บทที่ 2
แนวความคิดที่ใช้และทฤษฎีที่เกี่ยวข้อง

จากความสำคัญของปัญหาในบทที่ 1 การเลือกใช้วิธีการหรืออัลกอริทึมที่เหมาะสมในการเข้ารหัสยูอาร์แอลมีผลกระทบต่อประสิทธิภาพในการทำงานของเว็บพร็อกซี่เป็นอย่างยิ่ง การเลือกใช้อัลกอริทึมที่ไม่เหมาะสมนอกจากจะทำให้เว็บพร็อกซี่ทำงานได้ไม่มีประสิทธิภาพแล้วยังทำให้ความน่าเชื่อถือลดลง การเข้ารหัสยูอาร์แอลนอกจากใช้ประโยชน์กับเว็บพร็อกซี่แล้ว ยังมีการใช้งานในแอปพลิเคชั่นอื่นๆที่ทำงานกับยูอาร์แอลด้วย

เว็บพร็อกซี่และแอปพลิเคชั่นที่เกี่ยวข้องกับการประมวลผลยูอาร์แอลที่เป็นที่นิยมในปัจจุบันนิยมใช้ MD5 ในการเข้ารหัสยูอาร์แอลเพื่อย่อขนาด อย่างไรก็ดียังไม่มีงานวิจัยที่ทดลองเข้ารหัสยูอาร์แอลและเสนอแนะการเลือกใช้อัลกอริทึมสำหรับการเข้ารหัสยูอาร์แอลแต่อย่างใด ผู้วิจัยจึงเกิดแนวความคิดในการทดสอบอัลกอริทึมต่างๆ ในการเข้ารหัสยูอาร์แอล

อัลกอริทึมที่เลือกมาทำการทดสอบนั้น มี 3 กลุ่มใหญ่ด้วยกัน คือ

1.อัลกอริทึมที่ใช้เข้ารหัสเพื่อตรวจสอบความผิดพลาดในการสื่อสารข้อมูล

คือ อัลกอริทึมประเภท Cyclic Redundancy Check หรือ CRC

2.การเข้ารหัสแบบฮัฟแมน (Huffman Coding)

เป็นการเข้ารหัสตัวอักษรแบบหนึ่ง ซึ่งมีลักษณะของการย่อข้อมูลด้วย

3.ฟังก์ชันแฮช (Hash Function)

เป็นการเข้ารหัสชนิดหนึ่ง ฟังก์ชันแฮช ในงานวิจัยนี้ยังแบ่งออกเป็น 2 กลุ่ม คือ

- ฟังก์ชันแฮชแบบง่าย
- ฟังก์ชันแฮช ประเภท เมสเสจไดเจส (message-digest)

ในการวิเคราะห์คุณสมบัติของแต่ละอัลกอริทึมนั้นจะวัดจากพารามิเตอร์ 3 ตัวด้วยกัน คือ

1. **ความเร็วในการเข้ารหัสยูอาร์แอล**
   อัลกอริทึมใดใช้เวลาในการเข้ารหัสน้อยกว่า ถือว่ามีประสิทธิภาพมากกว่า
2. **ขนาดของรหัสที่ได้**
   อัลกอริทึมใดได้ผลลัพธ์หรือรหัสที่สั้นกว่า ถือว่ามีประสิทธิภาพมากกว่า
3. **ปริมาณการชนกันของข้อมูล** (Collision)
   อัลกอริทึมใดได้ผลลัพธ์ที่มีการชนกันของข้อมูลน้อยกว่า ถือว่ามีประสิทธิภาพมากกว่า

อัลกอริทึมที่ใช้ในการวิจัย

2.1 การเข้ารหัสแบบ Cyclic Redundancy Check

Cyclic Redundancy Check[8] เป็นกรรมวิธีในการเข้ารหัสข้อมูลเพื่อตรวจหาความผิดพลาดในการสื่อสารข้อมูล โดยข้อมูลจะถูกมองเป็นบล็อกของข้อมูลและนำบล็อกข้อมูลมาสร้าง Frame Check Sequence หรือ FCS ในการส่งข้อมูลนั้นจะส่งบล็อกข้อมูล+FCS ออกไปยังช่องทางสื่อสาร การสร้าง FCS ใช้วิธีหารและใช้เศษของการหารเป็นผลลัพธ์แบบ modulo 2 ซึ่งเป็นการบวกแบบไบนารีโดยไม่มีการทด (binary addition with no carries) หรือเป็นเพียงการทำ exclusive OR นั่นเอง

การเลือกตัวหารเพื่อทำการหารโดยใช้เศษของการหารเป็นผลลัพธ์ (modulo) จะขึ้นกับลักษณะของความผิดพลาดที่คาดว่าจะเกิดขึ้นและต้องการตรวจพบ ตัวหารจะอยู่ในรูปของฟังก์ชันโพลิโนเมียล ขนาดของ FCS จะมีขนาดขึ้นอยู่กับขนาดของตัวหารที่เลือกใช้ และมักจะสั้นกว่าข้อมูลต้นฉบับที่นำมาหา FCS และเนื่องจาก FCS เป็นเศษจากการหารด้วยตัวหารที่เลือกขึ้นมา ดังนั้นข้อมูลต้นฉบับที่แตกต่างกันก็อาจให้ FCS ที่เหมือนกันได้ และจะเรียกเหตุการณ์นี้ว่า "เกิดการชนกัน" (Collision) ของผลลัพธ์ ตัวหารที่ใช้ในงานวิจัยมีดังนี้ คือ

$$\text{CRC-16} \quad = X^{16}+X^{15}+X^{2}+1$$

$$\text{CRC-CCITT} \quad = X^{16}+X^{12}+X^{5}+1$$

$$\text{CRC-32} \quad = X^{32}+X^{26}+X^{25}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+1$$

โปรแกรมซึ่งเขียนขึ้นสำหรับใช้ในการวิจัยนี้ จะใช้ตัวหาร CRC-16, CRC-CCITTไม่ใช้อัลกอริทึมซึ่งใช้การหารตรงๆ แต่จะเป็นการทำงานในระดับบิต (bit) ของข้อมูล และมีการใช้ตารางแทนค่าในการทำโอเปอเรชั่น XOR ซึ่งจะทำให้โปรแกรมที่ได้มีประสิทธิภาพมากกว่า

2.2 การเข้ารหัสแบบฮัฟแมน (Huffman Coding)

การเข้ารหัสแบบฮัฟแมน (Huffman Coding)[9] กำเนิดขึ้นในปี ค.ศ. 1952 โดยใช้แนวความคิดในการหารหัสให้สั้นที่สุด (มีจำนวนบิตน้อยที่สุด) โดยเริ่มจากการคำนวณความถี่ของการเกิดสัญลักษณ์หรือตัวอักขระแต่ละตัวจากข้อมูลต้นฉบับ จากนั้นจะนำผลที่ได้มาสร้างเป็นตารางรหัส โดยกำหนดให้ตัวอักขระที่พบมากที่สุดมีรหัสที่สั้นที่สุด ทำให้รหัสที่ได้มีขนาดเล็กกว่าข้อมูลจริง และสามารถถอดรหัสได้รวดเร็วอีกด้วย

วิธีการเข้ารหัสแบบฮัฟแมนนี้ใช้ทั้ง dynamic และ static statistic scheme ในรูปแบบของการทำงานโดยการใช้สถิติ กล่าวคือ จะใช้สถิติของการปรากฏของข้อมูล ซึ่งจะพิจารณาสถิติของการปรากฏของข้อมูลจากข้อมูลที่ต้องการเข้ารหัสทั้งหมด ลักษณะการพิจารณานี้เองที่เป็น

ตัวแบ่งระหว่าง static และ dynamic กล่าวคือ dynamic scheme จะเป็นการวิเคราะห์สถิติของข้อมูลที่จะทำการเข้ารหัสทั้งหมดก่อน จากนั้นจึงนำไปใช้ในการเข้ารหัส ซึ่งเมื่อมีการเปลี่ยนข้อมูลที่จะเข้ารหัสไป ก็จะต้องมีการวิเคราะห์ทางสถิติทุกครั้ง ในขณะที่แบบ static นั้นจะวิเคราะห์ข้อมูลซึ่งถูกใช้เป็นตัวแทนของข้อมูลที่จะทำการเข้ารหัสเพียงครั้งเดียว จากนั้นจะใช้ค่าทางสถิตินี้กับทุกๆข้อมูลที่จะทำการเข้ารหัสต่อๆไป จะเห็นว่าการใช้แบบ dynamic นั้นจะให้ผลลัพธ์ที่ดีที่สุด แต่จะทำงานได้ช้ากว่า เนื่องจากต้องมีการวิเคราะห์หารหัสใหม่ทุกครั้งที่ข้อมูลขาเข้าเปลี่ยน

ในการวิจัยจะใช้การวิเคราะห์แบบ Static Scheme ผลลัพธ์ที่ได้นอกจากจะได้รหัสที่มีความเฉพาะตัวจากข้อมูลต้นฉบับแล้ว รหัสที่ได้ยังมีขนาดเล็กกว่าข้อมูลต้นฉบับแต่มีขนาดไม่คงที่ขึ้นอยู่กับข้อมูลต้นฉบับ ซึ่งถือได้ว่าเป็นการบีบอัดข้อมูลในลักษณะหนึ่งด้วย จากลักษณะดังกล่าว ทางผู้วิจัยจึงได้นำอัลกอริทึมนี้มาทดลองใช้ในการเข้ารหัสยูอาร์แอลที่เป็นข้อมูลตัวอักษร ซึ่งอาจจะมีความเหมาะสมในการเข้ารหัสด้วยวิธีการนี้

2.3 **ฟังก์ชั่นแฮช** (Hash Function)

2.3.1 **ฟังก์ชั่นแฮช คืออะไร**

ฟังก์ชั่นแฮช **H** เป็นการเปลี่ยนรูปข้อมูลจากข้อมูลเข้า **m** และได้ผลลัพธ์เป็นสตริงที่มีขนาดคงที่ (fixed-size string) ซึ่งเรียกว่า ค่าแฮช (hash value) ***h (h=H(m))*** ฟังก์ชันแฮชมีการใช้งานในการคำนวณโดยทั่วไป แต่เมื่อใช้งานกับกระบวนการในการสร้างรหัสลับ (cryptography) ฟังก์ชันแฮชที่ถูกนำมาใช้เพื่อการนี้จะมีความต้องการพื้นฐานอีกบางประการดังต่อไปนี้

ความต้องการพื้นฐานสำหรับฟังก์ชันแฮชที่ใช้ในกระบวนการเข้ารหัส

- ข้อมูลขาเข้า (input) จะมีความยาวเท่าใดก็ได้
- ผลลัพธ์ที่ได้ต้องมีขนาดคงที่
- ฟังก์ชัน **H(x)** จะต้องง่ายในการคำนวณ ไม่ว่าจะเป็นค่า **x** ใดๆ
- ฟังก์ชัน **H(x)** เป็นฟังก์ชันทางเดียว (one-way)
- ฟังก์ชัน **H(x)** ไม่มีการชนกัน (collision) ของผลลัพธ์

ฟังก์ชันแฮช **H** จะเรียกว่าเป็นฟังก์ชันทางเดียวก็ต่อเมื่อ ยากต่อการหาอินเวอร์ต (invert) กล่าวคือ เมื่อให้ค่าแฮช ***h*** มาหนึ่งค่าแล้ว เป็นไปไม่ได้ที่จะหาข้อมูลขาเข้า ***x*** อื่นๆ ที่ **H(*x*)=*h***

ถ้าให้ข่าวสารหรือข้อมูลขาเข้า ***x*** และไม่สามารถหาหรือคำนวณข่าวสาร ***y*** ซึ่งแตกต่างจาก ***x*** และ **H(*x*)=H(*y*)** จะเรียก **H** ว่าเป็นฟังก์ชันแฮชที่ปลอดการชนกันอย่างอ่อน (weakly

collision-free hash function) และจะเรียก **H** ว่าเป็นฟังก์ชันแฮชที่ปลอดการชนกันอย่างแข็ง (strongly collision-free hash function) **H** จะต้องไม่สามารถคำนวณ 2 ข่าวสาร x และ y ใดๆ ที่ $\mathbf{H}(x)=\mathbf{H}(y)$

ค่าของฟังก์ชันแฮชจะเป็นตัวแทนอย่างย่อซึ่งใช้แทนข่าวสารหรือเอกสารของข้อมูลที่นำมาหาค่าแฮชนั้นๆ ซึ่งค่าแฮชนี้จะเรียกว่า เมสเสจไดเจส (message-digest) การประยุกต์ใช้เมสเสจไดเจสซึ่งเป็นที่รู้จักกันดีก็คือ ลายนิ้วมืออิเล็กทรอนิกส์ (digital fingerprint) ของเอกสารขนาดใหญ่ ตัวอย่างของฟังก์ชันแฮชที่เป็นที่รู้จักกันดี ได้แก่ MD2 MD5 และ SHA เป็นต้น

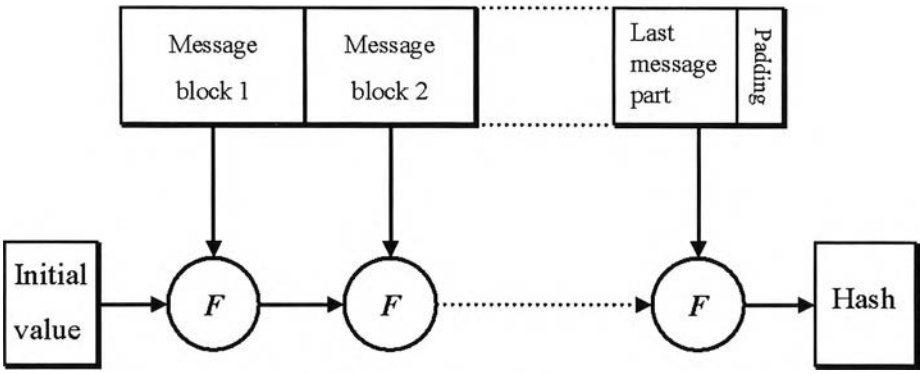
ฟังก์ชันแฮชมีบทบาทสำคัญในกระบวนการเข้ารหัสลับ นั่นคือ ใช้ในการตรวจสอบ *ความถูกต้องของข่าวสาร* (message integrity) และใช้ในการสร้าง *ลายเซ็นอิเล็กทรอนิกส์* (digital signature) เนื่องจากฟังก์ชันแฮชทำงานได้เร็วกว่าอัลกอริทึมในการเข้ารหัสหรืออัลกอริทึมในการสร้างลายเซ็นอิเล็กทรอนิกส์แบบอื่นๆ จึงถูกใช้งานในการคำนวณลายเซ็นอิเล็กทรอนิกส์ หรือใช้ในการตรวจสอบความถูกต้องของข้อมูลของเอกสาร โดยการประมวลผลกับค่าแฮชของเอกสารเหล่านั้นแทนที่จะทำกับเอกสารโดยตรง เพราะค่าแฮชจะมีขนาดเล็กกว่าเอกสารต้นฉบับมาก ยิ่งไปกว่านั้น เมสเสจไดเจสยังสามารถเปิดเผยได้โดยไม่ทราบว่าข้อมูลต้นฉบับเดิมที่นำมาสร้างนั้นเป็นอะไร ซึ่งเป็นสิ่งสำคัญในการทำ *การประทับเวลา* (timestamping) การใช้ฟังก์ชันแฮชทำให้สามารถตรวจสอบการประทับเวลาได้โดยไม่ต้องเปิดเผยถึงสิ่งที่อยู่ในเอกสาร

Damgård and Merkle [Dam90] [Mer90a] ได้นิยามฟังก์ชันแฮชว่าเป็น ฟังก์ชันในการบีบอัด (compression function) โดยฟังก์ชันในการบีบอัดนี้จะรับข้อมูลเข้าที่มีความยาวแน่นอน (fixed-length) และให้ผลลัพธ์ที่มีขนาดสั้นกว่าและคงที่ (fixed) การคำนวณฟังก์ชันแฮชอาจทำโดยทำการคำนวณซ้ำๆกันกับบล็อกของข้อมูลขาเข้าไปเรื่อยๆจนหมด ในการคำนวณนี้ข้อมูลขาเข้าจะถูกแบ่งออกเป็นบล็อกๆ เท่าๆกัน โดยขนาดของบล็อกขึ้นอยู่กับฟังก์ชันแฮช และข้อมูลขาเข้าจะต้องถูกเติม (pad) เพื่อให้มีขนาดเป็นจำนวนเท่าของขนาดบล็อก บล็อกข้อมูลจะถูกประมวลผลตามลำดับ และผลลัพธ์สุดท้ายจะเป็น ค่าแฮช ดังรูปที่ 2.10

นอกจากฟังก์ชันแฮชที่ใช้ในกระบวนการเข้ารหัสแล้ว การคำนวณแบบง่ายๆก็สามารถใช้เป็นฟังก์ชันแฮชได้เช่นเดียวกัน เช่น การนำข้อมูลขาเข้าทุกบล็อกมาบวกเข้าด้วยกันจากนั้นนำไปหารด้วย 256 และนำเฉพาะเศษที่ได้มาใช้เป็นค่าแฮชก็ได้

สำหรับการใช้ฟังก์ชั่นแฮชในการเข้ารหัสยูอาร์แอลนั้น การชนกันของข้อมูลเป็นเพียงปัจจัยหนึ่งที่ผู้วิจัยนำมาเป็นเครื่องชี้วัดประสิทธภาพของการเข้ารหัสยูอาร์แอลเท่านั้น ในบาง

กรณีการใช้ฟังก์ชันแฮชแบบที่ใช้การคำนวณง่ายๆ ก็อาจมีข้อดีและเหมาะสมในการใช้งาน ดังนั้นในงานวิจัยนี้ จึงได้เลือกฟังก์ชันแฮชที่มีการคำนวณง่ายๆ มาทดสอบเปรียบเทียบด้วย



รูปที่ 2.1 โครงสร้างการทำซ้ำของฟังก์ชั่นแฮช ซึ่งนิยามโดย Damgård และ Merkle

สำหรับในส่วนถัดจากนี้ไป จะเป็นการอธิบายการทำงานของฟังก์ชันแฮชที่ใช้ในงานวิจัยพอสังเขป โดยแบ่งฟังก์ชันแฮชออกเป็น 2 กลุ่ม คือ ฟังก์ชันแฮชในกลุ่มของ MD ซึ่งโดยทางทฤษฎีแล้วจะไม่เกิดการชนกันของข้อมูล กับฟังก์ชันแฮชที่ใช้วิธีการคำนวณง่ายๆ ซึ่งน่าจะมีความเร็วในการทำงานมากกว่าฟังก์ชันแฮชในกลุ่ม MD

ฟังก์ชั่นแฮชในกลุ่ม MD ที่ใช้ในงานวิจัย ประกอบด้วย MD2 MD4 MD5 และ SHA ซึ่งอาจแบ่งเป็น 2 กลุ่มย่อยได้อีก คือ กลุ่ม MD2, MD4, MD5 หนึ่งกลุ่ม และ SHA อีกหนึ่งกลุ่ม มีรายละเอียดการทำงานของแต่ละอัลกอริทึมดังต่อไปนี้

2.3.2 อัลกอริทึมในกลุ่ม MD (Message Digest Algorithms)

MD2 [Kal92], MD4 [Riv91b] [Riv92b] และ MD5 [Riv92c] เป็นอัลกอริทึมในกลุ่มการย่อข้อมูล (message-digest algorithm) ซึ่งพัฒนาขึ้นมาโดยศาสตราจารย์ Rivest แห่งมหาวิทยาลัย MIT เพื่อประยุกต์ใช้เป็น "ลายเซ็นอิเล็กทรอนิกส์" (digital signature) ข่าวสารขนาดใหญ่จะต้องถูกบีบอัดด้วยวิธีการที่ปลอดภัยก่อนจะกำกับด้วยรหัสส่วนตัว (private key) อัลกอริทึมทั้ง 3 จะรับข้อมูลเข้าที่มีขนาดใดๆขนาดหนึ่งและนำมาสร้างเป็นเมสเสจไดเจสที่มีขนาด 128 บิต โครงสร้างของทั้ง 3 อัลกอริทึม มีลักษณะที่คล้ายคลึงกัน MD2 มีการออกแบบแตกต่างออกไปจาก MD4 และ MD5 เนื่องจาก MD2 ถูกออกแบบขึ้นมาก่อนและให้ทำงานได้อย่างมีประสิทธิภาพกับเครื่องคอมพิวเตอร์ที่มีสถาปัตยกรรม 8 บิต ในขณะที่ MD4 ถูกออกแบบมาภายหลังเพื่อให้ทำงานได้ดีกับเครื่องคอมพิวเตอร์ที่มีสถาปัตยกรรม 32 บิต ส่วน MD5

ถูกออกแบบขึ้นมาล่าสุดเพื่อปรับปรุงให้มีความปลอดภัยมากขึ้นกว่า MD4 แต่ก็ทำงานได้ช้ากว่า MD4 ด้วย

อัลกอริทึมตัวสุดท้ายในกลุ่ม MD ก็คือ SHA ซึ่งย่อมาจาก Secure Hash Algorithm เป็นอัลกอริทึมที่กำหนดมาตรฐานในการแฮชที่ปลอดภัย (SHS, FIPS 180) พัฒนาโดย NIST [NIS93a] SHA-1 [NIS94c] ปรับปรุงจาก SHA ตีพิมพ์ในปี ค.ศ. 1994 โดยได้แก้ไขข้อบกพร่องของ SHA ที่ไม่ได้เผยแพร่

การออกแบบ SHA มีลักษณะคล้ายกันมากกับ MD4 ของศาสตราจารย์ Rivest นอกจากนั้น SHA-1 ยังได้รับการกำหนดให้เป็นมาตรฐาน ANSI X9.30 (part2) ด้วย

ข่าวสารที่จะเป็นข้อมูลขาเข้าของ SHA จะต้องมีความยาวไม่เกิน 2^{64} บิต ผลลัพธ์หรือเมสเสจไดเจสที่ได้จะมีขนาด 160 บิต SHA-1 ทำงานได้ช้ากว่า MD5 อย่างไรก็ตามขนาดของผลลัพธ์ที่มากกว่าทำให้ SHA-1 ปลอดภัยกว่าในเรื่องของการชนกันของผลลัพธ์ สามารถหาข้อมูลเพิ่มเติมได้จาก [Pre93] และ [Rob95b]

2.3.3 **อัลกอริทึมในกลุ่มฟังก์ชันแฮชอย่างง่าย** (Simple Hash Function)

ฟังก์ชันแฮชอย่างง่ายที่ใช้ในงานวิจัย มีทั้งหมด 4 ฟังก์ชันด้วยกัน คือ

- Digit Analysis Method
- Division Method
- Folding Method
- Midsquare Method

Folding method และ Digit Analysis จะได้ผลลัพธ์ที่มีความยาวรหัสคงที่ (Fixed Length) ส่วน Division method และ Midsquare method จะได้ผลลัพธ์ที่มีความยาวรหัสไม่คงที่ (Variable Length) แต่ละฟังก์ชันได้กำหนดให้มีขนาดของค่าแฮช 3 ค่า คือ 2,4 และ 8 ไบต์ รายละเอียดการทำงานของแต่ละฟังก์ชันมีดังนี้

<u>Digit Analysis Method</u>

วิธีการของ Digit Analysis คือการเลือกตัวอักษรหรือตัวเลขเพียงบางตำแหน่งจากข้อมูลต้นฉบับนำมาแปลงเป็นรหัสแอสกีและใช้เป็นค่าแฮช เช่น เลือกอักษรที่ตำแหน่ง 16 ถึง 19 (4 ตัวอักษร หรือ 4 ไบต์) ยกตัวอย่างเช่น มีข้อมูล “http://www.chula.ac.th/” ตัวอักษรที่เลือกมา คือ “a.ac” แปลงเป็นรหัสแอสกีได้ค่าแฮช ดังนี้ คือ 61 2E 61 63 (เลขฐานสิบหก) วิธีการนี้จะได้ค่าแฮชที่มีขนาดคงที่

Division Method

การหาค่าแฮชของ Division Method จะเริ่มจากการกำหนดขนาดของตัวหาร ซึ่งอาจจะเป็น 2 หรือ 4 ไบต์ จากนั้นจะหาจำนวนเฉพาะสูงสุดเท่าที่ขนาดของตัวหารจะมีได้ เช่น ขนาดตัวหาร 2 ไบต์ จะใช้ 251 เป็นตัวหาร เป็นต้น จากนั้นจะนำข้อมูลมาที่ต้องการหาค่าแฮชมาประมวลผลครั้งละ 1 บล็อก หารด้วยตัวหารที่กำหนด เศษที่ได้จากการหารคือค่าแฮชของบล็อกนั้น เมื่อหมดข้อมูลที่นำมาหาร ค่าแฮชก็คือ เศษที่ได้จากการหารมาเรียงต่อกันตามลำดับนั่นเอง ค่าแฮชที่ได้จากวิธีการนี้จะมีขนาดไม่คงที่ ขึ้นกับความยาวของข้อมูลที่จะมาทำการแฮช และขนาดของตัวการที่ใช้

Midsquare Method

Midsquare Method จะแบ่งข้อมูลขาเข้าออกเป็นบล็อกเช่นกัน จากนั้นจะนำข้อมูลในแต่ละบล็อกนั้นมายกกำลังสอง และนำเลขนัยสำคัญต่ำสุด 8 บิตมาเป็นค่าแฮชของบล็อกนั้น เมื่อประมวลผลจนหมดข้อมูลขาเข้าจะนำผลลัพธ์ที่ได้แต่ละบล็อกมาเรียงต่อกันและใช้เป็นค่าแฮช ซึ่งขนาดของค่าแฮชจะไม่คงที่ขึ้นกับความยาวของข้อมูล

Folding Method

วิธี Folding method ก็คือการพับไบต์ข้อมูลที่ต้องการตามจำนวนบิตที่ต้องการ บิตที่พับมารวมกันนั้น จะทำโอเปอเรชั่น XOR กัน ผลลัพธ์ที่ได้ คือ ค่าแฮช ตัวอย่าง เช่น ข้อมูล 44548897ACC จะนำมาหาค่าแฮชด้วยวิธีการนี้ ดังนี้

```
0100 0100 0101 0100 1000 1000 1001 0111 1010 1100 1100
```

หลังจากการผ่านการพับข้อมูลจะเป็นดังนี้

```
0100 0100
0101 0100
1000 1000
1001 0111
1010 1100
1100 0000
```

ผลลัพธ์ที่ได้ คือ `0110 0011`

ค่าแฮชที่ได้จะมีค่าคงที่ ไม่ขึ้นกับความยาวของข้อมูลที่นำมาหาค่าแฮชนั้น แต่ขึ้นกับความยาวของการพับบิตที่กำหนดขึ้น