การคัดสรร Hash Algorithm โดยการเปรียบเทียบประสิทธิภาพด้านความเร็ว ในการย่อยข้อมูลระหว่าง SHA-1, SHA-256, SHA-512 และ RIPEMD-160 เพื่อใช้พัฒนา Cloud Application
Hash Algorithm Selection using Throughput Measurement Comparison

เด่นเดช สวรรคทัต[1]

Dendej Sawarnkatat[1]

Received: 14 June 2016 ; Accepted: 31 October 2016

บทคัดย่อ

ในการพัฒนาระบบโปรแกรมประยุกต์ที่ต้องมีการบริหารจัดการด้านความมั่นคงปลอดภัยทั้งต่อผู้ใช้และ ปกป้องข้อมูลสำคัญขององค์กรนั้นส่วนสำคัญที่เกี่ยวเนื่องกับบริการเหล่านี้ ต้องอาศัยการใช้งานฟังก์ชัน ทางเดียว (one-way function) ด้วย Hash Algorithm ซึ่งการทำงานของฟังก์ชันเหล่านี้ จะต้องถูก เรียกใช้บ่อยครั้งมาก ยิ่งถ้าหากเป็นการใช้งานระบบงานบนกลุ่มเมฆ (Cloud Application Service) ที่รองรับการใช้งานที่มีจำนวนผู้ใช้พร้อมกันในระดับล้านคนต่อวินาทีขึ้นไปถ้าหากเลือก algorithm ที่ไม่เหมาะสมก็จะยิ่งส่งผลกระทบต่อประสิทธิภาพการทำงานอย่างมากโดยเฉพาะอย่างยิ่งเวลาในการประมวลผล (Processing Time) เพราะว่าระบบเหล่านี้มีการคำนวณอัตราค่าใช้จ่ายตามการใช้งาน หน่วยประมวลผลกลาง (CPU Time Usage Metering) ดังนั้นหากผู้พัฒนาระบบซอฟแวร์ดังกล่าวทราบถึง ประสิทธิภาพอย่างแน่ชัดของ Hash Algorithm ชนิดต่างๆ ย่อมสามารถเลือกใช้ได้อย่างเหมาะสมซึ่งใน เอกสารนี้ผู้วิจัยได้นำเอาเครื่องมือที่ช่วยวัดประสิทธิภาพของ Hash Algorithm ที่เป็นที่รู้จักมานำเสนอพร้อมผลการทดลองที่สามารถนำไปประยุกต์ใช้ได้ทันที

คำสำคัญ: การเข้ารหัสทางเดียว ความมั่นคงปลอดภัย

Abstract

Recently, business enterprises and large organizations worldwide have been migrating their core IT business operations, which once running on premise, to host on cloud platforms in order to reduce cost of ownership and increase return on investment. However, running application on cloud with optimum cost is not as simple as ones' thought because of several factors. One major factor which is the focus of this paper is how to selecting and implementing the efficient algorithms especially the frequently used such as hash algorithms which are part of user registration and authentication functionalities. For this reason, the author has setup the experiment on comparing various hash algorithms in term of execution speed. As a result of this experiment, any developers would readily adopt it as part of development process in no time.

Keywords: cloud application, hash algorithm, SHA-1, SHA-256, SHA-512, RIPEMD-160

บทนำ

สำหรับผู้บริหารด้านเทคโนโลยีสารสนเทศของ องค์กรธุรกิจหรือหน่วยงานขนาดใหญ่หลายๆ แห่งนั้นปัญหาที่มักเกิดขึ้นเป็นประจำคือการตอบข้อ สงสัยเกี่ยวกับความคุ้มค่าในการลงทุนทางด้าน เทคโนโลยีสารสนเทศอันประกอบด้วยระบบฮาร์ดแวร์พื้นฐาน (Hardware Infrastructure) ระบบซอฟแวร์ที่นำมาใช้จนถึงสถานที่ติดตั้งซึ่ง การจัดหาแต่ละครั้งเป็นการลงทุนที่มหาศาล ด้วยเหตุนี้การนำเอา cloud computing เข้ามาใช้แทนการลงทุนซื้อหาและติดตั้งระบบ ทั้งหมดนั้นย่อมช่วยให้องค์กรหรือหน่วยงาน เหล่านี้เข้าถึงระบบที่มีประสิทธิภาพ

[1] *อาจารย์ นักวิชาการอิสระ และ นักศึกษาปริญญาเอก คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ*

[1] *Independence Scholar and Doctoral Student at Faculty of Information Technology King Mongkut's University of Technology North Bangkok email: dendej@gmail.com*

สูงโดยใช้ เงินลงทุนเริ่มต้นที่ต่ำและที่สำคัญสามารถตอบคำถามที่เกี่ยวกับความคุ้มค่าการลงทุนได้โดยง่าย เพราะเป็นการจ่ายตามจริงของการใช้งาน (Pay per use) โดยปราศจากต้นทุนการบำรุงรักษา (maintenance cost) ผลที่ได้คือ หน่วยงานหรือองค์กรสามารถใส่ใจในงานสำคัญ (core business functionality) เป็นหลักคือการพัฒนาระบบสารสนเทศที่มี ประสิทธิภาพ พร้อมความสามารถด้านความมั่นคง ปลอดภัย เช่น การจัดการรหัสผ่านสำหรับยืนยัน ตัวตน (Authentication Password Management) การกำหนดรหัสเฉพาะสำหรับ โปรแกรมประยุกต์บนเว็บ (Web Application Session ID) และการสร้างรหัสการตรวจสอบ ความถูกต้องของข้อมูล (Data Integrity Verification) เป็นต้นซึ่งสิ่งเหล่านี้ล้วน อาศัยเทคโนโลยีการเข้ารหัสทางเดียว (one-way function) ด้วย Hash Algorithm สนับสนุนการดำเนินการทั้งสิ้น อย่างไรก็ตาม Hash algorithm นั้นมีมากมายหลายชนิดแต่ละชนิดก็มีคุณสมบัติ และประสิทธิภาพที่แตกต่างกันทำให้การเลือก Hash algorithm ไปพัฒนาใช้งานร่วมกันระบบ cloud computing ถือว่ามีนัยยะสำคัญเพราะ ปริมาณการใช้งานทรัพยากรชนิดต่างๆ บน cloud เช่น หน่วยประมวลผลกลาง (CPU Usage) หน่วยความจำ (Memory) ขนาดข้อมูลในเครือข่าย (Network Traffic Volume) ล้วนมีการ คิดค่าใช้จ่ายตามจริงทั้งสิ้น การเลือก Hash Algorithm ที่มีความเร็วสูงย่อมสิ้นเปลือง ทรัพยากรเหล่านี้น้อย แต่อย่างไรก็ตามปัจจุบัน Hash Algorithm ที่มีความเร็วสูงหลายชนิด สามารถถูกโจมตีได้ ดังนั้นการเลือกโดยใช้ปัจจัย ด้านความเร็วเพียงอย่างเดียวคงไม่เพียงพอ จึงควรที่จะเลือก Hash Algorithm ที่คุณสมบัติด้านความเร็วและความปลอดภัยสมดุลที่สุดมากกว่า

การเข้ารหัสแบบทิศทางเดียว (One-way Hashing Algorithm)

หมายถึงการใช้งานฟังก์ชันที่มีการประมวลผลข้อมูลเข้าซึ่งโดยปกติไม่จำกัดขนาดแล้วให้ข้อความผลลัพธ์ที่มีขนาดคงที่เสมอ (fixed size message) โดยจะเรียกว่าฟังก์ชันชนิดนี้ว่า Hash function หรือ Hash algorithm ซึ่งหากนำมาใช้งานด้านความมั่นคงปลอดภัย Hash algorithm ที่เลือกใช้ จะต้องมีลักษณะ สำคัญดังนี้

คุณสมบัติพื้นฐานของ Hash Function

- Pre-image attack resistance หมายถึงถ้ามีข้อความใดๆ ที่เป็นผลลัพธ์ จากฟังก์ชันชนิดนี้ จะต้องเป็นการยากที่จะ คำนวณย้อนกลับไปสู่ข้อมูลเข้า (pre-image input) ที่นำมาใช้
- Second pre-image attack resistance เนื่องจากขอบเขตข้อมูลเข้า (input domain) มีขนาดใหญ่กว่าขอบเขตของผลลัพธ์ (output range) ย่อมเป็นไปได้ที่มีข้อมูลเข้าอย่างน้อย สองชุดที่ไม่เหมือนกัน แต่ให้ผลลัพธ์ของ ฟังก์ชันเหมือนกันสำหรับกรณีนี้หมายถึง หากมีข้อมูลเข้าชุดหนึ่งจะต้องเป็นการยาก มากที่จะหาข้อมูลชุดอื่นๆ ที่ไม่เหมือนกันแต่ให้ผลลัพธ์การทำงานของฟังก็ชันที่ เหมือนกัน
- Collision resistance หมายถึงคุณสมบัติของ hash algorithm ที่มีความยากในการหา ข้อมูลเข้ามากกว่าหนึ่งชุดที่ให้ผลลัพธ์ เหมือนกัน

ประเภทการใช้งาน

- การใช้งาน Hash algorithm มีเป้าหมายหลักเพื่อใช้ในการตรวจสอบความถูกต้องของข้อความ (Value checking) ซึ่งการตรวจสอบสามารถ จำแนกออกเป็น 2 ลักษณะคือ
- การตรวจสอบการแก้ไขของข้อความด้วยค่ารหัส (Modification Detection Code) เป็นการนำเอา ข้อความเป้าหมายผ่าน ฟังก์ชันแบบ Hash ซึ่งจะได้ค่ารหัสผลลัพธ์ โดยอาจส่งไปยังปลายทางพร้อมกับข้อความ เพื่อที่ผู้รับจะได้ทำการพิสูจน์ความถูกต้องโดยการนำเอาข้อความที่ได้รับไปผ่านฟังก์ชันแบบ Hash ซึ่งหากผลลัพธ์ที่ได้ตรงกันก็เป็นเครื่องยืนยันว่าข้อความดังกล่าวมาจากต้นทางจริงและไม่ได้ถูกดัดแปลงแก้ไขใดๆ
- การสร้างรหัสพิสูจน์ข้อความ (Message Authentication Code) มีลำดับการทำงาน ใกล้เคียงกับ MDC แต่นำเอารหัสผลลัพธ์ (Hash Result) ไปผ่านการเข้ารหัสแบบอสมมาตร (Asymmetric encryption) ด้วยกุญแจส่วนตัว (private key) ของผู้ส่ง เพื่อสร้างเป็นลายมือชื่ออิเล็กทรอนิกส์ (Digital Signature) และเมื่อผู้รับปลายทาง ได้รับข้อความก็นำข้อความนั้นไปผ่าน ฟังก์ชันแบบ Hash เพื่อให้ได้ค่ารหัสผลลัพธ์ จากนั้นทำการถอดรหัสลายมือชื่อ อิเล็กทรอนิกส์ด้วยกุญแจสาธารณะ (public key) ของผู้ส่งแล้วนำไปเปรียบเทียบกับรหัส ผลลัพธ์หากเหมือนกันก็แสดงว่าข้อความดังกล่าวมาจากผู้ส่งจริงเพราะปราศจากการแก้ ไขดัดแปลง

SHA-1

เป็นอัลกอริทึมที่มีการทำงานแบบฟังก์ชันทิศทางเดียวที่สามารถรับข้อมูลเข้าที่มีขนาดต่างๆกัน เพื่อทำการบีบอัดข้อมูล (compression) ซึ่งผล ลัพธ์ของการทำงานจะเป็นข้อความที่ย่อย แล้ว (digested message) ที่มีขนาดคงที่ 160 บิท (20 ไบท์) มีการทำงานแบ่งออกเป็น 2 ขั้นตอน คือ การเตรียมการ (pre-processing) และการคำนวณ ค่า hash (hash computation) ดังนี้ (จาก SHA-2 Standard[8])

ขั้นตอนที่ 1 การเตรียมการโดยแบ่งข้อมูลเข้าออกเป็นชุดๆ ละ (block) ขนาด 512 บิท โดยข้อมูลชุดสุดท้ายจะมีการเติมเต็มต่อท้าย ซึ่งการเติมจะใส่บิทแรกเป็น 1 เพียงบิทเดียวเท่านั้นและตามด้วยบิทที่เป็น 0 ไป เรื่อยๆ จนกว่าจะมีขนาดเท่ากับ 448 บิท จากนั้น เพิ่มข้อมูลขนาด 64 บิทต่อท้ายในข้อมูลชุดสุด ท้าย ซึ่งเป็นความยาวของข้อมูลเข้า (ถ้าข้อมูลเข้ามีขนาดมากกว่า 2^{64} ก็จะใช้ค่าเศษของผลหาร ขนาดข้อมูลด้วย 2^{64}) จนทำให้ข้อมูลชุดสุดท้าย มีขนาด 512 บิท (448 +64) เหมือนกันชุดอื่นๆ

ขั้นตอนที่ 2

ขั้นตอนการคำนวณค่าhashเริ่มจากข้อมูลชุดแรกจนถึงชุดสุดท้ายเป็นลำดับโดยเมื่อประมวลผลข้อมูลแต่ละชุด เช่นชุดที่ B^i จะต้องคำนวณหาค่า hash H^i ซึ่งได้มาจากการคำนวณฟังก์ชันตรรกะ $f_t(b,c,d)$ ร่วมกับค่า W_t (message schedule) โดยสามารถแบ่งออกเป็น 4 ขั้นย่อยได้ดังนี้

1) กำหนดเงื่อนไขของ W_t (t คือลำดับของรอบที่ทำในขั้นที่ 3)

$$W_t = B_t^i, \quad 0 \le t \le 15$$

$$W_t = ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}),$$

$$16 \le t \le 7$$

หมายเหตุ: $ROTL^y(x)$ คือการหมุนบิท (rotate) ของ x ไปทางซ้ายจำนวน y บิท

2) เตรียมข้อมูลและฟังก์ชันที่จำเป็น โดยกำหนดค่าเริ่มต้นในการคำนวณจำนวน 5 ค่า (a,b,c,d,e) โดยใช้ค่า hash จากรอบก่อนหน้านี้ H_0^{i-1} จนถึง $H_4^{i-1}(a = H_0^{i-1}, \ldots, e = H_4^{i-1})$ ยกเว้นรอบแรกจะ กำหนดค่าเริ่มต้นดังนี้คือ

$$H_0^0 = 67452301_{16}, H_1^0 = EFCDAB89_{16},$$

$$H_2^0 = 98BADCFE_{16}, \quad H_3^0 = 10325476_{16},$$

$$H_4^0 = C3D2E1F0_{16}$$

ค่าคงที่ K_t ขนาด 32 บิท

$K_t = 5A827999_{16}$ เมื่อ $0 \le t \le 19$,

$K_t = 6ED9EBA1_{16}$ เมื่อ $20 \le t \le 39$,

$K_t = 8F1BBCDC_{16}$ เมื่อ $40 \le t \le 59$, $K_t = CA62C1D6_{16}$ เมื่อ $60 \le t \le 79$

ฟังก์ชันตรรกะ $f_t(b,c,d)$ ดังนี้

$$f_t(b,c,d) = \begin{cases} (b \wedge c) \vee (b \wedge d), & 0 \le t \le 19 \\ (b \oplus c \oplus d), & 20 \le t \le 39 \\ (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d), & 40 \le t \le 59 \\ (b \oplus c \oplus d), & 60 \le t \le 79 \end{cases}$$

3) ทำการคำนวณแบบวนซ้ำ 80 รอบ ($t \to [0 \ldots 79]$)

$$T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t,$$

$$e = d,\ d = c,\ c = ROTL^{30}(b),\ b = a,\ a = T$$

4) คำนวณค่าhashข้อมูลชุดปัจจุบัน H_0^i จนถึง H_7^i โดยการบวกแบบ modulo-32 ดังสมการ ต่อไปนี้

$$H_0^i = a + H_0^{i-1}, H_1^i = b + H_1^{i-1}, H_2^i = c + H_2^{i-1},$$

$$H_3^i = d + H_3^{i-1}, H_4^i = e + H_4^{i-1}$$

ซึ่งเมื่อคำนวณค่าhash ของข้อมูลชุดสุดท้ายเสร็จ ก็จะได้ค่า H_0^n ถึง H_7^n จึงนำมาสร้างค่า hash ผลสำเร็จ (output) คือ $H = H_0^n \| H_1^n \| H_2^n \| H_3^n \| H_4^n$ โดย ‖ คือการนำค่ามาเรียง ต่อกัน

SHA-256 และ SHA-512

เป็นอัลกอริทัมแบบฟังก์ชัน ทิศทางเดียวที่ถูกคิด ค้นเพื่อมาใช้งานแทน SHA-1 ที่ปัจจุบันพบว่าไม่ ปลอดภัยจากการโจมตีด้วย pre-image attack สำหรับการทำงานของ SHA-256 และ SHA-512 จะใช้การบีบอัดข้อมูล โดยให้ผลลัพธ์ของการทำงานเป็นข้อความที่ย่อยแล้วที่มีขนาดคงที่คือ 256 บิท (32 ไบท์) สำหรับ SHA-256 และ 512 บิท (64 ไบท์) สำหรับ SHA-512 ส่วนการ ทำงานแบ่งออกเป็น 2 ขั้นตอนเหมือนกับ SHA-1 คือ การเตรียมการและ การคำนวณค่าhash ดังนี้ (จาก S. Wan-zhong et al.[7]และ SHA-2 Standard[8])

ขั้นตอนที่ 1

แบ่งข้อมูลเข้าออกเป็นชุดซึ่ง SHA-256 แต่ละชุดจะมีขนาด 512 บิท ส่วน SHA-512 แต่ละชุดจะ มีขนาด 1024 บิท โดยข้อมูลชุดสุดท้ายของทั้ง สองชนิดจะมีการเติมบิทต่อท้ายเพิ่ม ซึ่งการเติม จะเริ่มจากบิทแรกโดย มีค่าเป็น 1 เพียงบิทเดียว เท่านั้นและตามด้วยบิทที่เป็น 0 ไปเรื่อยๆ จนกว่า จะมีขนาดเท่ากับ 448 บิท (SHA-256) หรือ 896 บิท (SHA-512) จากนั้นเพิ่มข้อมูลขนาด 64 บิท (SHA-256) หรือ 128 บิท (SHA-512) ต่อท้าย ในข้อมูลชุดสุดท้าย ซึ่งเป็นความยาวของข้อมูล เข้า (ถ้าข้อมูลเข้ามีขนาดมากกว่า 2^{64} บิทในกรณี SHA-256 หรือ 2^{128} บิท สำหรับ SHA-512 ก็จะใช้ ค่าเศษของผลหารขนาดข้อมูลด้วย 2^{64} หรือ 2^{128} ตามลำดับ) จนทำ ให้ข้อมูลชุดสุดท้ายมีขนาด 512 บิท (448+64) สำหรับ SHA-256 หรือ 1024 บิท (896+128) สำหรับ SHA-512 เหมือนกันชุด อื่นๆ

ขั้นตอนที่ 2

การคำนวณค่า hash เริ่มจากข้อมูลชุดแรกจนถึง ชุดสุดท้ายเป็นลำดับ โดยเมื่อประมวลผลข้อมูลแต่ ละชุด เช่นชุดที่ B^i จะต้องคำนวณหาค่า hash H^i ซึ่งได้มาจากการคำนวณฟังก์ชันพิเศษ σ_0, σ_1 ร่วมกับค่า W_t โดยสามารถแบ่งออกเป็น 4 ขั้นย่อยได้ดังนี้

1) กำหนดเงื่อนไขของ W_t (t คือลำดับของรอบที่ทำในขั้นที่ 3)

SHA-256:

$W_t = B_t^i,\ 0 \leq t \leq 15$

$W_t = \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16},$

$16 \leq t \leq 63$

โดย

$\sigma_1^{256} =$

$ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

$\sigma_0^{256} =$

$ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x)$

SHA-512:

$W_t = B_t^i, \quad \&0 \leq t \leq 15$

$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16},$

$16 \leq t \leq 79$

$W_t = B_t^i, \quad 0 \leq t \leq 15$

$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16},$

$16 \leq t \leq 79$

โดย

$\sigma_1^{512} = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x)$

$\sigma_0^{512} = ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x)$

หมายเหตุ:

$ROTR^y(x)$ คือการหมุนบิท (rotate) ของ x ไปทางขวาจำนวน y บิท และ $SHR^y(x)$ คือการเลื่อนบิท (shift) ของ x ไปทาง ขวาจำนวน y บิท

2) เตรียมข้อมูลและฟังก์ชันที่จำเป็น โดยใช้ค่า เริ่มต้นในการคำนวณจำนวน 8 ค่า (a,b,c,d,e, f,g,h) ร่วมกับค่า hash จากรอบก่อนหน้านี้ H_0^{i-1} จนถึง $H_7^{i-1}\ (a = H_0^{i-1}, \ldots, g = H_7^{i-1})$ ยกเว้น รอบแรกจะกำหนดค่าเริ่มต้นดังนี้คือ

SHA-256:

$H_0^0 = 6a09e667_{16}, H_1^0 = bb67ae85_{16},$

$H_2^0 = 3c6ef372_{16}, H_3^0 = a54ff53a_{16},$

$H_4^0 = 510e527f_{16}, H_5^0 = 9b05688c_{16},$

$H_6^0 = 9b05688c_{16}, H_7^0 = 5be0cd19_{16}$

SHA-512:

$H_0^0 = cbbb9d5dc1059ed8_{16},$

$H_1^0 = 629a292a367cd507_{16},$

$H_2^0 = 9159015a3070dd17_{16},$

$H_3^0 = 152fecd8f70e5939_{16},$

$H_4^0 = 67332667ffc00b31_{16},$

$H_5^0 = 8eb44a8768581511_{16},$

$H_6^0 = db0c2e0d64f98fa7_{16},$

$H_7^0 = 47b5481dbefa4fa4_{16}$

ค่าคงที่ K_t^{256} ขนาด 32 บิท จำนวน 64 ค่า กรณี SHA-256 (Figure 1)

ค่าคงที่ K_t^{512} ขนาด 64 บิท จำนวน 80 ค่า กรณี SHA-512 (Figure 2)

ฟังก์ชันตรรกะ

$Ch(x,y,z),\ Maj(x,y,z),\ \Sigma_0(x),\ \Sigma_1(x)$ ดังนี้

$Ch(x,y,z) = (x \wedge y) \oplus (\bar{x} \wedge z)$

$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

SHA-256:

$\Sigma_0(x) = ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$

$\Sigma_1(x) = ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$

SHA-512:

$\Sigma_0(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$

$\Sigma_1(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$

3) ทำการคำนวณแบบวนซ้ำ 64 รอบ (SHA-256 | $t \rightarrow [0 \ldots 63]$) หรือ 80 รอบ (SHA-512 | $t \rightarrow [0 \ldots 79]$)

$T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_t + W_t,$

$T_2 = \Sigma_0(a) + Maj(b,c,d),$

$h = g, g = f, f = e, e = d + T, d = c, c = b, b = a,$

$a = T_1 + T_2$

4) คำนวณค่า hash ข้อมูลชุดปัจจุบัน H_0^i จนถึง H_7^i โดยการบวกแบบ modulo-32 (SHA-256) หรือ modulo-64 (SHA-512) ดังสมการต่อไปนี้

$H_0^i = a + H_0^{i-1}, H_1^i = b + H_1^{i-1}, H_2^i = c + H_2^{i-1},$

$H_3^i = d + H_3^{i-1}, H_4^i = e + H_4^{i-1}, H_5^i = f + H_5^{i-1},$

$H_6^i = g + H_6^{i-1}, H_7^i = h + H_7^{i-1}$

ซึ่งเมื่อคำนวณค่า hash ของข้อมูลชุดสุดท้ายเสร็จก็จะได้ค่า H_0^n ถึง H_7^n จึงนำมาสร้างค่า hash ผลสำเร็จ $H = H_0^n \parallel H_1^n \parallel H_2^n \parallel H_3^n \parallel H_4^n \parallel H_5^n \parallel H_6^n \parallel H_7^n$

RIPEMD-160

เป็นอัลกอริทัมฟังก์ชันทิศทางเดียวที่ได้รับการ พัฒนาต่อจากอัลกอริทัม RIPEMD เนื่องด้วย RIPEMD นั้นเป็นที่ทราบกันดีว่าสามารถเกิด collision ได้โดยง่าย ส่วนหนึ่งมาจากลักษณะการ ทำงานที่ถูกออกแบบมาให้มีการทำงานเช่นเดียวกันกับอัลกอริทัมชนิด MD4 แต่ RIPEMD-160 ได้รับการปรับปรุงประสิทธิภาพให้สามารถแก้ปัญหาดังกล่าวได้ ในส่วนการทำงานจะรับข้อมูลเข้า ขนาดต่าง ๆ แล้วให้ผลลัพธ์เป็นข้อความที่ย่อยแล้ว ขนาดคงที่ 160 บิท มีขั้นตอนการทำงานมี 2 ส่วน คือ การเตรียมการและการคำนวณค่าhash เช่น เดียวกับอัลกอริทัมชนิดอื่น ๆ (จาก H. Dobbertin et al.[6] และ A. Bosselaers[5])

ขั้นตอนที่ 1

แบ่งข้อมูลเข้าออกเป็นชุดขนาด 512 บิทโดยข้อ มูลชุดสุดท้ายจะมีการเติมบิทต่อท้ายเพิ่ม ซึ่งการ เติมจะให้บิทแรกเป็น 1 เพียงบิทเดียวเท่านั้นและ ตามด้วยบิทที่เป็น 0 ไปเรื่อยๆ จนกว่าจะมีขนาด เท่ากับ 448 บิท จากนั้นเพิ่มข้อมูลขนาด 64 บิทต่อท้าย ในข้อมูลชุดสุดท้ายซึ่งเป็นความยาวของข้อมูลเข้า (ถ้าข้อมูลเข้ามีขนาดมากกว่า 2^{64} บิท ก็จะใช้ค่าเศษของผลหารขนาดข้อมูลด้วย 2^{64}) จนทำให้ข้อมูลชุดสดท้ายมีขนาด 512 (448+64) บิท เหมือนกับชุดอื่นๆ

ขั้นตอนที่ 2

การคำนวณค่า hash เริ่มจากข้อมูลชุดแรกจนถึงชุดสุดท้ายเป็นลำดับ โดยเมื่อประมวลผลข้อมูล แต่ละชุด เช่น ชุดที่ B^i จะต้องคำนวณหาค่า hash คือ H^i ซึ่งได้มาจากการคำนวณ ฟังก์ชัน $f_t(b,c,d)$, $f'_t(b,c,d), R_t$, R'_t ร่วมกับค่า W_t และ W'_t โดยสามารถแบ่งออกเป็น 4 ขั้น ย่อยได้ดังนี้

1) กำหนดเงื่อนไขของ W_t และ W'_t (t คือลำดับ ของรอบที่ทำในขั้นที่ 3)

$W_t = B^i_t$ เมื่อ $0 \leq t \leq 15$

$W_t = B^i_{u[t]}$ เมื่อ

$$u[16..79] = \begin{cases} [7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8], 16 \leq t \leq 31 \\ [3,10,14,4,9,15,8,1,2,7,0,6,13,11,5,12], 32 \leq t \leq 47 \\ [1,9,11,10,0,8,12,4,13,3,7,15,14,5,6,2], 48 \leq t \leq 63 \\ [4,0,5,9,7,12,2,10,14,1,3,8,11,6,15,13], 64 \leq t \leq 79 \end{cases}$$

$W'_t = B^i_{u[t]}$ เมื่อ

$$u[0..79] = \begin{cases} [5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12], 0 \leq t \leq 15 \\ [5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12], 16 \leq t \leq 31 \\ [15,5,1,3,7,14,6,9,11,8,12,2,10,0,4,13], 32 \leq t \leq 47 \\ [8,6,4,1,3,11,15,0,5,12,2,13,9,7,10,14], 48 \leq t \leq 63 \\ [12,15,10,4,1,5,8,7,6,2,13,14,0,3,9,11], 64 \leq t \leq 79 \end{cases}$$

2) เตรียมข้อมูลและฟังก์ชันที่จำเป็น โดยกำหนดค่าเริ่มต้นในการคำนวณจำนวน 5 ค่า มี 2 ชุดคือ (a,b,c,d,e) และ (a’,b’,c’,d’,e’) โดยเรียงลำดับ ใช้ค่า hash จากรอบก่อนหน้านี้ H_0^{i-1} จนถึง H_4^{i-1} ทั้งสองชุด $(a = H_0^{i-1}, \ldots, e = H_4^{i-1}, a' = H_0^{i-1}, \ldots, e' = H_4^{i-1})$ ยกเว้นรอบแรกจะกำหนดค่าเริ่มต้นดังนี้คือ

$H_0^0 = 67452301_{16}$, $H_1^0 = EFCDAB89_{16}$,

$H_2^0 = 98BADCFE_{16}$, $H_3^0 = 10325476_{16}$,

$H_4^0 = C3D2E1F0_{16}$

ค่าคงที่ K_t และ K'_t ขนาด 32 บิท

$K_t = 0_{16}$ เมื่อ $0 \leq t \leq 15$, $K_t = 5A82799_{16}$ เมื่อ $16 \leq t \leq 31$, $K_t = 6ED9EBA1_{16}$ เมื่อ $32 \leq t \leq 47$, $K_t = 8F1BBCDC_{16}$ เมื่อ $48 \leq t \leq 63$ $K_t = A953FD4E_{16}$ เมื่อ $64 \leq t \leq 79$, $K'_t = 50A28BE6_{16}$ เมื่อ $0 \leq t \leq 15$ $K'_t = 5C4DD124_{16}$ เมื่อ $16 \leq t \leq 31$, $K'_t = 6D703EF3_{16}$ เมื่อ $32 \leq t \leq 47$, $K'_t = 7A6D76E9_{16}$ เมื่อ $48 \leq t \leq 63$, $K'_t = 0_{16}$ เมื่อ $64 \leq t \leq 79$

ฟังก์ชันหมุนบิท R_t, R'_t

$R_t = ROTL^{u[t]}(x)$ เมื่อ t และ u มีค่าดังนี้

$$u[0..79] = \begin{cases} [11,14,15,12,5,8,7,9,11,13,14,15,6,7,9,8], 0 \leq t \leq 15 \\ [7,6,8,13,11,9,7,15,7,12,15,9,11,7,13,12], 16 \leq t \leq 31 \\ [11,13,6,7,14,9,13,15,14,8,13,6,5,12,7,5], 32 \leq t \leq 47 \\ [11,12,14,15,14,15,9,8,9,14,5,6,8,6,5,12], 48 \leq t \leq 63 \\ [9,15,5,11,6,8,13,12,5,12,13,14,11,8,5,6], 64 \leq t \leq 79 \end{cases}$$

$R'_t = ROTL^{u[t]}(x)$ เมื่อ t และ u มีค่าดังนี้

$$u[0..79] = \begin{cases} [8,9,9,11,13,15,15,5,7,7,8,11,14,14,12,6], 0 \leq t \leq 15 \\ [9,13,15,7,12,8,9,11,7,7,12,7,6,15,13,11], 16 \leq t \leq 31 \\ [9,7,15,11,8,6,6,14,12,13,5,14,13,13,7,5], 32 \leq t \leq 47 \\ [15,5,8,11,14,14,6,14,6,9,12,9,12,5,15,8], 48 \leq t \leq 63 \\ [8,5,12,9,12,5,14,6,8,13,6,5,15,13,11,11], 64 \leq t \leq 79 \end{cases}$$

ฟังก์ชันตรรกะ $f_t(b,c,d)$, $f'_t(b,c,d)$ ดังนี้

$$f_t(b,c,d) = \begin{cases} (b \oplus c \oplus d), 0 \leq t \leq 15 \\ (b \wedge c) \vee (\bar{b} \wedge d), 16 \leq t \leq 31 \\ (b \vee \bar{c}) \oplus d, 32 \leq t \leq 47 \\ (b \wedge c) \vee (c \wedge \bar{d}), 48 \leq t \leq 63 \\ b \oplus (c \oplus \bar{d}), 64 \leq t \leq 79 \end{cases}$$

$$f'_t(b,c,d) = \begin{cases} b \oplus (c \oplus \bar{d}), 0 \leq t \leq 15 \\ (b \wedge c) \vee (c \wedge \bar{d}), 16 \leq t \leq 31 \\ (b \vee \bar{c}) \oplus d, 32 \leq t \leq 47 \\ (b \wedge c) \vee (\bar{b} \wedge d), 48 \leq t \leq 63 \\ (b \oplus c \oplus d), 64 \leq t \leq 79 \end{cases}$$

3) ทำการคำนวณแบบวนซ้ำ 80 รอบ $(t \rightarrow [0...79])$

$T = R_t(a + f_t(b,c,d) + K_t + W_t) + e,\ a = e,$
$e = d,\ d = ROTL^{10}(c),$ $c = b, b = T$

$T' = R'_t(a' + f'_t(b',c',d') + K'_t + W'_t) + e',$

$a' = e', e' = d', d' = ROTL^{10}(c'),$

$c' = b', b' = T'$

4) คำนวณค่า hash ข้อมูลชุดปัจจุบัน H_0^i จนถึง H_4^i โดยการบวกแบบ modulo-32 ดังสมการต่อไปนี้

$T = H_1^{i-1} + c + d', H_1^i = H_2^{i-1} + d + e',$

$H_2^i = H_3^{i-1} + e + a', H_3^i = H_4^{i-1} + a + b',$

$H_4^i = H_0^{i-1} + b + c', H_0^i = T$

ซึ่งเมื่อคำนวณค่า hash ของข้อมูลชุดสุดท้ายเสร็จ ก็จะได้ค่า H_0^n ถึง H_4^n จึง นำมาสร้างค่า hash ผลสำเร็จ $H = H_0^n \parallel H_1^n \parallel H_2^n \parallel H_3^n \parallel H_4^n$

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Figure 1 32-Bit Constants for SHA-256

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817
```

Figure 2 64-Bit Constants for SHA-512

งานวิจัยที่เกี่ยวข้อง

การศึกษาเชิงเปรียบเทียบคุณสมบัติการใช้งาน ของ Hash Algorithm ที่ปรากฏส่วนใหญ่ไม่ได้ เน้นหนักไปในเชิงวิชาการ แต่เป็นให้ความสำคัญ ในการนำไปใช้จริงนั้นคือนำไปพัฒนาโปรแกรม ประยุกต์มากกว่า โดยเริ่มจาก Wei Dai[3] ได้นำเสนอผลการทดสอบ Hash Algorithm ชนิดต่างๆ บนคอมพิวเตอร์ที่มีหน่วยประมวล ผลกลางหลายชนิดโดยใช้โปรแกรมประยุกต์ที่ พัฒนาด้วยชุดคำสั่ง (library) ของ Crypt++ ส่วน Lonewolfer[4] ได้นำเสนอผลการทดสอบ การใช้งานชุดคำสั่ง (library) หลายแบบที่เกี่ยวกับ การใช้งาน hash algorithm เช่น cityhash, farmhash, murmurhash3, spookyv2, cfarmhash และ C++11 std::hash ซึ่งผลที่ได้จะเป็นการเปรียบเทียบชุดคำสั่งเหล่านี้กับข้อมูล เข้าในรูปของ array ที่ขนาดต่างๆ และสุดท้าย D. Assencio[2] ได้แนะนำการใช้งาน OpenSSL Performance Analysis Tool เบื้องต้นในการทดสอบประสิทธิภาพการเข้ารหัส ประเภทต่างๆ รวมไปถึง Hash Algorithm ด้วย ซึ่งได้มีการนำเสนอผลการทดสอบบางส่วนที่เป็น การเปรียบเทียบเวลาที่ใช้ในการประมวลผลของ MD5 และ HMAC-MD5 เพื่อเป็นการสาธิตการ ใช้งานอีกด้วย

วิธีการวิจัย

เป้าหมายของเอกสารฉบับนี้คือนำเสนอผลการวิเคราะห์เชิงเปรียบเทียบประสิทธิภาพทางด้าน เวลาในการทำงาน (speed benchmark) ของ Hash Algorithm 4 ชนิด คือ SHA-1, SHA-256, SHA-512 และ RIDEMD-160 โดยมีลักษณะ การทดลองและองค์ประกอบดังนี้

เครื่องมือและอุปกรณ์

เครื่องคอมพิวเตอร์รุ่น Apple's Mac Book Pro หน่วยประมวลผลกลาง (CPU) ชนิด Intel Core i5 2.3 GHz หน่วยความจำหลัก (main memory) ขนาด 8 GB ระบบเก็บข้อมูลถาวร (Hard drive) ขนาด 1 TB ระบบปฏิบัติการ (Operation System) ชนิด Mac OS X 10.11.3 (El Capitan)

โปรแกรมฐานระบบ Script Engine ชนิด NodeJS (Google's V8 Engine) version 5.8.0

โปรแกรม OpenSSL รุ่น 0.9.8zh

โปรแกรมภาษา JavaScript สำหรับเรียกใช้ Open SSL

ขั้นตอนการดำเนินการ

เริ่มจากทำการติดตั้งโปรแกรม NodeJS จากนั้นพัฒนาโปรแกรมประยุกต์เพื่อเรียกใช้ OpenSSL ด้านการเปรียบเทียบความเร็ว (speed test) ตามที่อธิบายไว้โดย Ristic[1] ในการทดสอบ การทำงานของ Hash Algorithm ทั้ง 4 ชนิดที่ได้เลือกไว้ โดยสั่งให้โปรแกรมทำงานโดยแต่ละ algorithm จะมีการทำงานซ้ำจำนวน 16 รอบและ ทำการเก็บบันทึกผลลัพธ์ที่ได้เพื่อนำไปวิเคราะห์ ซึ่งรูปที่ 1 แสดงขั้นตอนการทำงานโดยสังเขป

ผลการวิจัย

จากการทดสอบการทำงานของ OpenSSL Speed Test ซึ่งมีเข้ารหัสของ Hash Algorithm ด้วยข้อมูลทดสอบขนาด 16, 64, 256, 1024 และ 8192 ไบท์ ติดต่อกันเป็นเวลา 3 วินาที โดยนับจำนวนรอบของการทำงาน (loop count) ของแต่ละ algorithm (Table 1) ซึ่งผลลัพธ์ ที่ได้สามารถ จำแนกได้ตามขนาดข้อมูลดังนี้

ข้อมูลขนาด 16 Byte (128 bit) นั้น SHA-1, SHA-256 และ RIPEMD-160 จะต้องการทำ padding เพื่อให้ข้อมูลมีขนาด 512 bit และ 1024bit สำหรับ SHA-512 ตามขนาดของ block size ซึ่งทุก algorithm ก็จะย่อยข้อมูล (digest) เพียง 1 รอบจากผลจะเห็นได้ว่า SHA-1 ทำงานได้จำนวนครั้งมากที่สุด เพราะมีการทำงานที่ซับซ้อนน้อยที่สุดอีกทั้งมีผลลัพธ์ ขนาด 160bit ซึ่งน้อยที่สุดเมื่อเ ทียบกับชนิดอื่นๆ รองลงมาคือ RIP-EMD-160 ที่มีขนาดผลลัพธ์เท่ากับคือ 160 Bit ต่อมาก็คือ SHA-256 ซึ่งมีผลลัพธ์ขนาด 256 Bit และสุดท้ายคือ SHA-512 ที่มีผลลัพธ์ขนาด 512bit

ข้อมูลขนาด 64 Byte (512 bit) มีขนาดเท่ากับ block size ของ SHA-1, SHA-256 และ RIPEMD-160 หลังจากผ่านการ padding จะมีขนาดข้อมูล ก่อนประมวลผลอยู่ที่ 1024 bit ทำให้ในเวลาย่อยข้อมูลต้องทำ 2 รอบต่อ ครั้ง ส่วน SHA-512 ที่มี block size เท่ากับ 1024bit จึงมีการย่อยข้อมูล 1 รอบต่อ ครั้ง ดังนั้นจะเห็นว่าความเร็วในการย่อยข้อมูลของ SHA-512 สูงกว่าทั้ง SHA-256 และ RIPEMD-160 เนื่องจาก algorithm ทั้งสอง นี้มีการทำงานภายในที่มีลำดับ ขั้นตอนมากกว่า อย่างไรก็ตาม SHA-1 ก็ยัง มีความเร็วที่สุดเพราะมีความซับซ้อนน้อยกว่าอีก 3 ชนิดอยู่มากจึงยังมีผลการทำงาน เป็นอันดับแรก และตาม มาด้วย SHA-512, RIPEMD-160 และ SHA-256 ตามลำดับ

ข้อมูลขนาด 256 Byte (2048 bit) เมื่อนำ มาย่อยด้วย SHA-1, SHA-256 และ RIPEMD-160 โดยหลังจาก padding แล้วจะมีขนาดเท่ากับ 2560 bit ทำให้ในการ ย่อยจะต้องทำทั้งหมด 2560/512 = 5 รอบ แต่เมื่อใช้กับ SHA-512 จะถูก padding จนได้ขนาด 3072 bit ซึ่งต้องทำทั้งหมด 3072/1024 = 3 รอบ ซึ่งทำให้เร็วกว่า SHA-256 และ RIPEMD-160 ดังนั้นลำดับ ความเร็ว จึงเหมือนเดิมคือ SHA-1 เร็วที่สุด รองลงมาคือ SHA-512 และตามมาด้วย RIPEMD-160 และสุดท้าย คือ SHA-256

ข้อมูลขนาด 1024 Byte (8192 bit) และ 8192 Byte (65536 bit) ผลการเปรียบเทียบ ความเร็วในการทำงานยังคงเหมือนเดิมคือ SHA-1 มีความเร็วสูงสุดแล้วตามมาด้วย SHA-512 ซึ่งเหตุผลที่ทำให้ SHA-512 ทำงานได้เร็วกว่า SHA-256 และ RIPEMD-160 เนื่องจากเมื่อข้อมูลมีขนาดใหญ่เมื่อเทียบกับ Block Size มากๆส่งผลให้ การแบ่งเป็น block ในการย่อยแต่ละรอบ SHA-512 จะมีจำนวน block น้อยกว่านั้นคือครึ่งหนึ่ง ผลก็คือทำให้สามารถย่อยข้อมูล ได้เร็วกว่า SHA-256 และ RIPEMD-160 เพราะใช้ จำนวนรอบน้อยกว่าในการย่อยนั้นเอง

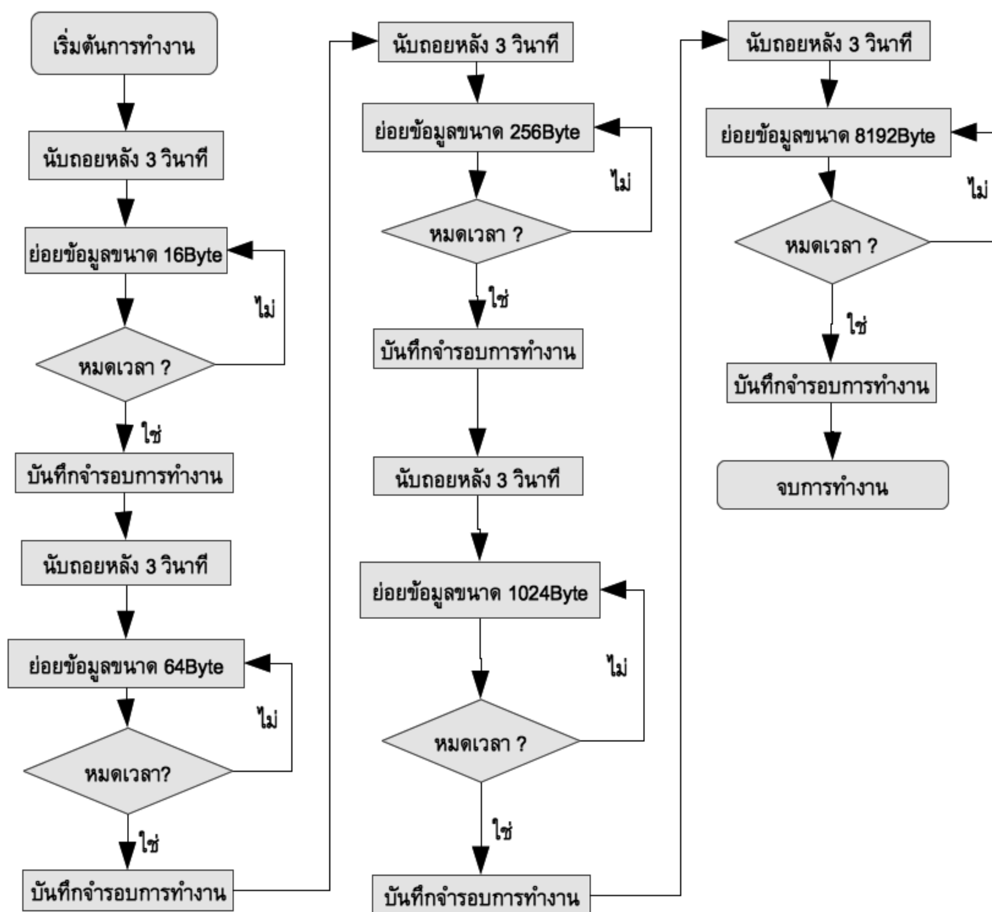

Figure 3 Flowchart of OpenSSL Caller Application

Table 1 Testing results showing number of loop count for each algorithm

Test Data Block size	Loop Count			
	SHA-1	SHA-256	SHA-512	RIPEMD-160
16 Bytes	5883385	3829946	2646815	3966983
64 Bytes	4409469	2190872	2496280	2365671
256 Bytes	2054471	921496	1140823	1072700
1024 Bytes	778829	270388	407891	360938
8192 Bytes	104233	37605	58370	48794

Table 2 Calculating throughputs from loop count and test data size for each algorithm

Hash Algorithms	Throughput (Kbytes / Second)				
	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
SHA-1	31425.24	94133.25	175459.61	266058.11	284704.82
SHA-256	20431.17	46792.18	78768.1	92321.12	102692.06
SHA-512	14119.74	53278.75	97459.55	139400.07	159433.5
RIPEMD-160	21176.56	50544.48	91733.56	123249.39	133414.74

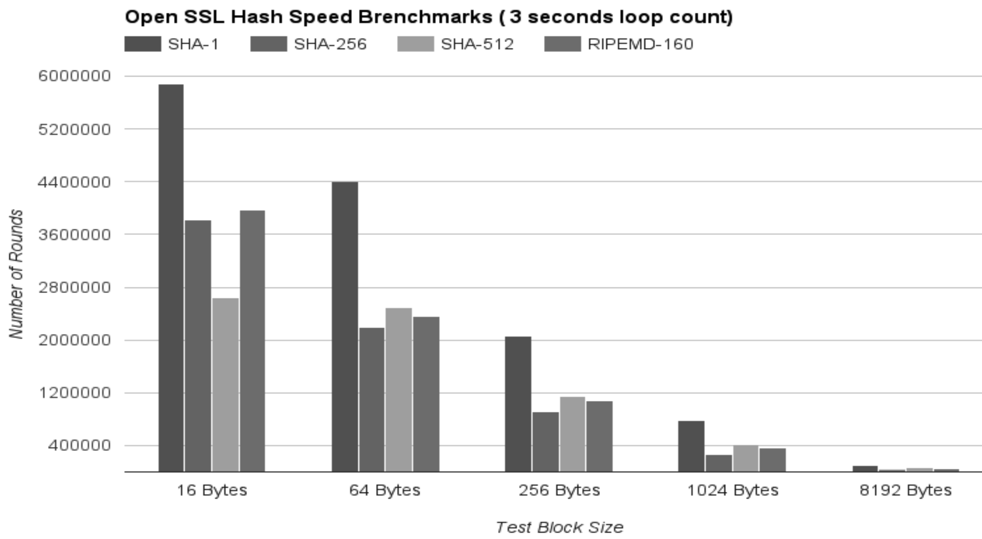

Figure 4 Bar chart showing testing result comparison from Table 1

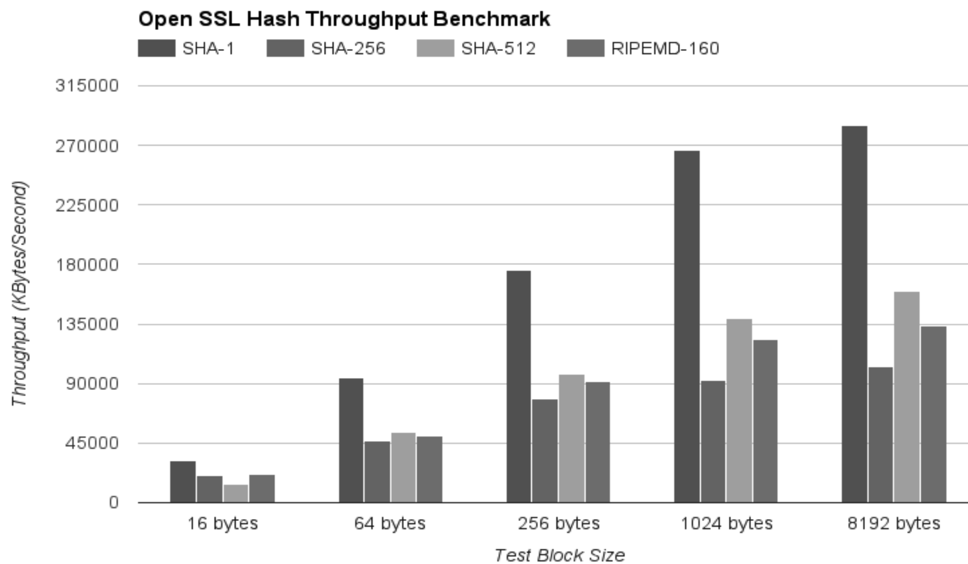

Figure 5 Bar chart showing throughput comparison for each algorithm

วิจารณ์และสรุปผล

จากผลการทดลองพบว่าปัจจัยที่ส่งผลโดยตรงต่อความเร็วในการทำงานคือขนาดของผลลัพธ์ (output size) ขนาดของข้อมูลที่ถูกย่อยแต่ละรอบ (block size) และจำนวนขั้นตอนย่อยภายใน (internal step count/ complexity) นอกจากนี้ประสิทธิภาพการประมวลผลในการย่อยข้อมูลต่อ เวลา (Throughput) ของแต่ละ Algorithm เพิ่มขึ้นตามขนาดของข้อมูลทดสอบ โดยผลการ เปรียบเทียบพบว่า SHA-1 เป็น algorithm ที่มีประสิทธิภาพสูงสุด รองลงมาคือ SHA-512 ตามมาด้วย RIPEMD-160 และต่ำที่สุดคือ SHA-256 อย่างไรก็ตามเป็นที่ทราบกันดีว่า SHA-1 นั้นพบว่ามีช่องโหว่ที่ไม่ปลอดภัยจากการ โจมตีดังนั้น Hash algorithm ที่เหมาะสมต่อ การใช้งานสำหรับ cloud application ในปัจจุบัน มากที่สุดคือ SHA-512 ซึ่งการโจมตียังเป็น ไปได้ยากมากด้วยสมรรถนะของคอมพิวเตอร์ในปัจจุบัน แต่ควรมีการประเมินเป็นระยะๆ ในอนาคตเพราะมีความเป็นไปได้สูงที่จะสามารถ ถูกโจมตีได้เมื่อประสิทธิภาพของคอมพิวเตอร์เพิ่มทวีขึ้นในราคาเท่าเดิมสุดท้ายนี้เนื่องจากผลการ ทดลองที่ได้เป็นเพียงผลจากการทำงานของโปร แกรม OpenSSL เพียงอย่างเดียวเท่านั้น ไม่ได้ทดสอบโปรแกรมประยุกต์หรือชุดคำสั่ง (Library) ชนิดอื่นๆประกอบซึ่งอาจมีความแตก ต่างในด้านผลการทดลองจึงควรที่จะมีการดำเนินการทดลองที่เหมือนกันกับชุดคำสั่ง หรือ โปรแกรมอื่นๆ เพื่อให้ได้ข้อสรุปที่ชัดเจน และ แม่นยำยิ่งขึ้น

เอกสารอ้างอิง

1. Ristic, “Performance”, OpenSSL CookBook, Feisty Duck Limited, May 2013.
2. D. Assencio, “Performance of cryptographic algorithms in OpenSSL”, Diego Assencio Blog, http://diego.assencio.com/?index=0e18f5485ff4de58d94e494e3649c6eb, April 9, 2014.
3. Wei Dai, “Crypto++ 5.6.0 Benchmarks”, Crypto++ Library 5.6.3 Website, https://www.cryptopp.com/benchmarks.html, March 31, 2009.
4. Lonewolfer, “Benchmarking Hash Functions”, Lonewolfer Blog, https://lonewolfer.wordpress.com/2015/01/05/benchmarking-hash-functions, January 5, 2015
5. A. Bosselaers, “The RIPEMD160 Page”,http://homes.esat.kuleuven.be/~bosselae/ripemd160.html
6. H. Dobbertin, A. Bosselaers, B. Preneel, “RIPEMD-160, a strengthened version of RIPEMD”, Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71-82.
7. S. Wanzhong, G. Hongpeng, He Huilei, and D. Zibin, “Design and Optimized Implementation of the Sha-2 (256, 384, 512) Hash Algorithms”, 7th International Conference on ASIC (2007), pp. 858-61.
8. SHA-2 Standard, National Institute of Standards and Technology (NIST), “Secure Hash Standard”, FIPS PUB 180-4, http://dx.doi.org/10.6028/NIST.FIPS.180-4