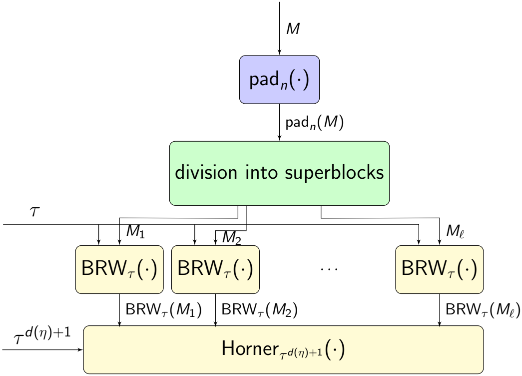
Hash2L: flowchart



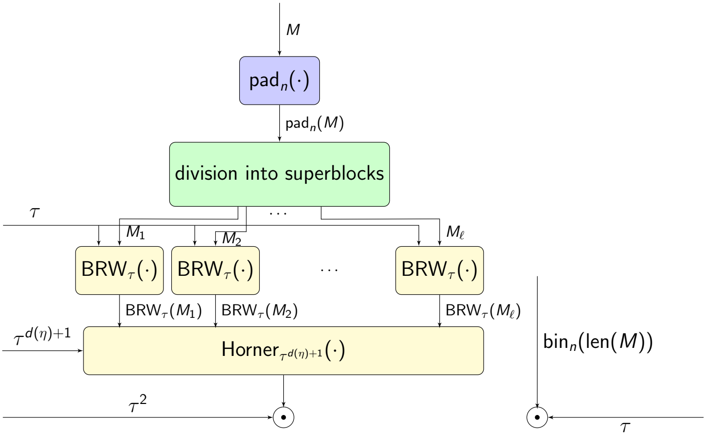
Hash2L: flowchart



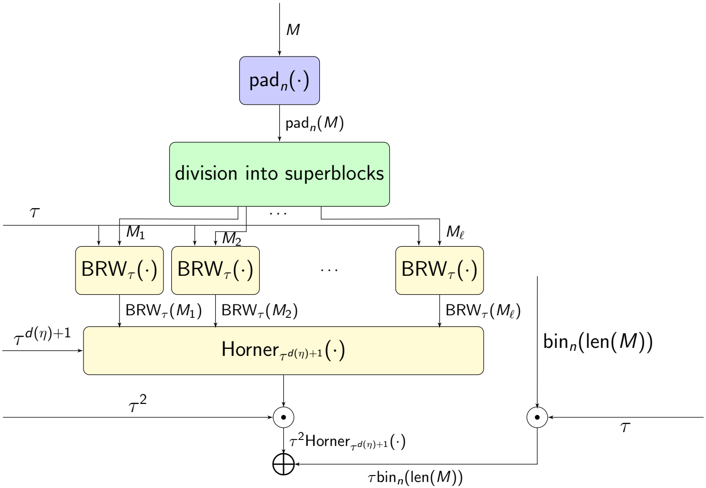
Hash2L: flowchart



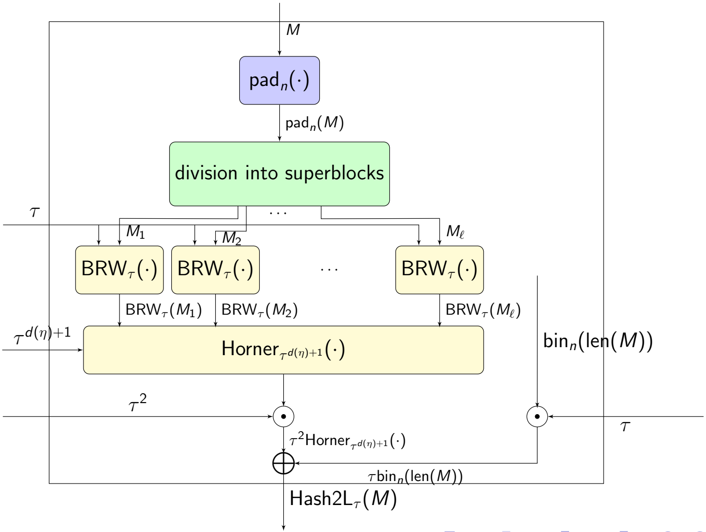
Hash2L: flowchart



Hash2L: flowchart



Hash2L: flowchart



Hash2L: security

- The AXU-bound for Hash2L is $\frac{\ell(d(\eta)+1)+1}{2^n}$ for two distinct messages M and M' with $\text{len}(M) \geq \text{len}(M')$ and ℓ is the number of super-blocks in M. Here, η is the number of blocks in a full super-block.

  Note: The last super-block may be a partial one.

Outline

1 Introduction

2 Our Contribution
- Design
- Implementation

3 Implementation Results

4 Other Contributions

Implementation

- The implementation uses Intel intrinsics, specially the instruction `pclmulqdq`: takes as input two degree 64 polynomials over $\mathbb{F}_2$ and returns their product as degree 128 polynomial.
- Timing measurements on both Haswell and Skylake.

Implementation (contd.)

Some major optimisations:

- Batch size: grouping `pclmulqdq` instructions for m independent multiplications together for better instruction pipelining; we have checked for batch sizes ≤ 4. Finally, we used batch size 3 for $n = 128$ and 1 for $n = 256$ for both BRW and Horner.

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$
\begin{aligned}
&\mathrm{BRW}_\tau(m_1, \ldots, m_{31}) \\
&\quad = \quad \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) \; + \; \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})
\end{aligned}
$$

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\begin{aligned}\mathrm{BRW}_\tau(m_1, \ldots, m_{31}) \\ = \quad \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) \; + \; \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})\end{aligned}$$

normal strategy:

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\mathrm{BRW}_\tau(m_1, \ldots, m_{31})$$

$$= \quad \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) \; + \; \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$$

normal strategy:

field multiplication;
one reduction

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\mathrm{BRW}_\tau(m_1, \ldots, m_{31})$$
$$= \quad \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) \; + \; \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$$

normal strategy:

field multiplication; *one* reduction

one final reduction

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$
\begin{aligned}
&\mathrm{BRW}_\tau(m_1, \ldots, m_{31}) \\
&= \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) + \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})
\end{aligned}
$$

normal strategy:

- $(\tau^{16} + m_{16})$ ⇑ field multiplication; *one* reduction
- $+$ ⇑ XOR the results
- $\mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$ ⇑ *one* final reduction

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\begin{aligned} &\mathrm{BRW}_\tau(m_1, \ldots, m_{31}) \\ &\quad = \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) + \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31}) \end{aligned}$$

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\begin{aligned}\mathrm{BRW}_\tau(m_1,\ldots,m_{31}) \\ = \quad \mathrm{BRW}_\tau(m_1,\ldots,m_{15})(\tau^{16}+m_{16}) \; + \; \mathrm{BRW}_\tau(m_{17},\ldots,m_{31})\end{aligned}$$

Delayed reduction strategy:

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\mathrm{BRW}_\tau(m_1,\ldots,m_{31})$$

$$= \quad \mathrm{BRW}_\tau(m_1,\ldots,m_{15})(\tau^{16}+m_{16}) \;+\; \mathrm{BRW}_\tau(m_{17},\ldots,m_{31})$$

$\Uparrow$

Delayed reduction strategy: only polynomial multiplication; no reduction

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\mathrm{BRW}_\tau(m_1, \ldots, m_{31}) = \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) + \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$$

Delayed reduction strategy:

- $(\tau^{16} + m_{16})$ ⇑ only polynomial multiplication; no reduction
- $\mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$ ⇑ avoid final reduction

Implementation (contd.)

- Using *delayed reduction strategy* for computing BRW Polynomials: for $\eta = 31$, 8 reductions suffice.

$$\mathrm{BRW}_\tau(m_1, \ldots, m_{31})$$
$$= \mathrm{BRW}_\tau(m_1, \ldots, m_{15})(\tau^{16} + m_{16}) + \mathrm{BRW}_\tau(m_{17}, \ldots, m_{31})$$

Delayed reduction strategy:

only polynomial multiplication; no reduction

XOR the results and do *one* reduction on the sum

avoid final reduction

Outline

1. Introduction
2. Our Contribution
3. Implementation Results
4. Other Contributions

Timing Measurements: for $\mathbb{F}_{2^{128}}$

	length of message in bytes			
	512	1024	4096	8192
Hash2L	0.88	0.687	0.498	0.463
GHASH (Gueron)	1.15 (23.5%)	1.02 (32.6%)	0.93 (46.5%)	0.91 (49.1%)
POLYVAL (Gueron)	1.09 (19.3%)	0.81 (15.2%)	0.602 (17.3 %)	0.567 (18.3%)

Table : Cycles per byte for computing Hash2L, GHASH and POLYVAL on **Haswell**.

	length of message in bytes			
	512	1024	4096	8192
Hash2L	0.667	0.468	0.33	0.301
GHASH (Gueron)	0.89 (25.1%)	0.77 (39.2%)	0.67 (50.7%)	0.65 (53.7%)
POLYVAL (Gueron)	0.79 (15.6%)	0.55 (14.9%)	0.369 (10.6%)	0.339 (11.2%)

Table : Cycles per byte for computing Hash2L, GHASH and POLYVAL on **Skylake**.

Timing Measurements: for $\mathbb{F}_{2^{256}}$

	length of message in bytes			
	512	1024	4096	8192
Hash2L	1.4	0.95	0.718	0.67

Table : Cycles per byte for computing Hash2L on **Haswell**.

	length of message in bytes			
	512	1024	4096	8192
Hash2L	1.11	0.758	0.562	0.525

Table : Cycles per byte for computing Hash2L on **Skylake**.

Another measure

According to bit operations per bit of the digest

- - Bernstein and Chou (SAC-2014) report this count for a pseudo-dot product based hash function implementation over $\mathbb{F}_{2^{256}}$, based on the Fast Fourier Transform (FFT) based multiplication algorithm to be 29.
  - But, this figure excludes the cost for generating the long key, which is expected to be significant in a platform not supporting AES-NI instructions.
- - For Hash2L, this cost is at most about 46 for $\eta = 31$.
  - But, in this case there is no hidden cost for generating the key.

Outline

1 Introduction

2 Our Contribution

3 Implementation Results

4 Other Contributions

Appendix

In the paper you can find the following also:

- detailed construction of vecHash2L.
- detailed security proofs for both Hash2L and vecHash2L.
- detail on implementation of field multiplication
- precise counts of arithmetic operations for computing BRW.
- more detail on implementation of BRW.
- analysis of timing measurements obtained.
- detail calculation of bit operations count w.r.t. the SAC-2014 paper of Bernstein and Chou.

Thank You!