การวิเคราะห์ความปลอดภัยของฟังก์ชั่นแฮช
Effectiveness Analysis and Hash Function

วนิดา แซ่ตั้ง* และศิรปัฐช์ บุญครอง
คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

Wanida Saetang* & Sirapat Boonkrong
Faculty of Information Technology,
King Mongkut's University of Technology North Bangkok

บทคัดย่อ

ฟังก์ชั่นแฮชเป็นกลไกที่ช่วยในการตรวจสอบความถูกต้องสมบูรณ์ของข้อมูล ดังนั้น เพื่อให้การเข้าถึงระบบสารสนเทศเป็นไปอย่างปลอดภัยและมั่นใจได้ว่าไม่มีการปลอมแปลงข้อมูลเกิดขึ้น การยืนยันตนเพื่อเข้าสู่ระบบ (Authentication) จึงได้นำฟังก์ชั่นแฮชมาใช้ในการจัดเก็บและตรวจสอบความถูกต้องของรหัสผ่าน หากฟังก์ชั่นแฮชที่นำมาใช้มีความแข็งแรงมาก กล่าวคือ ข้อมูลต่างกันเพียงเล็กน้อย แต่ค่าผลลัพธ์ของฟังก์ชันแฮชที่ได้แตกต่างกันมาก นั่นหมายถึงจะทำให้ระบบมีความปลอดภัยมากเช่นกัน งานวิจัยเรื่องนี้จึงได้นำเสนอการวิเคราะห์ความปลอดภัยของฟังก์ชั่นแฮช โดยการพิจารณาผลต่างบิต (Avalanche Effect) ของค่าแฮชเมื่อมีการเปลี่ยนแปลงข้อมูลตั้งต้นเพียง 1 บิต ทำการทดลองกับฟังก์ชั่นแฮชแบบ Modification Detection Codes (MDC) และ Message Authentication Codes (MAC) กับข้อมูลที่มีรูปแบบอักขระแตกต่างกัน 5 แบบ ได้แก่ ตัวอักษรพิมพ์ใหญ่ ตัวอักษรพิมพ์เล็ก ตัวเลข สัญลักษณ์ และแบบผสม จากการทดลองโดยใช้ OpenSSL เป็นเครื่องมือ สามารถสรุปผลได้ว่า ฟังก์ชั่นแฮชแบบ MDC อัลกอริทึม SHA-1 มีผลต่างบิตมากที่สุด ในอักขระแบบสัญลักษณ์ และฟังก์ชั่นแบบ MAC อัลกอริทึม SHA-256 มีผลต่างบิตมากที่สุดในอักขระแบบผสม

คำสำคัญ: วิเคราะห์ความปลอดภัย ฟังก์ชั่นแฮช แฮชรหัสผ่าน ผลต่างบิต

* ผู้ประสานงานหลัก (Corresponding Author)
e-mail: wanidasaetang@gmail.com

Abstract

A hash function is a mechanism that can be used to verify the integrity of the data. It is mainly used to provide safe access to information and ensure that no tampering occurs. Authentication has included the hash functions to store and verify the validity of the password. If the used hash function is very strong, it allows the system to be very safe even though there is a slight change in the input and the output possesses significant changes. The aim of this paper was to analyze the security of hash functions by considering the avalanche effect when a single bit was changed. The experiments were implemented with Modification Detection Codes (MDC) and Message Authentication Codes (MAC) categories with upper case letters, lower case letters, numbers, symbols and mixes. The tool used in the experiments was OpenSSL. From the results, it could be concluded that the categories of MDC and SHA-1 had the most avalanche effect for symbols and characters and the categories of MAC and SHA-256 had the most avalanche effect for mixed characters

Keywords: Security Analysis, Hash Function, Password Hashing, Avalanche Effect

บทนำ

การเก็บรหัสผ่านในฐานข้อมูลหรือในไฟล์ข้อมูลเป็นวิธีการปกติในการใช้งานรหัสผ่านของแอพพลิเคชั่นต่าง ๆ อยู่แล้ว ดังนั้น ฐานข้อมูลหรือไฟล์ดังกล่าวจึงเปรียบเสมือนกุญแจเข้าสู่ระบบทั้งหมดที่ผู้ดูแลระบบต้องคอยระมัดระวังไม่ให้ผู้ไม่ประสงค์ดีนำออกไปได้ หรือถ้านำออกไปได้ก็ต้องนำไปใช้ประโยชน์ไม่ได้ ผู้เชี่ยวชาญด้านความมั่นคงปลอดภัย (Wimokthanan, 2012) จึงแนะนำว่า ไม่ควรเก็บรหัสผ่านไว้แบบ Plain Text และหนึ่งในวิธีที่นิยมนำมาใช้ในการเข้ารหัสผ่าน คือ ฟังก์ชันแฮช Stallings (2014) กล่าวว่า ฟังก์ชันแฮชมักจะใช้สำหรับลายเซ็นดิจิทัล นอกจากนี้ยังถูกนำมาใช้สำหรับการตรวจสอบข้อความ และนำมาใช้ในการจัดเก็บรหัสผ่านในฐานข้อมูลด้วย เนื่องจากฟังก์ชั่นแฮชมีคุณสมบัติการกระจายที่ดี (Pongsuwan, 2016) กล่าวคือ ข้อความเดียวกันเมื่อผ่านฟังก์ชั่นแฮชแล้วจะต้องได้ผลลัพธ์เหมือนเดิมเสมอ และหากข้อความที่ต่างกันเพียงเล็กน้อยเมื่อผ่านฟังก์ชั่นแฮชแล้วควรจะต้องได้ผลลัพธ์ที่ต่างกันมาก และที่สำคัญคือ ไม่ควรมีข้อความใด ๆ ตั้งแต่ 2 ข้อความขึ้นไป ที่ผ่านฟังก์ชั่นแฮชแล้วจะได้ผลลัพธ์ที่เหมือนกัน แต่อย่างไรก็ตามแม้โอกาสในการเดารหัสผ่านจากค่าแฮชจะทำได้ยากแต่ก็สามารถทำได้ เช่น Collision Attack, Brute Force Attack, Rainbow Table, Length Extension Attack เป็นต้น

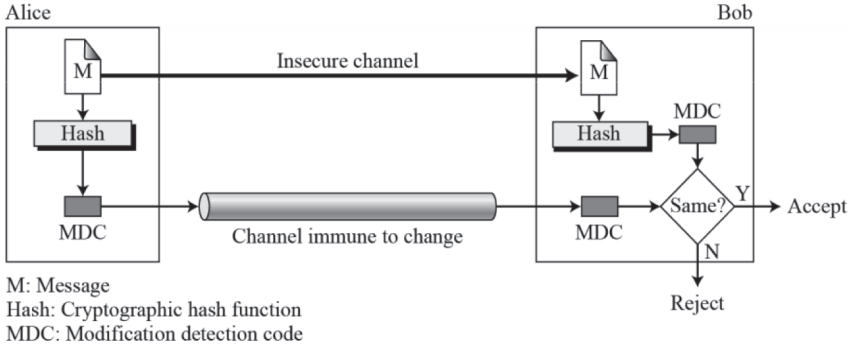
ในการกำหนดรหัสผ่านของระบบหรือเว็บไซต์ต่าง ๆ ไม่ว่าจะเป็นธุรกรรมทางการเงิน การรับส่งจดหมายอิเล็กทรอนิกส์ หรือการลงทะเบียนเข้าใช้งานระบบต่าง ๆ มักจะมีข้อแนะนำในการกำหนดรหัสผ่าน เช่น Hotmail (Microsoft, 2016) กำหนดไว้ว่า รหัสผ่านต้องประกอบด้วยอักขระอย่างน้อย 8 ตัว และต้องมีอักขระต่อไปนี้อย่างน้อย 2 รายการ ได้แก่ ตัวพิมพ์ใหญ่ ตัวพิมพ์เล็ก ตัวเลข และสัญลักษณ์ ในส่วนของ Facebook (Facebook, 2016) กำหนดไว้ว่า ป้อนอักขระอย่างน้อย 6 ตัวอักษร ให้มีผสมกันทั้งตัวเลข ตัวอักษร และเครื่องหมายวรรคตอน (เช่น ! และ &) ซึ่งการกำหนดเช่นนี้น่าจะเป็นการสร้างความแข็งแรงให้กับรหัสผ่านวิธีหนึ่ง

จากงานวิจัยของ Khayal et al. (2009) ได้ทำการวิเคราะห์ความปลอดภัยของฟังก์ชั่นแฮช ในด้านความเร็วของการเข้ารหัสผ่าน งานวิจัยของ Ratna et al. (2013) ได้วิเคราะห์และเปรียบเทียบ เวลาที่ใช้ในการโจมตีแบบ Brute Force ความเร็วในการประมวลผล และวัดการใช้งาน CPU และงานวิจัยของ Wang & Cao (2013) ทำการเปรียบเทียบความปลอดภัยของฟังก์ชั่นแฮชในด้านอัตราการชนกัน (Collision Rate) ความยาวค่าแฮช และความเร็วในการเข้ารหัส จากงานวิจัยข้างต้นจะเห็นได้ว่าในการวิเคราะห์ความปลอดภัยของฟังก์ชั่นแฮชนั้นสามารถทำได้หลากหลายวิธี และอีกวิธีหนึ่งที่สามารถวิเคราะห์ความปลอดภัยได้ คือ การเปรียบเทียบผลต่างบิตของค่าแฮช นั่นคือ Avalanche Effects ซึ่งวิธีการนี้จะแสดงค่าแฮชในระดับบิต เมื่อข้อมูลตั้งต้นหรือคีย์มีการเปลี่ยนแปลงเพียงเล็กน้อยหรือเพียง 1 บิต จะทำให้ค่าแฮชที่ได้เปลี่ยนแปลงไปมาก ดังนั้น ถ้าค่าแฮชที่ได้จากอัลกอริทึมการเข้ารหัสใด ๆ มีความแตกต่างกันมาก จะส่งผลให้การคาดเดารหัสผ่านทำได้ยาก ซึ่งหมายถึงอัลกอริทึมนั้นจะมีความแข็งแรงมากด้วย ดังนั้น งานวิจัยนี้จึงทำการทดสอบ Avalanche Effects ของฟังก์ชั่นแฮชแบบ MDC และ MAC โดยมีข้อมูลที่ใช้ในการทดลองแตกต่างกัน 5 แบบ ได้แก่ ตัวอักษรพิมพ์ใหญ่ ตัวอักษรพิมพ์เล็ก ตัวเลข สัญลักษณ์ และแบบผสม เปรียบเทียบผลต่างบิตจากการเปลี่ยนแปลงข้อมูลตั้งต้นเพียงบิตเดียว เพื่อหาว่าฟังก์ชันแฮชแบบใดมีประสิทธิผลสูงกว่ากันภายใต้การทดลองที่กำหนด สำหรับเป็นแนวทางการเลือกใช้ฟังก์ชั่นแฮชที่แข็งแรงในการจัดเก็บรหัสผ่าน เพื่อสร้างความปลอดภัยให้กับระบบและลดการชนกันของข้อมูล

1. ทฤษฎีที่เกี่ยวข้อง

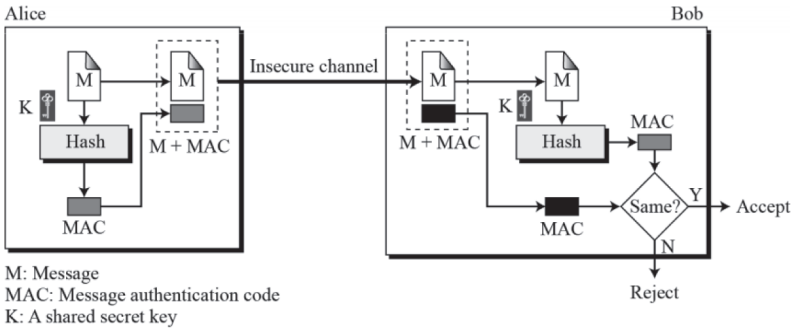
1.1 Cryptography Hash Function มีคุณสมบัติในการย่อยข้อมูล กลุ่มบิตนำเข้าที่มีขนาดความยาวแปรเปลี่ยนได้ (Variable Length) จะถูกลดขนาดไปเป็นกลุ่มบิตส่งออกที่มีขนาดความยาวคงที่ (Fixed-Length) เรียกว่า ค่าแฮช (Hash Value) ซึ่งหลักการทำงานของฟังก์ชั่นแฮชจะมีลักษณะเป็น One Way Functions หมายถึง จะไม่สามารถนำค่าแฮชเข้าฟังก์ชันแฮชทำกระบวนการย้อนกลับเพื่อให้ได้ข้อมูลเดิมกลับมาได้ การย่อยข้อมูลด้วยฟังก์ชั่นแฮชมี 2 ประเภท ได้แก่ Modification Detection Code (MDC) และ Message Authentication Code (MAC) (Swaminathan et al., 2006; Charasrisakul & Boonkrong, 2013; Kuacharoen, 2013; Stallings, 2014)

MDC เป็นการย่อยข้อมูลที่สามารถพิสูจน์ความสมบูรณ์ของข้อความได้ คือ ข้อความที่ยังไม่ได้รับการเปลี่ยนแปลง เช่น ผู้ส่งต้องการส่งข้อความไปยังผู้รับและมั่นใจได้ว่าข้อมูลจะไม่ถูกเปลี่ยนแปลงระหว่างทาง ผู้ส่งจะต้องสร้างข้อความและนำไปผ่านฟังก์ชั่นแฮช ซึ่งได้ข้อความที่ย่อยเรียบร้อยแล้ว (MDC) จากนั้นส่งทั้งข้อความต้นฉบับและ MDC ไปยังผู้รับ เมื่อถึงผู้รับ ผู้รับจะทำการแฮชข้อความต้นฉบับและเปรียบเทียบกับ MDC ถ้าเหมือนกันแสดงว่าข้อมูลไม่ถูกเปลี่ยนแปลง ดังภาพที่ 1 (Forouzan, 2011; Kuacharoen, 2013)



ภาพที่ 1 Modification Detection Code (MDC)
ที่มา: Forouzan (2011)

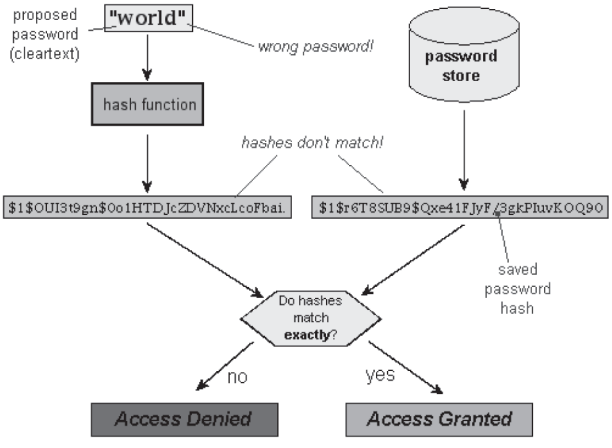
MAC คล้ายกับ MDC แต่มีข้อแตกต่าง คือ MAC สามารถตรวจสอบได้ว่าเจ้าของข้อความเป็นใคร เพราะมีคีย์เข้ามาเกี่ยวข้อง การส่งข้อความผู้ส่งจะทำการการย่อยข้อความ โดยนำคีย์และข้อความไปผ่านฟังก์ชั่นแฮชจะได้ MAC จากนั้นส่งข้อความไปพร้อมกับ MAC ดังภาพที่ 2 ทำให้ MAC มีความปลอดภัยสูงขึ้น ผู้โจมตีไม่สามารถเปลี่ยนแปลงข้อความหรือค่าแฮชได้ถ้าไม่ทราบว่าคีย์คืออะไร แต่ความปลอดภัยของ MAC นั้นจะขึ้นอยู่กับอัลกอริทึมที่ใช้ภายใน อัลกอริทึมที่เป็นที่นิยมในฟังก์ชั่นแฮช ได้แก่ MD5, SHA-1 และ SHA-2 Family (Forouzan, 2011; Kuacharoen, 2013)



ภาพที่ 2 Message Authentication Code (MAC)
ที่มา: Forouzan (2011)

การทำงานของอัลกอริทึม MD5 ข้อมูลจะถูกแบ่งออกเป็นบล็อก แต่ละบล็อกมีขนาด 512 บิต จากนั้นนำแต่ละบล็อกเข้าสู่กระบวนการประมวลผล 5 ขั้นตอน ค่าแฮชที่ได้มีขนาด 128 บิต สำหรับอัลกอริทึม SHA-1 ข้อมูลจะถูกแบ่งข้อมูลออกเป็นบล็อก แต่ละบล็อกมีขนาด 512 บิต การประมวลผลมี 5 ขั้นตอน เช่นเดียวกับอัลกอริทึม MD5 แต่มีรายละเอียดในการประมวลผลที่แตกต่างกัน และค่าแฮชที่ได้นั้นจะมีขนาดมากกว่า คือ 160 บิต (Ratna et al., 2013)

1.2 Passwords Hashing คือ การเก็บรหัสผ่านในรูปแบบของ Hash หรือ Cryptographic Hash เป็นการเก็บรหัสผ่านโดยใช้ฟังก์ชั่นแฮชและเก็บค่าแฮชลงในฐานข้อมูลแทนการเก็บรหัสผ่าน เมื่อมีผู้ใช้ทำการล็อกอินเข้าสู่ระบบ ระบบจะนำรหัสผ่านที่ผู้ใช้ป้อนเข้ามาไปผ่านแฮชแบบเดียวกันกับที่ใช้เก็บรหัสผ่านในระบบ ถ้าผลที่ได้ออกมาตรงกับค่าที่เก็บเอาไว้แสดงว่า รหัสผ่านที่ผู้ใช้งานป้อนเข้ามานั้นถูกต้อง ดังภาพที่ 3 (Friedl, 2004; Wimokthanan, 2012)



ภาพที่ 3 Passwords Hashing
ที่มา: Friedl (2004)

1.3 Avalanche Effect คุณสมบัติที่พึงประสงค์ของอัลกอริทึมการเข้ารหัสใด ๆ ที่มีการเปลี่ยนแปลงเล็กน้อยทั้งใน Plain Text หรือ Key ควรก่อให้เกิดการเปลี่ยนแปลงเป็นอย่างมากใน Cipher Text โดยเฉพาะอย่างยิ่งการเปลี่ยนแปลงในหนึ่งบิตของ Plain Text หรือหนึ่งบิตของ Key ควรส่งผลให้มีการเปลี่ยนแปลงในหลาย ๆ บิตของ Cipher Text ดังภาพที่ 4 หากมีการเปลี่ยนแปลงน้อยมากอาจทำให้การโจมตีสามารถทำได้ง่ายขึ้นเพราะ PlainText หรือ Key มีขนาดลดลง (Friedl, 2004; Stallings, 2014)

```
First 32 bits
0111 0101 1100 1101 1011 1111 1110 1011 75cdbfeb file1
0110 1111 1011 1110 0011 0111 1111 0001 6fbe37f1 file2
...X X.X. .XXX ..XX X... X... ...X X.X. 13 bits different

Second 32 bits
0111 0000 1010 0000 0110 1101 0100 0010 70a06d42 file1
1110 1110 1010 0000 1111 1000 0000 0010 eea0f802 file2
X..X XXX. .... .... X..X .X.X .X.. .... 10 bits different

Third 32 bits
0010 0001 0000 1001 0011 1000 1101 1010 210938da file1
1011 1101 0111 1001 0010 1110 1010 1000 bd792ea8 file2
X..X XX.. .XXX .... ...X .XX. .XXX ..X. 14 bits different

Fourth 32 bits
1000 1000 1100 0100 0010 1001 1001 0001 88c42991 file1
1000 0101 1100 1101 0000 0011 1110 0010 85cd03e2 file2
.... XX.X .... X..X ..X. X.X. .XXX ..XX 13 bits different
```

ภาพที่ 4 Avalanche Effect
ที่มา: Friedl (2004)

2. งานวิจัยที่เกี่ยวข้อง

2.1 Wangkeeree & Boonkrong (2015) ได้ศึกษา A Performance Comparison of MAC Hash Function using CBC-MAC and HMAC Algorithms ในด้านของการวิเคราะห์ Avalanche Effects ที่มีการเปลี่ยนแปลงคีย์ไป 1 บิต จำนวน 30 แบบ แต่ใช้ข้อมูลขนาดเดียว ผลการทดลองพบว่าค่าเฉลี่ยของ Avalanche Effects ของ HMAC-MD5 และ HMAC-SHA1 เท่ากับร้อยละ 50.00 และร้อยละ 48.98 ตามลำดับ ซึ่งตรงกับหลักการของ Collision Resistant ที่มีค่าเฉลี่ยค่อนข้างสูงที่มีจำนวนบิตต่างกัน

2.2 Pandey et al. (2014) ได้ศึกษา Efficient SHA-176 Secured and Integrated Algorithm ในด้านของการทดสอบ Avalanche Effect ซึ่งประกอบด้วย 4 อัลกอริทึม ได้แก่ ESHA (Efficient SHA), SHA-1, SHA-192(1) และ SHA-192(2) ผลการทดลองพบว่า SHA-1 ให้ผลต่างบิตอยู่ที่ 52.5% ซึ่งแตกต่างอย่างชัดเจนเมื่อเปรียบเทียบกับอัลกอริทึมอื่น ๆ เพราะฉะนั้น SHA-1 จะมีความปลอดภัยกว่าอัลกอริทึมอื่น ๆ ที่จะใช้สำหรับตรวจสอบความสมบูรณ์ของข้อมูล

2.3 Phukseng et al. (2014) ได้ศึกษา Performance Evaluation of MD5 and SHA-1 MDC Hash Functions ประสิทธิภาพของอัลกอริทึม MD5 และ SHA-1 ในด้านการให้ค่าแฮชที่แตกต่างกันเมื่อมีการเปลี่ยนแปลงข้อมูลนำเข้านั้น ทั้งสองอัลกอริทึมให้ผลลัพธ์ที่ใกล้เคียงกัน คือ มีสัดส่วนความแตกต่างกันของผลลัพธ์ที่ได้อยู่ที่ประมาณ 50% เช่นเดียวกัน

จากงานวิจัยข้างต้น พบว่า มีการทดลอง Avalanche Effects ของฟังก์ชันแฮชใน 2 แบบ คือ 1) การทดลองเปลี่ยนแปลงข้อมูลตั้งต้น โดยใช้ข้อมูลขนาดเดียวตลอดการทดลอง 2) การทดลองการเปลี่ยนแปลงคีย์ โดยใช้ข้อมูลขนาดเดียวตลอดการทดลอง จะเห็นว่าการทดลองทั้ง 2 แบบยังไม่มีความแตกต่างในด้านข้อมูลที่ใช้ในการทดลอง ดังนั้น งานวิจัยเรื่องนี้จึงได้นำเสนอวิธีการทดลองแบบใหม่ คือ การทดลองเปลี่ยนแปลงข้อมูลตั้งต้นเช่นเดียวกับการทดลองในแบบที่ 1 แต่มีความหลากหลายในด้านข้อมูลที่ใช้ในการทดลอง โดยใช้ข้อมูลที่มีอักขระแตกต่างกัน 5 แบบ ได้แก่ ตัวอักษรพิมพ์ใหญ่ ตัวอักษรพิมพ์เล็ก ตัวเลข สัญลักษณ์ และแบบผสม

วิธีการวิจัย

งานวิจัยทำการวิเคราะห์ Avalanche Effect ของฟังก์ชั่นแฮช โดยการทดลองแฮชข้อมูลและเปรียบเทียบผลต่างบิต เมื่อมีการเปลี่ยนแปลงบิตตั้งต้นอย่างน้อย 1 บิต ฟังก์ชั่นที่ให้ผลต่างบิตสูงเป็นฟังก์ชั่นที่มีความแข็งแรงสูง ในการทดลองมีรายละเอียดของการดำเนินการดังนี้

1. ฟังก์ชั่นแฮชที่ใช้ในการทดลอง
   1.1 แบบ MDC ได้แก่ MD5, SHA-1, SHA-256 และ SHA-512
   1.2 แบบ MAC ได้แก่ MAC-MD5, MAC-SHA-1, MAC-SHA-256 และ MAC-SHA-512
2. ข้อมูลที่ใช้ทดลอง
   2.1 ตัวอักษรพิมพ์ใหญ่ (Uppercase Letters) จำนวน 30 ไฟล์
   2.2 ตัวอักษรพิมพ์เล็ก (Lowercase Letters) จำนวน 30 ไฟล์
   2.3 ตัวเลข (Numbers) จำนวน 30 ไฟล์
   2.4 สัญลักษณ์ (Symbols) จำนวน 30 ไฟล์
   2.5 แบบผสม (Mixes) จำนวน 30 ไฟล์
3. เครื่องมือที่ใช้ในการทดลอง
   3.1 ซอฟต์แวร์ : OpenSSL เวอร์ชั่น 1.0.1 ภายใต้ระบบปฏิบัติการ Ubuntu เวอร์ชั่น 12.04
   3.2 ฮาร์ดแวร์ : คอมพิวเตอร์พกพา Intel(R) Core i7- 5500U, CPU@2.4GHz, RAM 8 GB, Hard Disc 500 GB
4. ขั้นตอนวิธีการทดลอง

4.1 กำหนดข้อมูลตั้งต้น สำหรับตัวอักษรพิมพ์ใหญ่ คือ “SECURITY” เข้าสู่แฮชฟังก์ชั่นโดยใช้รูปแบบคำสั่ง ดังนี้ “openssl dgst [hash algorithm] [-out filename] [filename]” (Heinlein, 2016) ดังนั้น คำสั่งที่ใช้คือ “openssl dgst md5 –out hashvalue.txt test.txt” ได้ค่าแฮชออกมาอยู่ในรูปเลขฐาน 16 จากนั้นทำการแปลงค่าแฮชที่ได้ให้อยู่ในรูปฐาน 2 และบันทึกผล

4.2 เปลี่ยนแปลงข้อมูล 1 บิต จากข้อมูลตั้งต้น คือ “SECURITY” เปลี่ยนเป็น “SECUSITY” จาก R เปลี่ยนเป็น S ซึ่ง R ระดับบิต คือ 01010010 และ S ระดับบิต คือ 01010011 มีการเปลี่ยนบิตสุดท้าย 1 บิต จาก 0 เป็น 1 นำเข้าสู่แฮชฟังก์ชั่นโดยใช้รูปแบบคำสั่งเดียวกับข้อ 4.1 ได้ค่าแฮชออกมาอยู่ในรูปเลขฐาน 16 จากนั้นทำการแปลงค่าแฮชที่ได้ให้อยู่ในรูปฐาน 2 และบันทึกผล (ตารางที่ 1)

4.3 เปรียบเทียบความแตกต่างระหว่างค่าแฮชตั้งต้น (ผลที่ได้จากข้อ 4.1) กับค่าแฮชที่เกิดจากการเปลี่ยนแปลงข้อมูล 1 บิต (ผลที่ได้จากข้อ 4.2)

ตารางที่ 1 ขั้นตอนวิธีการทดลอง

ลำดับ	กระบวนการ	ข้อมูลตั้งต้น	ข้อมูลถูกเปลี่ยนแปลง 1 บิต
1	ข้อมูล	SECURITY	SECUSITY
2	Ascii	R -> 0 1 0 1 0 0 1 0	S -> 0 1 0 1 0 0 1 1
3	คำสั่งที่ใช้	openssl dgst md5 –out hashvalue.txt test.txt	openssl dgst md5 –out hashvalue2.txt test2.txt
4	ค่าแฮช (Hex)	219ce424cc367fba 8a5a0210792d4dc0	ee30b02869e3f479 acb45810cabc0dad
5	ค่าแฮช (Binary)	0010000110011100 1110010000100100 1100110000110110 0111111110111010 1000101001011010 0000001000010000 0111100100101101 0100110111000000	1110111000110000 1011000000101000 0110100111100011 1111010001111001 1010110010110100 0101100000010000 1100101010111100 0000110110101101
6	ผลต่างบิต	61	

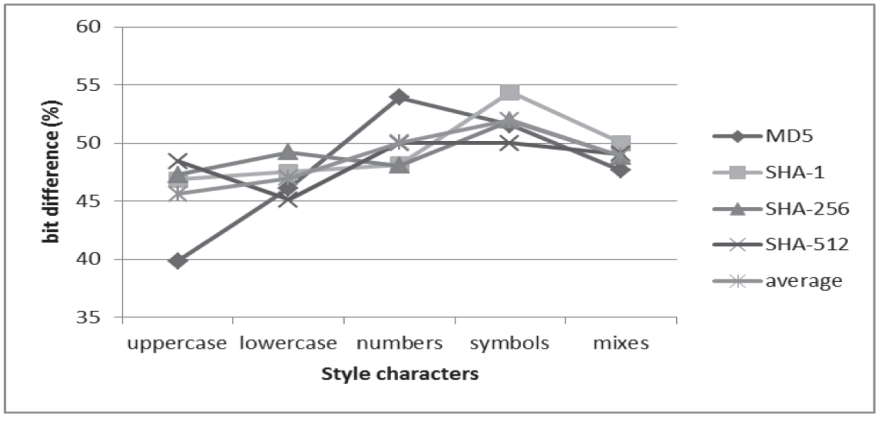
ผลการวิจัย

ผลจากการเปรียบเทียบ Avalanche Effects ของแต่ละฟังก์ชันกับชนิดข้อมูลแต่ละประเภท ให้ผลการทดลองแสดงใน ตารางที่ 2 แสดง Avalanche Effect ฟังก์ชั่นแฮชแบบ MDC กับรูปแบบอักขระที่แตกต่างกัน และตารางที่ 3 Avalanche Effect ฟังก์ชั่นแฮชแบบ MAC กับรูปแบบอักขระที่แตกต่างกัน

ตารางที่ 2 Avalanche Effect ฟังก์ชั่นแฮชแบบ MDC กับรูปแบบอักขระที่แตกต่างกัน

MDC	ผลต่างบิต (เปอร์เซ็นต์)					
	Uppercase	Lowercase	Number	Symbols	Mixes	Average
MD5	39.84	46.09	53.90	51.56	47.65	47.81
SHA-1	46.87	47.50	48.12	54.37	50.00	49.37
SHA-256	47.26	49.22	48.05	51.95	48.82	49.06
SHA-512	48.43	45.11	50.00	50.00	49.02	48.51
Average	45.60	46.98	50.02	51.97	48.87	

จากตารางที่ 2 เป็นการแสดงค่าเฉลี่ยคิดเป็นเปอร์เซ็นต์ของผลต่างบิต ฟังก์ชั่นแฮชแบบ MDC กับอักขระรูปแบบต่าง ๆ จะเห็นได้ว่า SHA-1 มีผลต่างบิตสูงสุด คือ 54.37% ในรูปแบบอักขระที่เป็นสัญลักษณ์ และ MD5 มีผลต่างบิตน้อยที่สุด อยู่ที่ 39.84% ในอักขระแบบตัวอักษรพิมพ์ใหญ่ ซึ่งแตกต่างกัน 36.47% ดังภาพที่ 5

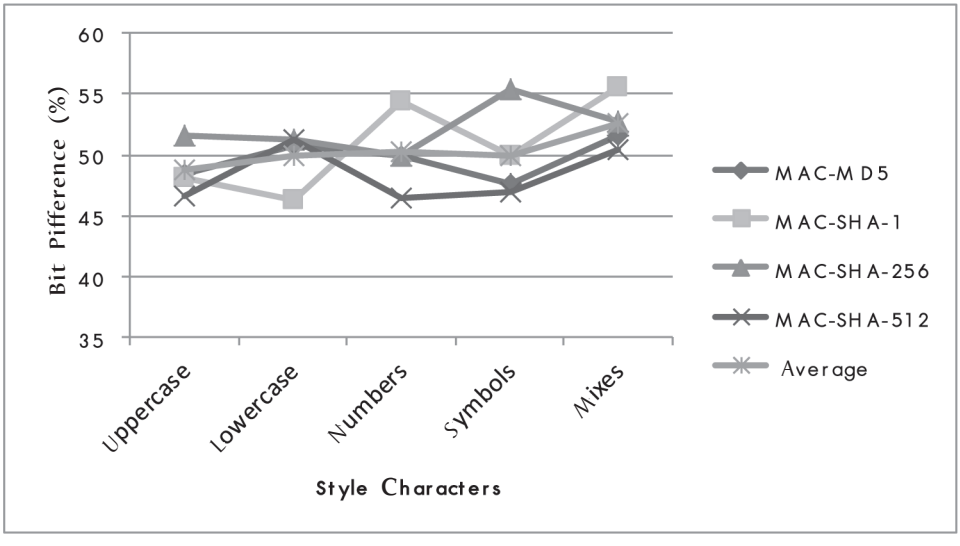


ภาพที่ 5 เปรียบเทียบ Avalanche Effect ฟังก์ชั่นแฮชแบบ MDC กับรูปแบบอักขระที่แตกต่างกัน

ตารางที่ 3 Avalanche Effect ฟังก์ชั่นแฮชแบบ MAC กับรูปแบบอักขระที่แตกต่างกัน

MAC	ผลต่างบิด (เปอร์เซ็นต์)					
	Uppercase	Lowercase	Number	Symbols	Mixes	Average
MD5	48.43	50.78	50.00	47.65	51.56	49.68
SHA-1	48.12	46.25	54.37	50.00	55.62	50.87
SHA-256	51.56	51.17	50.00	55.46	52.73	52.18
SHA-512	46.67	51.17	46.48	46.87	50.39	48.32
Average	48.69	49.84	50.21	49.99	52.57	

จากตารางที่ 3 เป็นการแสดงค่าเฉลี่ยคิดเป็นเปอร์เซ็นต์ของผลต่างบิด ฟังก์ชั่นแฮชแบบ MAC กับอักขระรูปแบบต่าง ๆ จะเห็นว่า MAC-SHA-1 มีผลต่างบิตสูงสุด คือ 55.62% ในรูปแบบอักขระแบบผสม และ MAC-SHA-1 เช่นเดียวกันที่มีผลต่างบิตน้อยที่สุด อยู่ที่ 46.25% ในรูปแบบอักขระตัวอักษรพิมพ์เล็ก ซึ่งแตกต่างกัน 20.25% ดังภาพที่ 6



ภาพที่ 6 เปรียบเทียบ Avalanche Effect ฟังก์ชั่นแฮชแบบ MAC กับรูปแบบอักขระที่แตกต่างกัน

สรุปผลการวิจัย

จากจุดมุ่งหมายของการวิจัยนี้คือการทดสอบ Avalanche Effects แฮชฟังก์ชั่นแบบ MDC ได้แก่ MD5, SHA-1, SHA-256, SHA-512 และ MAC ได้แก่ MAC-MD5, MAC-SHA-1, MAC-SHA-256, MAC-SHA-512 ข้อมูลที่ใช้ในการทดลองเป็นอักขระที่มีรูปแบบแตกต่างกัน ได้แก่ ตัวอักษรพิมพ์ใหญ่ ตัวอักษรพิมพ์เล็ก ตัวเลข สัญลักษณ์ และแบบผสม เปรียบเทียบผลต่างบิตจากการเปลี่ยนแปลงข้อมูลตั้งต้นเพียงบิตเดียว และจากการทดลองสามารถสรุปผลได้ว่า ฟังก์ชั่นแฮชประเภท MDC อัลกอริทึม SHA-1 มีผลต่างบิตเฉลี่ยมากที่สุด อยู่ที่ 49.37% ซึ่งสอดคล้องกับงานวิจัย Pandey et al. (2014) ที่พบว่า SHA-1 ให้ผลต่างบิตมากว่าอัลกอริทึม ESHA, SHA-1, SHA-192(1) และ SHA-192(2) อยู่ที่ 52.5% ส่วนอัลกอริทึม MD5 ให้ผลต่างบิตเฉลี่ยน้อยที่สุด อยู่ที่ 47.81% ซึ่งใกล้เคียงกับผลการทดลองของ Friedl (2004) ที่พบว่ามีการเปลี่ยนแปลงของค่าแฮชประมาณ 50 บิต หรือประมาณ 39% รูปแบบอักขระแบบสัญลักษณ์ มีผลต่างบิตเฉลี่ยมากที่สุด อยู่ที่ 51.97% และอักขระแบบตัวอักษรพิมพ์ใหญ่ มีผลต่างบิตเฉลี่ยน้อยที่สุด อยู่ที่ 45.60% สำหรับฟังก์ชั่นแฮชประเภท MAC ที่มีการใช้คีย์ในการแฮชข้อมูล MAC-SHA-256 ให้ผลต่างบิตเฉลี่ยมากที่สุด อยู่ที่ 52.18% ส่วน MAC-SHA-512 ให้ผลต่างบิตเฉลี่ยน้อยที่สุด อยู่ที่ 48.32% รูปแบบอักขระแบบผสม ให้ผลต่างบิตเฉลี่ยมากที่สุด อยู่ที่ 52.57% และ อักขระแบบตัวอักษรพิมพ์ใหญ่มีผลต่างบิตเฉลี่ยน้อยที่สุด อยู่ที่ 48.69%

จากผลการทดลองสามารถวิเคราะห์ได้ว่า สาเหตุที่ทำให้อัลกอริทึม MD5 มี Avalanche Effects น้อยกว่าอัลกอริทึมแบบอื่น ๆ เนื่องจากคุณสมบัติทางโครงสร้างของ MD5 ที่มีค่าแฮช เท่ากับ 128 บิต ซึ่งน้อยกว่า SHA-1, SHA-256 และ SHA-512 ทำให้ MD5 มีผลต่างบิตน้อยที่สุดในประเภท MDC สำหรับรูปแบบอักขระแบบผสมให้ผลต่างบิตสูง เนื่องจากมีความหลากหลายของอักขระแบบต่างๆ ประกอบกันทำให้ผลต่างบิตมีความแตกต่างกันมากกว่าอักขระแบบเดียว

จากการทดลองข้างต้น รูปแบบอักขระแบบผสมมีค่าเฉลี่ยผลต่างบิตมากที่สุด อยู่ที่ 52.57% ในประเภท MAC ซึ่งสามารถนำผลการทดลองนี้ไปประยุกต์ใช้กับกำหนดการตั้งรหัสผ่านได้ คือ ควรตั้งรหัสผ่านที่เป็นอักขระผสม เพราะการใช้ตัวเลข สัญลักษณ์ และตัวอักษรพิมพ์เล็กกับพิมพ์ใหญ่ผสมกันในรหัสผ่านจะทำให้บุคคลอื่นคาดเดารหัสผ่านได้ยากขึ้น ตัวอย่างเช่น รหัสผ่านยาว 8 อักขระที่ประกอบด้วยตัวเลข สัญลักษณ์ และตัวอักษรพิมพ์เล็กและพิมพ์ใหญ่ผสมกันนั้นคาดเดาได้ยากกว่า เพราะมีชุดค่าผสมที่เป็นไปได้มากกว่ารหัสผ่านยาว 8 อักขระที่ประกอบด้วยตัวอักษรพิมพ์เล็กอย่างเดียวถึง 30,000 ชุด (Google, 2016) และการจัดเก็บรหัสผ่านในฐานข้อมูล จากผลการทดลอง MAC-SHA-256 ให้ผลต่างบิตสูงสุด นั่นหมายถึงฟังก์ชั่น MAC-SHA-256 มีความแข็งแรงมากกว่าฟังก์ชั่นอื่นๆ สามารถใช้เป็นฟังก์ชั่นในการแฮชรหัสผ่านเก็บลงฐานข้อมูลได้เพื่อสร้างความแข็งแรงและปลอดภัยให้กับระบบได้วิธีหนึ่ง แต่อย่างไรก็ตามในการกำหนดและจัดเก็บรหัสผ่านในฐานข้อมูลเพื่อตรวจสอบความถูกต้อง นอกจาก

การวิเคราะห์ด้านความแข็งแรง ซึ่งสามารถพิจารณาได้จาก Avalanche Effect แล้ว ยังต้องมีการพิจารณาประสิทธิภาพด้านอื่น ๆ ประกอบด้วย เช่น ความเร็วในการประมวลผล ปริมาณการใช้ทรัพยากรในเครื่อง และการชนกันของข้อมูล เป็นต้น

References

Charasrisakul, K. & Boonkrong, S. (2013). Performance Comparison between Web Application and Windows Application. *The 9th National Conference on Computing and Information Technology* (pp. 437-442). Bangkok : King Mongkut's University of Technology North Bangkok.

Facebook. (2016). *Create an account.* Retrieved may 9, 2016, from facebook: https://en-gb.facebook.com

Forouzan, B. A. (2011). *Chapter 11 Message Integrity and Message Authentication.* Retrieved from https://cap430.files.wordpress.com/2011/03/ch11_12_131.pdf

Friedl, S. (2004). *An Illustrated Guide to Cryptographic Hashes.* Retrieved from http://www.unixwiz.net/techtips/iguide-crypto-hashes.html

Google. (2016). *Google helps.* Retrieved from https://support.google.com/accounts/answer/32040?hl=th

Heinlein, P. (2016). *OpenSSL Command-Line HOWTO.* Retrieved from https://www.madboa.com/geek/openssl/

Khayal, S. H., Khan, A., Bibi, N. & Ashraf, T. (2009). Analysis of password login phishing based protocols for security improvements. *IEEE Internaqtional Conference on Emerging Technologies,* (pp. 368-371). Islamabad, Pakistan.

Kuacharoen, P. (2013). *Design and Implementation of a Secure System: A Case of Online Lottery System.* Bangkok : National Institute of Development Administration.

Microsoft. (2016). *Create an account.* Retrieved from https://signup.live.com

Pandey, U. C., Gulati, A. & Richh, V. (2014). Efficient SHA-176 Secured and Integrated Algorithm. *International Journal of Computer Engineering and Applications,* 7, 115-122.

Phukseng, T., Thoopjeen, S. & Boonkrong, S. (2014). Performance Evaluation of MD5 and SHA-1 MDC Hash Functions. *The 8th Srinakharinwirot University Research Conference* (pp. 215-221). Bangkok : Srinakharinwirot University.

Pongsuwan, T. (2016). *The SHA-1 Secure Hash Function.* Retrieved from www.msit.mut.ac.th/member/filemanager/share_file/bon/.../Digital%20Signature.doc

Ratna, A. A. P., Purnamasari, P. D., Shaugi, A. & Salman, M. (2013). Analysis and comparison of MD5 and SHA-1 algorithm implementation in Simple-O authentication based security system. *IEEE International Conference on Quality in Research* (pp. 99-104). Yogyakarta, Indonesia.

Stallings, W. (2014). *Cryptography and Network Security Principles and Practices (6th Edition).* NJ: Pearson Prenticer Hall.

Swaminathan, A., Yinian, M. & Min, W. (2006). Robust and secure image hashing. *IEEE Transactions on Information Forensics and Security,* 1(2), 215-230. DOI : 10.1109/TIFS.2006.873601

Wang, Z. & Cao, L. (2013). Implementation and Comparison of Two Hash Algorithms. *IEEE International Conference on the Computational and Information Sciences* (pp. 721-725). Hubei, China.

Wangkeeree, N. & Boonkrong, S. (2015). A Comparison of MAC Hash Function using CBC-MAC and HMAC Algorithms. *The 6th Rajamangala University of Technology international Conference* (pp. 184-195). Bangkok : Rajamangala University of Technology Phra Nakhon.

Wimokthanan, P. (2012). *Password, Hash and Rainbow table.* Retrieved from https://www.thaicert.or.th/papers/technical/2012/pa2012te013.html

คณะผู้เขียน

นางสาววนิดา แซ่ตั้ง

ภาควิชาเทคโนโลยีสารสนเทศ คณะเทคโนโลยีสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
e-mail: wanidasaetang@gmail.com

รองศาสตราจารย์ ดร. ศิรปัฐช์ บุญครอง

ภาควิชาการสื่อสารข้อมูลและเครือข่าย คณะเทคโนโลยีสารสนเทศ
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ
e-mail: sirapat.b@it.kmutnb.ac.th