บทที่ 2
ความรู้พื้นฐานและงานวิจัยที่เกี่ยวข้อง

2.1 หลักการฝังและตรวจจับลายน้ำทั่วไป

การฝังและตรวจจับลายน้ำวิธีต่าง ๆ มักมีหลักการโดยรวมคล้ายกัน คือฝังข้อมูลลายน้ำลงในข้อมูลต้นฉบับและสามารถดึงข้อมูลนั้นออกมาได้หากมีกุญแจที่ถูกต้อง ทั้งนี้อาจมีกระบวนการเพิ่มเติมแล้วแต่กรณี เช่น อาจมีการเปลี่ยนแปลงค่าของจุดภาพหรือสัมประสิทธิ์การแปลงไปเล็กน้อยเพื่อช่วยในเรื่องการมองไม่เห็น (Imperceptibility) ของลายน้ำ โดยสรุปวิธีการฝังและตรวจจับลายน้ำจะเริ่มจากการออกแบบสัญญาณลายน้ำ W ซึ่งจะถูกฝังเข้าไปในสัญญาณต้นฉบับ โดยทั่วไปลายน้ำจะถูกสร้างโดยอาศัยกุญแจ K ร่วมกับข้อมูลเกี่ยวกับลิขสิทธิ์ I ดังนี้

$$W = f_0(I, K) \tag{2.1}$$

และบางครั้งอาจจะขึ้นกับสัญญาณต้นฉบับ X ที่จะถูกฝังด้วย

$$W = f_0(I, K, X) \tag{2.2}$$

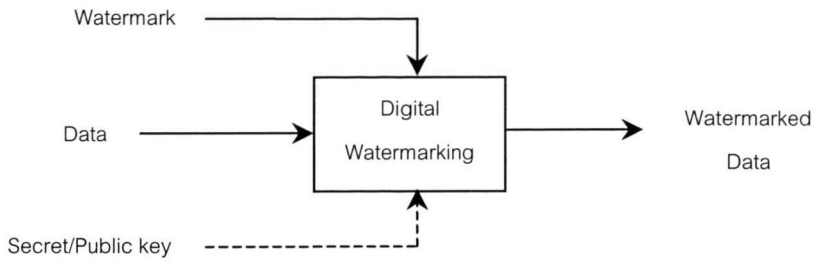
หลังจากได้ลายน้ำแล้ว จึงออกแบบวิธีการฝังลายน้ำซึ่งจะฝังลายน้ำ W ที่สร้างลงในข้อมูลต้นฉบับ X เพื่อให้ได้ข้อมูลที่ถูกฝังแล้ว Y ดังแสดงในรูปที่ 2.1

$$Y = f_1(X, W) \tag{2.3}$$

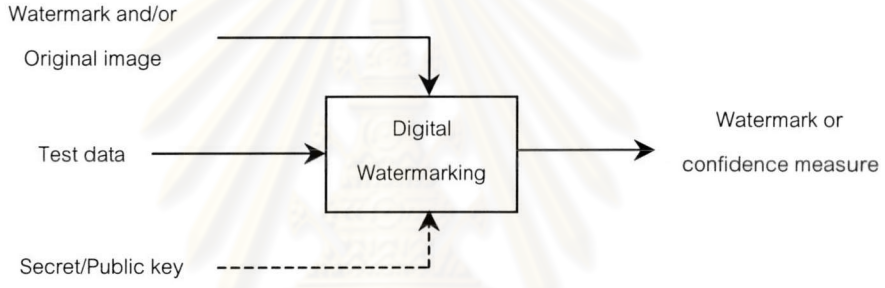
วิธีการตรวจจับลายน้ำจากสัญญาณที่ผสมกันอยู่สามารถทำได้ทั้งแบบที่ใช้และไม่ใช้ข้อมูลต้นฉบับ X โดยใช้กุญแจ K ร่วมด้วย ดังแสดงในรูปที่ 2.2

$$\hat{I} = g(X, Y, K) \tag{2.4}$$

$$\hat{I} = g(Y, K) \tag{2.5}$$



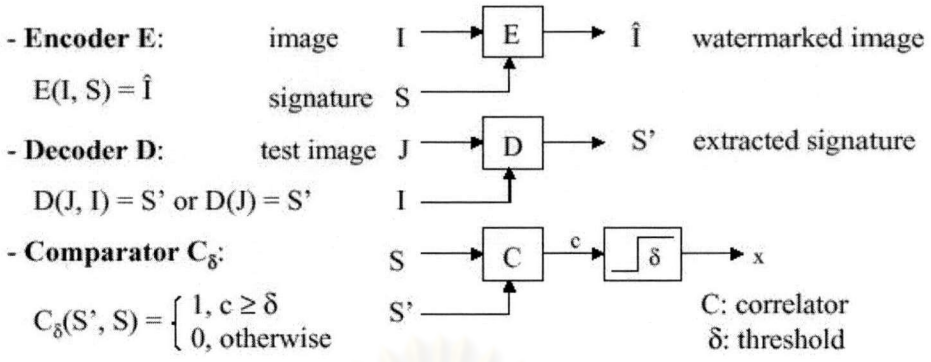
รูปที่ 2.1 วิธีการฝังลายน้ำทั่วไป



รูปที่ 2.2 วิธีการตรวจจับลายน้ำทั่วไป

2.2 การแอบอ้างสิทธิ์เกี่ยวกับลายน้ำ

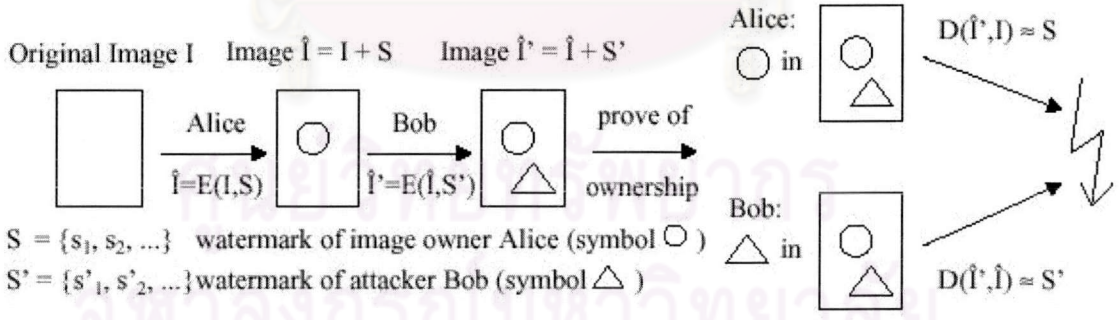
วัตถุประสงค์สำคัญประการหนึ่งของการใส่ลายน้ำในข้อมูล ก็คือเพื่อแสดงสิทธิ์ความเป็นเจ้าของข้อมูลนั้น แต่อาจมีบุคคลอื่นซึ่งมิใช่เจ้าของข้อมูลกระทำการบางอย่างกับวิธีใส่ลายน้ำที่ยังมีช่องโหว่ทำให้ตนเองมีหลักฐานที่ทำให้เชื่อได้ว่าเป็นเจ้าของข้อมูลนั้นได้ พิจารณาการใส่ลายน้ำทั่วไป หากแทนขั้นตอนการใส่ลายน้ำด้วยสัญลักษณ์ดังรูปที่ 2.3 โดยหลักการแล้วมี 2 ขั้นตอนคือ $E(.)$ คือการใส่ลายน้ำ และ $D(.)$ คือการตรวจจับลายน้ำจากรูปทดสอบ J โดยอาจใช้ต้นฉบับ I หรือไม่ใช้ ส่วน $C_\delta(.)$ คือตัวเปรียบเทียบว่าลายน้ำสองแบบเหมือนกันหรือไม่เป็นส่วนเพิ่มเติม จากรูปนี้อาจเขียนได้ว่าการใส่ลายน้ำคือกระบวนการ (E, D, C_δ)



รูปที่ 2.3 กระบวนการใส่ลายน้ำ

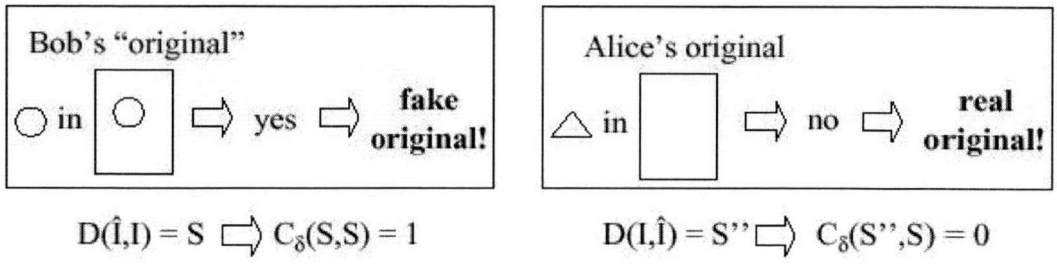
2.2.1 Multiple watermarking

การแอบอ้างสิทธิ์แบบที่ง่ายที่สุดคือการใส่ลายน้ำซ้ำลงไปในรูปที่มีลายน้ำอยู่แล้ว (multiple watermarking) ตัวอย่างเช่น ในรูปที่ 2.4 จากรูปต้นฉบับ I Alice ใส่ลายน้ำ S ของตนลงไปได้รูป $\hat{I}$ ส่วน Bob ต้องการเป็นแสดงความเป็นเจ้าของรูปนั้น จึงใส่ลายน้ำ S' ลงไปในรูป $\hat{I}$ ได้รูป $\hat{I}'$ แล้วนำไปแอบอ้างว่าเป็นของตน โดยในที่นี้การมองเห็นไม่ใช่ประเด็นสำคัญ เนื่องจากรูป $\hat{I}$ และ $\hat{I}'$ นั้นถ้ามองด้วยตาเปล่าจะไม่เห็นความแตกต่าง เมื่อเกิดเหตุการณ์ดังนี้ หากนำรูปที่สงสัยไปตรวจจับลายน้ำก็จะปรากฏลายน้ำทั้งของ Alice และ Bob จึงไม่สามารถพิสูจน์ได้ว่าใครเป็นเจ้าของที่แท้จริง



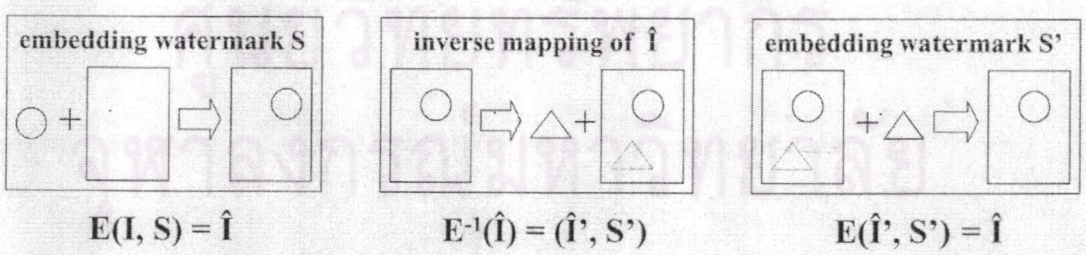
รูปที่ 2.4 Multiple watermarking

วิธีแก้ปัญหานี้ทำได้ดังรูปที่ 2.5 วิธีการคือ นำรูปต้นฉบับของแต่ละคนมาพิสูจน์ รูปต้นฉบับของผู้ที่เป็นเจ้าของที่แท้จริงจะต้องไม่มีลายน้ำของอีกคนหนึ่งอยู่

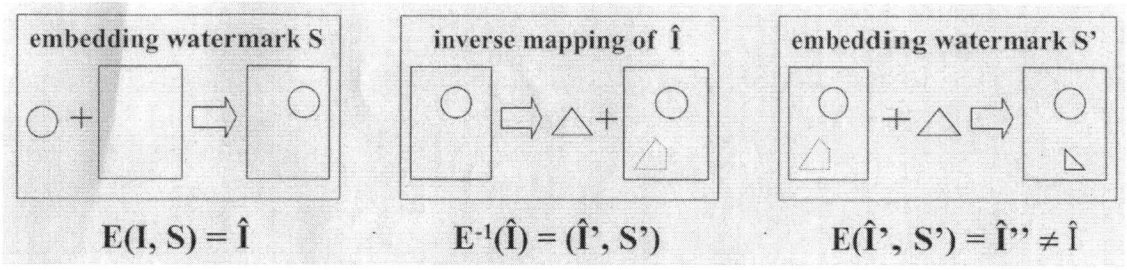


รูปที่ 2.5 การแก้ปัญหา multiple watermarking

นอกจากวิธีการใส่ลายน้ำซ้ำลงไปข้างต้นแล้ว ยังมีการแอบอ้างสิทธิ์แบบอื่น ๆ อีก โดยอาศัยช่องโหว่ของการใส่ลายน้ำบางวิธีซึ่งเป็นกระบวนการที่สามารถทำกลับได้ การทำกลับได้นี้ก็มี 2 แบบ คือ invertible watermarking scheme และ quasi-invertible watermarking scheme แสดงในรูปที่ 2.6 และ 2.7 ตามลำดับ ลักษณะของการใส่ลายน้ำที่เป็นแบบ invertible คือหากมีรูปที่ใส่ลายน้ำ $\hat{I}$ อยู่ซึ่งเกิดจากรูปต้นฉบับ I และลายน้ำ S จะสามารถทำกลับการใส่ลายน้ำได้รูป $\hat{I}'$ และลายน้ำ S' ออกมาได้โดย $\hat{I}'$ นี้มองด้วยตาเปล่ามีลักษณะเหมือน $\hat{I}$ กรณีเช่นนี้ทำให้เกิดการอ้างสิทธิ์ในรูปรูปเดียวคือ $\hat{I}$ ได้ โดยฝ่ายที่ใช้การทำกลับจะอ้างว่ารูปนี้เกิดจากต้นฉบับ $\hat{I}'$ และลายน้ำ S' ของตน ส่วนการทำกลับได้แบบ quasi-invertible ก็มีลักษณะที่คล้ายกัน ต่างกันตรงที่เมื่อได้ $\hat{I}'$ และ S' ออกมาแล้วทำการใส่ลายน้ำจะไม่ได้ $\hat{I}$ แต่จะได้รูปอีกรูปหนึ่งแทนด้วย $\hat{I}''$ ซึ่งมองด้วยตาเปล่าเหมือนกับ $\hat{I}$

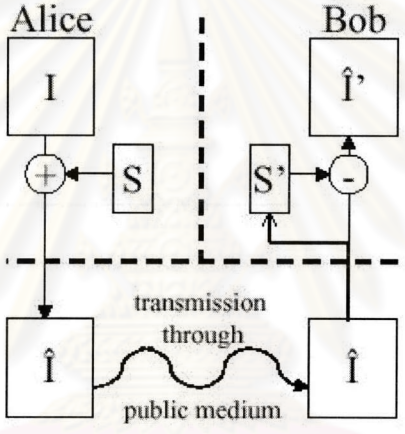


รูปที่ 2.6 invertible watermarking scheme



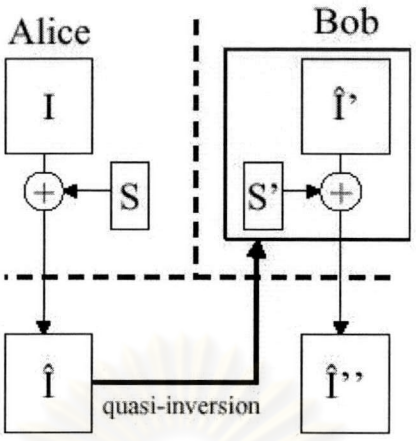
รูปที่ 2.7 quasi-invertible watermarking scheme

2.2.2 SWICO-Attack



รูปที่ 2.8 SWICO-Attack

จากรูป Bob ต้องการอ้างว่าตนเองเป็นเจ้าของรูป $\hat{I}$ ซึ่งเป็นของ Alice โดย Alice ใส่ลายน้ำเอาไว้ด้วยคือ S เขาจะนำรูป $\hat{I}$ มาทำกลับการใส่ลายน้ำ (invertible watermarking scheme) เช่น อาจสร้างลายน้ำของเขา S' ขึ้นมา แล้วนำไปลบออกจาก $\hat{I}$ ได้รูปต้นฉบับปลอม $\hat{I}'$ วิธีการนี้เรียกว่า Single Watermarked Image Counterfeit Original (SWICO) จะเห็นได้ว่าทั้ง Bob และ Alice สามารถอ้างว่าตนเป็นเจ้าของรูป $\hat{I}$ ได้ทั้งคู่ โดยแต่ละคนต่างก็มีรูปต้นฉบับของตน



รูปที่ 2.9 TWICO-Attack

2.2.3 TWICO-Attack

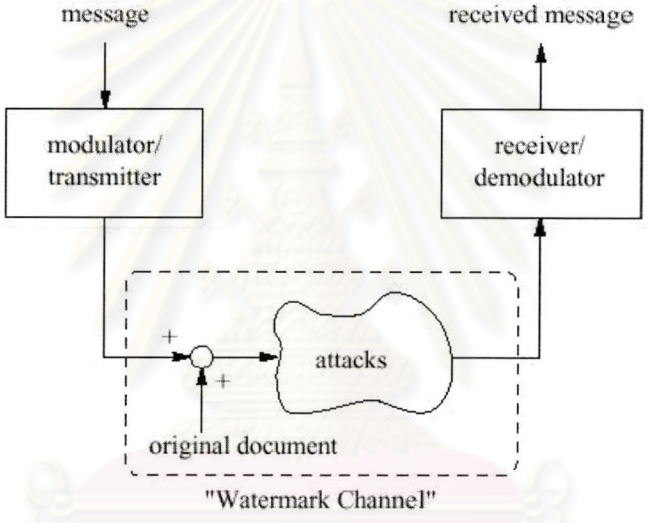
TWICO (Twin Watermark Images Counterfeit Original) เป็นการใช้การทำกลับการใส่ลายน้ำเช่นกัน แต่มีลักษณะเป็น quasi-invertible กล่าวคือ รูป $\hat{I}$ จะถูกนำมาผ่านกระบวนการ quasi-inversion ได้รูปต้นฉบับปลอม $\hat{I}'$ และลายน้ำอีกแบบหนึ่ง S' ออกมา โดยถ้าจะให้ได้รูปที่ใส่ลายน้ำ Bob ก็จะต้องนำ $\hat{I}'$ มาฝังด้วยลายน้ำ S' ได้รูป $\hat{I}''$ ที่มองด้วยตาเปล่าแล้วเหมือนรูป $\hat{I}$ ของ Alice ทำให้เขาอ้างว่าตนเองเป็นเจ้าของรูปที่แม้จะไม่ได้เหมือนกันทุกประการแต่มองดูแล้วเหมือนรูปเดียวกันนี้ได้เช่นเดียวกับ Alice

จะเห็นได้ว่า การแอบอ้างสิทธิ์ด้วยวิธีทั้งสามแบบข้างต้นได้แก่ multiple watermarking, SWICO-Attack, และ TWICO-Attack มักมีสาเหตุมาจากการที่ต้องใช้รูปต้นฉบับในการตรวจจับ (ยกเว้น multiple watermarking) ทำให้เกิดการสร้างรูปต้นฉบับปลอมและลายน้ำอีกแบบขึ้น ดังนั้นหากจะแก้ปัญหาจึงควรตรวจจับลายน้ำโดยไม่ต้องใช้รูปต้นฉบับ แต่หากทำเช่นนั้นก็จำเป็นต้องยอมแลกกับความสูญเสียบางอย่างไป คือวิธีที่ตรวจจับลายน้ำโดยไม่ใช้ต้นฉบับจะมีความทนทาน (Robustness) น้อยกว่าและสามารถใส่ข้อมูลได้น้อยกว่า อย่างไรก็ตามก็จำเป็นต้องพิจารณาถึงความจำเป็นและลักษณะการใช้งานของลายน้ำด้วย ในบาง application อาจไม่จำเป็นต้องให้ลายน้ำมีความทนทานสูงหรือใส่ข้อมูลจำนวนมากก็ได้

2.3 Spread Spectrum watermarking

การใส่ลายน้ำสามารถพิจารณาเป็นระบบสื่อสารได้ดังรูปที่ 2.10 ข้อความ (message) ซึ่งเป็นลายน้ำถูกส่งผ่านช่องทางสื่อสาร (watermark channel) โดยมีข้อมูลที่ถูกฝังลายน้ำนั้นรวมทั้ง attacks ต่าง ๆ เป็นสัญญาณรบกวน ซึ่ง attacks ต่าง ๆ นี้แบ่งได้ 2 ประเภทคือ

- erasure / alteration คือ การพยายามเอาลายน้ำออกหรือเปลี่ยนแปลงส่วนที่มีลายน้ำจนไม่สามารถตรวจจับได้
- jamming คือ การเปลี่ยนแปลงส่วนที่มีลายน้ำโดยไม่ได้มีจุดประสงค์ที่จะเอาลายน้ำออกแต่ต้องการให้ตรวจจับได้ยากขึ้น



รูปที่ 2.10 communication model ของระบบการใส่ลายน้ำ

ปัจจุบันมีการใส่ลายน้ำหลายวิธีที่ใช้หลักการของสเปรดสเปกตรัม (Spread Spectrum) ทั้งกับข้อความ (text watermarking), รูปภาพ (image watermarking), หรือสัญญาณวิดีโอ (video watermarking) [15] ลายน้ำจะถูกส่งโดยมอดูเลตกับสัญญาณที่มีลักษณะคล้ายสัญญาณรบกวนซึ่งมีขนาดแอมปลิจูดต่ำ เพื่อให้ได้คุณสมบัติสามประการที่สำคัญของลายน้ำคือ ความปลอดภัย (security), ความทนทาน (robustness) และการสังเกตด้วยตาเปล่าไม่เห็น (imperceptibility) ตัวอย่างรูปแบบหนึ่งของสเปรดสเปกตรัมที่ใช้คือ direct-sequence spread spectrum (DSSS) ซึ่งมีรายละเอียดดังนี้

2.3.1 Transmission (Watermark Embedding)

ลายน้ำหรือข้อมูลที่จะส่งคือ $b_0 b_1 b_2 \ldots$ ซึ่งแต่ละบิตมีค่าเป็น +1 หรือ –1 ก่อนที่จะส่งบิตข้อมูลนี้แต่ละบิตจะถูกเรียงซ้ำ N ครั้ง ได้ redundant message $m[n]$ เช่น ถ้า N=3 จะได้ $m[n]$ คือ $b_0 b_0 b_0 b_1 b_1 b_1 b_2 b_2 b_2 \ldots$ จากนั้น $m[n]$ จะถูกมอดูเลตด้วยสัญญาณ $c[n]$ และคูณด้วย $\sqrt{E/N}$ ดังนั้นสัญญาณที่จะส่งคือ $s[n] = \sqrt{E/N} \cdot m[n] \cdot c[n]$ แสดงการฝังลายน้ำดังรูปที่ 2.11 โดย $s[n]$ จะถูกส่งโดยการเปลี่ยนแปลงลักษณะบางอย่างของข้อมูลต้นฉบับ เช่น สำหรับการใส่ลายน้ำกับข้อความก็อาจใช้การเปลี่ยนแปลงระยะบรรทัด หรือถ้าเป็นการใส่ลายน้ำในรูปภาพก็ใช้การบวก $s[n]$ เข้าไปโดยตรงกับค่าของแต่ละจุดภาพหรือสัมประสิทธิ์การแปลงก็ได้

2.3.2 Spreading Sequence

สัญญาณ $c[n]$ ซึ่งนำไปมอดูเลตบิตข้อมูลที่เรียงซ้ำแล้ว เรียกว่า spreading sequence ซึ่งมีคุณสมบัติพิเศษดังนี้

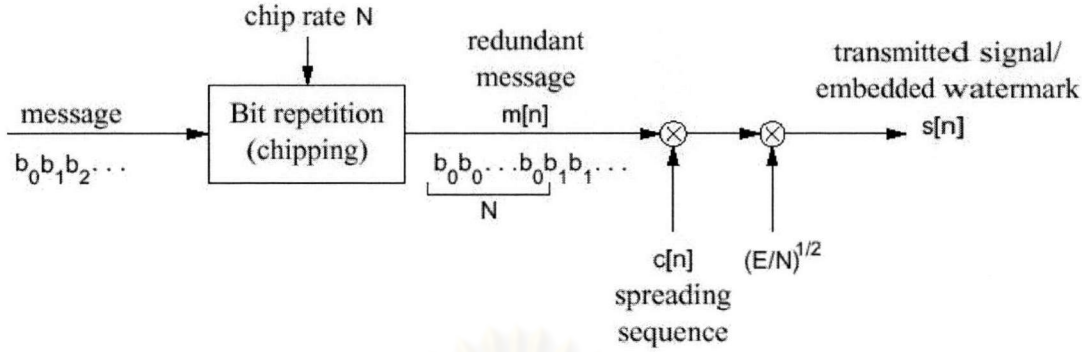
$$c[n] \in \{+1, -1\} \tag{2.6}$$

$$\sum_{n=0}^{N-1} c[n] \cdot c[n + Ni] = N \tag{2.7}$$

$$\frac{1}{N} \sum_{n=0}^{N-1} c[n] = 0 \tag{2.8}$$

$$\frac{1}{N} \sum_{n=0}^{N-1} c[n] \cdot c[n + k] = \delta[k], \qquad 0 \le k \le N - 1 \tag{2.9}$$

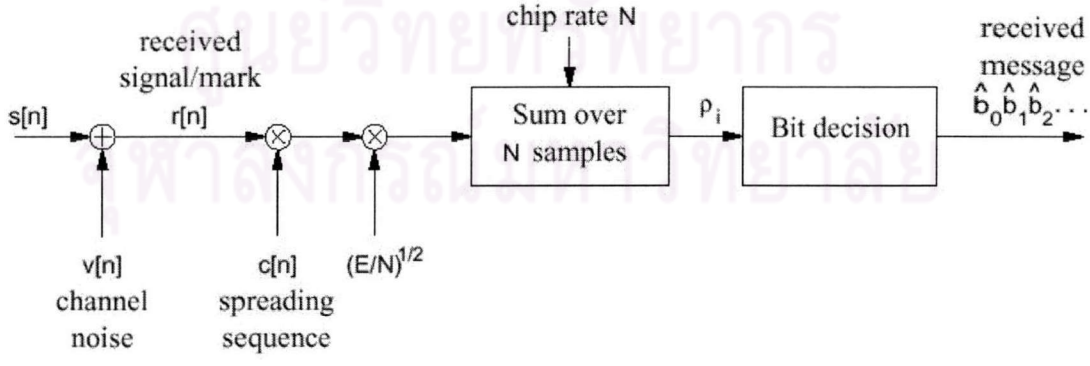
สมการ (2.6) แสดงให้เห็นว่า $c[n]$ เป็นสัญญาณไบนารี แต่ก็สามารถเป็นสัญญาณที่ไม่ใช่ไบนารี เช่น ลำดับที่มีการแจกแจงแบบเกาส์ได้เช่นกัน สมการ (2.7), (2.8) และ (2.9) เรียกว่าคุณสมบัติความเป็นคาบ (periodicity), ค่าเฉลี่ยเป็นศูนย์ (zero-mean) และ ความเป็นคาบของค่าอัตสหสัมพันธ์ (periodic autocorrelation) ตามลำดับ เพื่อความปลอดภัย $c[n]$ ควรถูกสร้างได้อย่างไม่ซับซ้อนนักด้วยกุญแจ (key) แต่จะไม่สามารถสร้างได้หากไม่มีกุญแจโดยใช้เพียงส่วนหนึ่งของสัญญาณนั้น สัญญาณที่มีคุณสมบัติดังกล่าวข้างต้นเรียกว่า pseudo-noise signals



รูปที่ 2.11 การใส่ลายน้ำโดยใช้วิธีสเปรดสเปกตรัม

2.3.3 Watermark Channel

หลังจากการใส่ลายน้ำ (watermark embedding) หรือผ่านขั้นตอนการส่งสัญญาณ (transmission) แล้ว สัญญาณ $s[n]$ ก็จะผ่านช่องสัญญาณที่มีสัญญาณรบกวนซึ่งโดยทั่วไปมักจะพิจารณาให้เป็น additive white Gaussian noise (AWGN) $v[n]$ มีค่าความแปรปรวน σ_v^2/N ดังนั้นสัญญาณที่รับได้จึงเขียนได้เป็น $r[n]=s[n]+v[n]$ ในกรณีของการใส่ลายน้ำ สัญญาณรบกวนนี้คือข้อมูลต้นฉบับและ attacks ต่าง ๆ นั่นเอง ส่วนของ attacks ก็อาจจะได้แก่การถ่ายเอกสารหรือการพิมพ์สำหรับการใส่ลายน้ำในข้อความ หรือการบีบอัดแบบ JPEG หรือ MPEG สำหรับการใส่ลายน้ำในรูปภาพและวิดีโอตามลำดับ ช่องสัญญาณและส่วนของภาครับแสดงดังรูปที่ 2.12



รูปที่ 2.12 ช่องสัญญาณและส่วนของภาครับในวิธีการใส่ลายน้ำแบบสเปรดสเปกตรัม

2.3.4 Reception (Watermark Recovery)

เมื่อภาครับได้รับสัญญาณ $r[n]$ ก็จะทำการตรวจจับข้อมูลที่ถูกส่งมาหรือลายน้ำที่ฝังลงไป ที่ภาครับจำเป็นต้องมี spreading sequence $c[n]$ เดียวกับที่ใช้ในการส่งข้อมูลอยู่ด้วยเพื่อให้สามารถดึงข้อมูลนั้นออกมาได้ การตรวจจับจะใช้ correlation detector ซึ่งจะคำนวณค่าสหสัมพันธ์

$$\rho_i = \sqrt{E/N}\sum_{n=0}^{N-1} r[n-Ni]\cdot c[n] \text{ สำหรับแต่ละค่า } i \qquad (2.10)$$

จากคุณสมบัติของ $c[n]$ เราจะได้ว่า $\rho_i == Eb_i + \sqrt{E/N}\sum_{n=0}^{N-1} v[n]\cdot c[n]$ โดยถ้า $\rho_i \geq 0$ ที่ภาครับจะตัดสินว่า $\hat{b}_i = +1$ ไม่เช่นนั้นจะตัดสินว่า $\hat{b}_i = -1$

ค่าที่ใช้วัด performance ได้แก่ อัตราส่วนกำลังของสัญญาณต่อสัญญาณรบกวน (signal-to-noise ratio : SNR) ในกรณีนี้ $SNR_{std} = E/\sigma_v^2$ และความน่าจะเป็นของความผิดพลาด (probability of error : P(E)) ซึ่งเป็นความน่าจะเป็นที่จะตรวจจับ $\hat{b}_i$ ผิดพลาด ในกรณีนี้ $P_{E,std} = Q\left(\sqrt{E/\sigma_V^2}\right)$ โดย $Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-y^2/2}dy$

สำหรับ AWGN ที่มีกำลังจำกัด สเปรดสเปกตรัมไม่ได้ให้ผลที่ดีกว่าการมอดูเลตวิธีอื่น แต่หากกำลังของ AWGN σ_v^2/N มีค่าจำกัด จะมีข้อได้เปรียบวิธีอื่น ๆ ดังนี้

- *Imperceptibility* แอมปลิจูดของสัญญาณที่ส่งไป $s[n]$ มีค่าเท่ากับ $\sqrt{E/N}$ ที่แต่ละค่า n ดังนั้นหากเลือกใช้ค่า N ที่ใหญ่มาก ๆ $\sqrt{E/N}$ ก็จะมีค่าเล็กมากได้เท่าที่ต้องการเพื่อให้ลายน้ำไม่ทำให้เกิดการมองเห็น แต่ในขณะเดียวกันกำลังรวมของสัญญาณของทุก N samples ยังมีค่าเท่ากับ E เช่นเดิม นั่นคือเราสามารถส่งสัญญาณหรือฝังลายน้ำที่มีกำลังสูง E โดยใช้การเปลี่ยนแปลงแอมปลิจูดในบริเวณกว้างได้ นอกจากนี้เนื่องจากสเปกตรัมกำลังของ $s[n]$ มีค่า $\Phi_{SS}\left(e^{jw}\right) = E$ ซึ่งแสดงว่า $s[n]$ มีลักษณะคล้าย white noise จะไม่มีบริเวณที่สูงขึ้นมาในสเปกตรัมกำลังให้สังเกตเห็นได้ว่ามีการส่งสัญญาณหรือลายน้ำ ทำให้ปลอดภัยจากผู้ที่จะเอาลายน้ำออกได้

- *Security* หากไม่มี spreading sequence $c[n]$ จะไม่สามารถดึงข้อมูลที่ถูกฝังอยู่ออกมาได้ ถ้าจะหา $c[n]$ จากเอกสารที่ถูกฝังลายน้ำก็ทำได้ยากเพราะ $s[n]$ มีลักษณะคล้าย white noise มีแอมปลิจูดต่ำ และถึงแม้จะสามารถประมาณบางส่วนของ $c[n]$ ได้ก็ยังยากที่

จะหา $c[n]$ ทั้งหมดเพราะคุณสมบัติความเป็น pseudo-noise ของมัน ดังนั้นกล่าวได้ว่าลายน้ำมีความปลอดภัย

- *Robustness* ผู้ที่พยายามจะเอาลายน้ำออกหรือจะทำการ jamming จะไม่สามารถทำได้เพราะไม่รู้ตำแหน่งหรือบริเวณที่เอกสารถูกเปลี่ยนแปลงด้วย $s[n]$ และหากจะทำให้ลายน้ำไม่สามารถตรวจจับได้ก็จะต้องเปลี่ยนแปลงทุกตำแหน่งของเอกสารซึ่งจะทำมากเกินไปไม่ได้เพราะจะทำให้เอกสารนั้นเสียไป

- *Multiple Watermarks* ลายน้ำหรือข้อความมากกว่า 1 แบบสามารถถูกฝังลงไปได้ เช่น ถ้ามีข้อความ $m_k[n]$ และ pseudo-noise sequences $c_k[n]$ จะสามารถออกแบบให้แต่ละ sequence ตั้งฉากกัน (mutually orthogonal) ได้ นั่นคือ $\sum_{n=0}^{N-1} c_k[n] \cdot c_l[n] = 0, \quad k \neq l$ ทำให้แต่ละข้อความ $m_k[n]$ สามารถถูกฝังและตรวจจับได้อย่างเป็นอิสระต่อกัน

ตัววัดความทนทานของการใส่ลายน้ำแบบสเปรดสเปกตรัมคือ SNR และ P(E) โดยสำหรับ σ_v^2 และ SNR คงที่ P(E) จะดีขึ้นถ้ากำลังของการส่ง E และ chip rate N เพิ่มขึ้น อย่างไรก็ตามหาก $\sqrt{E/N}$ มีค่าสูงเกินไปก็จะทำให้ลายน้ำเกิดการมองเห็นและหาก N เพิ่มขึ้นจำนวนของบิตข้อมูลที่จะฝังลงไปได้จะน้อยลง ดังนั้นอัลกอริทึมในการใส่ลายน้ำจึงต้องพิจารณาถึงระดับความทนทานและการมองไม่เห็นของลายน้ำให้เหมาะสมสำหรับความยาวของข้อความที่จะฝังที่ต้องการ

2.4 Hash Function [16]

hash function H คือ การแปลงแบบหนึ่งที่รับข้อมูลนำเข้า m แล้วให้ข้อมูลออกเป็นสายอักขระขนาดคงที่ (fixed-size string) เรียกว่า hash value h ดังสมการ (2.11)

$$h = H(m) \tag{2.11}$$

hash function ที่มีลักษณะข้างต้นสามารถนำไปใช้ในการคำนวณได้หลายอย่าง แต่หากจะใช้ในการเข้ารหัสลับ (Cryptography) จำเป็นจะต้องมีคุณสมบัติเพิ่มเติม คุณสมบัติพื้นฐานของ cryptographic hash function มีดังนี้

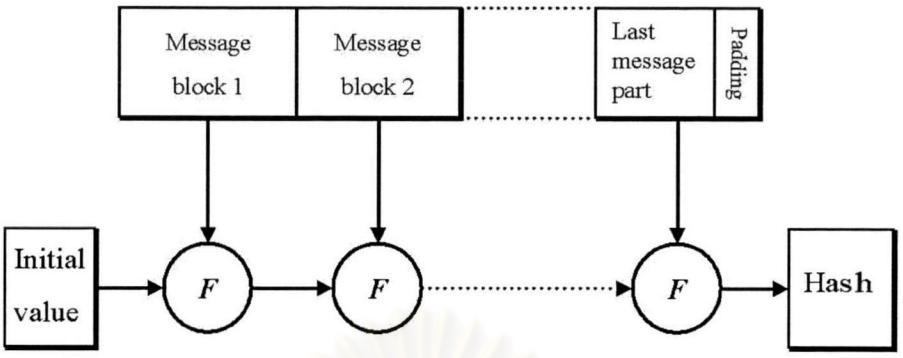
- ข้อมูลนำเข้ามีขนาดเท่าใดก็ได้
- ข้อมูลออกมีขนาดคงที่

- H(x) สามารถคำนวณได้อย่างไม่ยุ่งยากซับซ้อนสำหรับ x ใดใด
- H(x) เป็นกระบวนการทางเดียว (one-way operation)
- H(x) เป็น collision-free

H(x) มีลักษณะเป็นกระบวนการทางเดียวก็ต่อเมื่อสำหรับ hash value h จะไม่สามารถหาข้อมูลนำเข้า x ที่ให้ hash value h = H(x) ได้ สำหรับกรณีที่กำหนด message x ใดใดแล้วไม่สามารถหา message y ที่ทำให้ H(x) = H(y) ได้ เรียก H ว่าเป็น weakly collision-free hash function และ H จะเรียกว่าเป็น strongly collision-free hash function เมื่อไม่สามารถหา 2 message ใดใด x และ y ที่ทำให้ H(x) = H(y) ได้

อาจกล่าวได้ว่า hash value เป็นตัวแทนของข้อความที่ยาวกว่าหรือเอกสารซึ่งมันถูกหามา ทำให้เรียกได้อีกอย่างหนึ่งว่า "message digest" หรือ "digital fingerprint" ตัวอย่าง hash function ที่เป็นที่รู้จักกันดีได้แก่ MD2, MD5 และ SHA ความสำคัญและการนำ hash function ไปใช้คือเรื่องของ integrity check และลายเซ็นดิจิทัล (digital signature) เนื่องจากสามารถทำได้เร็วกว่าอัลกอริทึมการเข้ารหัสลับ โดยประยุกต์ใช้กระบวนการเข้ารหัสลับกับ hash value ของเอกสารซึ่งมีขนาดเล็กกว่าตัวเอกสารเอง นอกจากนั้นยังทำให้สามารถเปิดเผย hash value หรือ digest นี้ได้โดยไม่จำเป็นต้องเปิดเผยเอกสารหรือข้อความนั้นด้วย

Damgard และ Merkle นิยาม hash function ในรูปของ compression function ซึ่งเป็นแนวคิดที่มีอิทธิพลต่อการใช้ hash function กับการเข้ารหัสลับมาก compression function จะรับข้อมูลนำเข้าขนาดคงที่แล้วให้ข้อมูลออกขนาดคงที่และสั้นลง เมื่อกำหนด compression function จะสามารถนิยาม hash function ได้ว่า เป็นการนำแต่ละบล็อกของข้อความมาทำ compression function ซ้ำกันไปทีละบล็อกจนจบข้อความ ซึ่งข้อความขนาดใดใดนั้นจะถูกแบ่งเป็นบล็อกโดยการเติมส่วน padding เข้าไปก่อนเพื่อให้มีความยาวเป็นจำนวนเท่าของความยาวบล็อก ความยาวบล็อกจะมีค่าเท่าไรขึ้นอยู่กับ compression function กระบวนการของ hash function จะเริ่มจากค่าเริ่มต้น (initial value) และข้อความบล็อกแรกจะถูกนำไปผ่าน compression function F ได้ข้อมูลออกซึ่งจะเป็นข้อมูลนำเข้าของ compression function พร้อมกับข้อความบล็อกต่อไป ทำเช่นนี้จนครบทุกบล็อกจะได้ข้อมูลออกคือ hash value แสดงขั้นตอนของนิยาม hash function นี้ได้ตามรูปที่ 2.13



รูปที่ 2.13 Damgard / Merkle iterative structure for hash functions

2.4.1 MD2, MD4 และ MD5

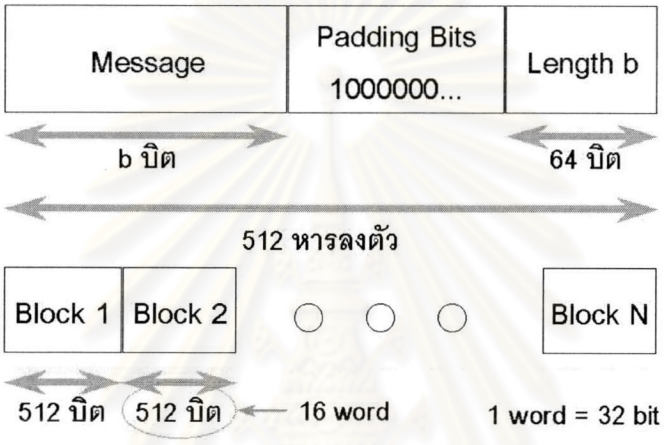
MD2, MD4 และ MD5 เป็นอัลกอริทึมที่ถูกพัฒนาขึ้นโดย Rivest เพื่อใช้สำหรับวิธีลายเซ็นดิจิทัล ซึ่งข้อมูลจะต้องถูกทำให้มีขนาดเล็กลงก่อนเข้ารหัสด้วยกุญแจส่วนตัว (private key) ทั้งสามอัลกอริทึมจะรับข้อมูลนำเข้าขนาดใดใดแล้วให้ข้อมูลออกขนาด 128 บิต ถึงแม้ว่าโครงสร้างของอัลกอริทึมเหล่านี้จะคล้ายกันแต่ MD2 จะมีลักษณะที่ค่อนข้างต่างจาก MD4 และ MD5 โดย MD2 พัฒนาขึ้นเพื่อใช้กับเครื่อง 8 บิตในขณะที่ MD4 และ MD5 ใช้กับเครื่อง 32 บิต MD2 พัฒนาขึ้นในปี 1989 โดย Rivest ข้อความจะถูก padded เพื่อให้ความยาวในหน่วยไบต์หารด้วย 16 ลงตัว จากนั้นต่อท้ายด้วย checksum ขนาด 16 ไบต์แล้วจึงนำไปหา hash value Roger และ Chauvaud พบว่าจะเกิด collisions ขึ้นถ้าไม่มีส่วน checksum

MD4 พัฒนาขึ้นในปี 1990 โดย Rivest ข้อความจะถูก padded เพื่อให้ความยาวในหน่วยบิตเมื่อรวมกับ 64 แล้วหารด้วย 512 ลงตัว แล้วต่อท้ายด้วยส่วนที่แสดงความยาวของข้อความขนาด 64 บิต จากนั้นจะนำไปผ่านกระบวนการในลักษณะของ Damgard / Merkle iterative structure ดังรูปที่ 2.13 โดยขนาดของบล็อกคือ 512 บิตและแต่ละบล็อกจะถูกประมวลผล 3 รอบ Dobbertin ได้แสดงให้เห็นว่า collisions สำหรับ MD4 สามารถเกิดขึ้นได้ และยังแสดงให้เห็นว่า MD4 อัลกอริทึมที่ไม่มีการประมวลผลรอบที่ 3 จะไม่เป็นกระบวนการทางเดียว นั่นแสดงให้เห็นว่า MD4 ได้ถูกพิสูจน์แล้วว่าไม่มีประสิทธิภาพ (broken)

MD5 เกิดขึ้นในปี 1991 โดย Rivest มีลักษณะคล้าย MD4 แต่มีความปลอดภัยสูงกว่าแม้ว่าจะช้ากว่าก็ตาม อัลกอริทึม MD5 จะมีการประมวลผล 4 รอบที่ต่างจาก MD4 เล็กน้อย ส่วนขนาดของ hash value และ padding ยังคงเดิม

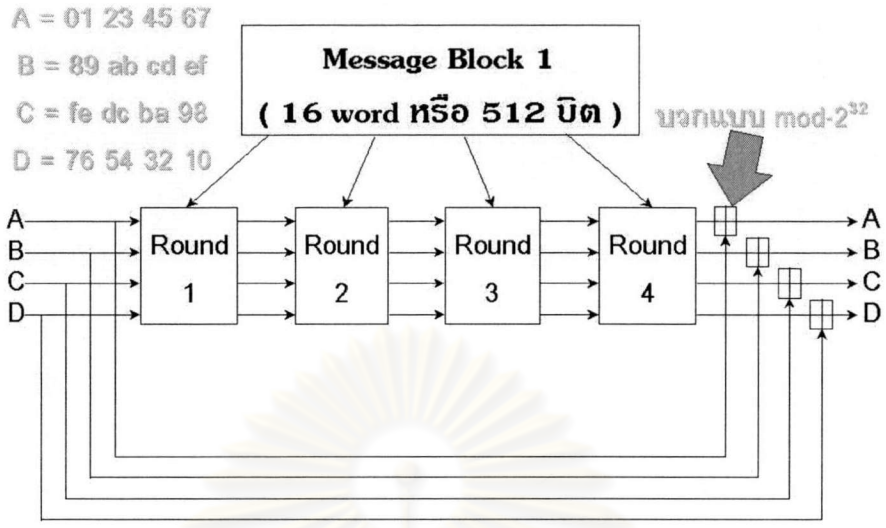
2.4.2 MD5 Algorithm

ขั้นตอนของอัลกอริทึม MD5 มีหลักการเป็นดังรูปที่ 2.13 ข้อความ (message) จะถูกนำมาต่อด้วยส่วน padding bits และส่วนเลขไบนารีแสดงขนาดของข้อความขนาด 64 บิต ความยาวของ padding bits จะเป็นจำนวนบิตที่ทำให้ส่วนของข้อมูลทั้งหมดในหน่วยบิตรวมแล้วหารด้วย 512 ลงตัว แล้วแบ่งข้อมูลดังกล่าวเป็นบล็อกขนาด 512 บิตหรือ 16 word (1 word = 32 บิต) แสดงการแบ่งเป็นบล็อกนี้ดังรูปที่ 2.14



รูปที่ 2.14 การแบ่งบล็อกของ MD5

ขั้นตอนการหา hash value แสดงดังรูปที่ 2.15 ค่าเริ่มต้น A, B, C, D จะถูกนำไปผ่านการประมวลผลรอบที่ 1 (Round 1) โดยมีข้อความบล็อกแรกเป็นข้อมูลนำเข้าด้วย ข้อมูลออกที่ได้จะถูกนำไปเป็นข้อมูลนำเข้าของการประมวลผลรอบต่อไปพร้อมกับข้อความบล็อกนั้น ทำจนถึงรอบที่ 4 ข้อมูลออกที่ได้จะถูกนำไปเป็นข้อมูลนำเข้าของการประมวลผลของบล็อกถัดไป ซึ่งจะมีขั้นตอนเดียวกัน



รูปที่ 2.15 ขั้นตอนการหา hash value ของ MD5

การประมวลผลในแต่ละรอบแสดงดังรูปที่ 2.16 – 2.19 ข้อมูลนำเข้า A,B,C,D และแต่ละ word ของข้อมูลหนึ่งบล็อกจะถูกนำไปผ่านฟังก์ชั่น FF, GG, HH, และ II ในรอบที่ 1, 2, 3 และ 4 ตามลำดับ แสดงการทำงานของแต่ละฟังก์ชั่นดังรูปที่ 2.20 – 2.23 โดยในหนึ่งรอบจะทำฟังก์ชั่นนั้น ๆ 16 ครั้ง ข้อมูลออกของรอบหนึ่ง A, B, C, D จะถูกนำไปเป็นข้อมูลนำเข้าของอีกรอบหนึ่งจนถึงรอบสุดท้ายคือรอบที่ 4 แล้วก็ทำเช่นนี้ไปจนถึงบล็อกสุดท้ายก็จะได้ข้อมูลออก A, B, C, D คือ hash value 4 word ซึ่งมีขนาด 128 บิต

Round 1

T[k] = integer part of
2^{32} x abs(sin(k))

X[0] คือ "Message Block" word ที่ 0
...
X[15] คือ "Message Block" word ที่ 15

1. FF (A, B, C, D, X[0], 7, T[1])	9. FF (A, B, C, D, X[8], 7, T[9])
2. FF (D, A, B, C, X[1], 12, T[2])	10. FF (D, A, B, C, X[9], 12, T[10])
3. FF (C, D, A, B, X[2], 17, T[3])	11. FF (C, D, A, B, X[10], 17, T[11])
4. FF (B, C, D, A, X[3], 22, T[4])	12. FF (B, C, D, A, X[11], 22, T[12])
5. FF (A, B, C, D, X[4], 7, T[5])	13. FF (A, B, C, D, X[12], 7, T[13])
6. FF (D, A, B, C, X[5], 12, T[6])	14. FF (D, A, B, C, X[13], 12, T[14])
7. FF (C, D, A, B, X[6], 17, T[7])	15. FF (C, D, A, B, X[14], 17, T[15])
8. FF (B, C, D, A, X[7], 22, T[8])	16. FF (B, C, D, A, X[15], 22, T[16])

รูปที่ 2.16 การประมวลผลในรอบที่ 1

Round 2

17. GG (A, B, C, D, X[1], 5, T[17])
18. GG (D, A, B, C, X[6], 9, T[18])
19. GG (C, D, A, B, X[11], 14, T[19])
20. GG (B, C, D, A, X[0], 20, T[20])
21. GG (A, B, C, D, X[5], 5, T[21])
22. GG (D, A, B, C, X[10], 9, T[22])
23. GG (C, D, A, B, X[15], 14, T[23])
24. GG (B, C, D, A, X[4], 20, T[24])
25. GG (A, B, C, D, X[9], 5, T[25])
26. GG (D, A, B, C, X[14], 9, T[26])
27. GG (C, D, A, B, X[3], 14, T[27])
28. GG (B, C, D, A, X[8], 20, T[28])
29. GG (A, B, C, D, X[13], 5, T[29])
30. GG (D, A, B, C, X[2], 9, T[30])
31. GG (C, D, A, B, X[7], 14, T[31])
32. GG (B, C, D, A, X[12], 20, T[32])

รูปที่ 2.17 การประมวลผลในรอบที่ 2

Round 3

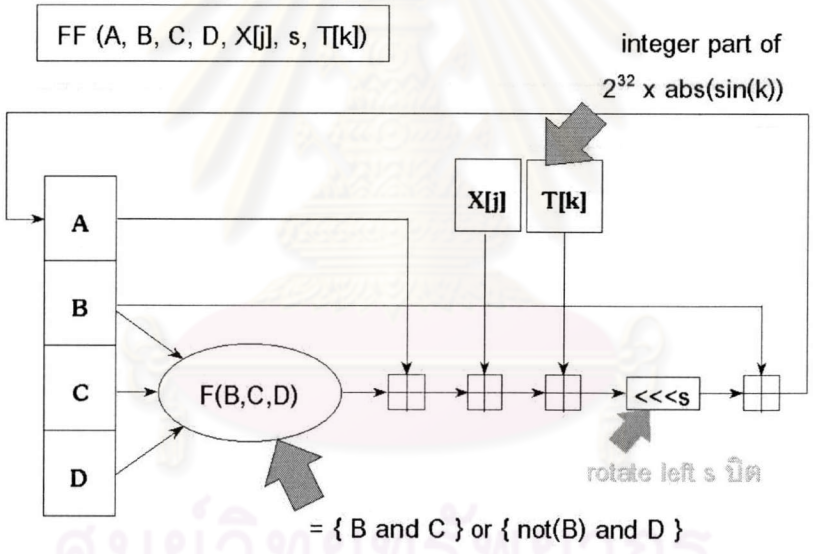
33. HH (A, B, C, D, X[5], 4, T[33])
34. HH (D, A, B, C, X[8], 11, T[34])
35. HH (C, D, A, B, X[11], 16, T[35])
36. HH (B, C, D, A, X[14], 23, T[36])
37. HH (A, B, C, D, X[1], 4, T[37])
38. HH (D, A, B, C, X[4], 11, T[38])
39. HH (C, D, A, B, X[7], 16, T[39])
40. HH (B, C, D, A, X[10], 23, T[40])
41. HH (A, B, C, D, X[13], 4, T[41])
42. HH (D, A, B, C, X[0], 11, T[42])
43. HH (C, D, A, B, X[3], 16, T[43])
44. HH (B, C, D, A, X[6], 23, T[44])
45. HH (A, B, C, D, X[9], 4, T[45])
46. HH (D, A, B, C, X[12], 11, T[46])
47. HH (C, D, A, B, X[15], 16, T[47])
48. HH (B, C, D, A, X[2], 23, T[48])

รูปที่ 2.18 การประมวลผลในรอบที่ 3

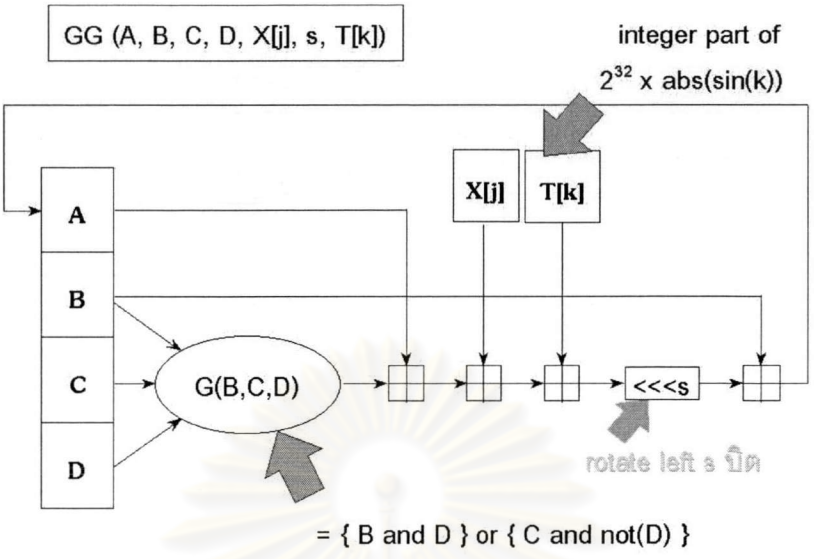
Round 4

49. II (A, B, C, D, X[0], 6, T[49])	57. II (A, B, C, D, X[8], 6, T[57])
50. II (D, A, B, C, X[7], 10, T[50])	58. II (D, A, B, C, X[15], 10, T[58])
51. II (C, D, A, B, X[14], 15, T[51])	59. II (C, D, A, B, X[6], 15, T[59])
52. II (B, C, D, A, X[5], 21, T[52])	60. II (B, C, D, A, X[13], 21, T[60])
53. II (A, B, C, D, X[12], 6, T[53])	61. II (A, B, C, D, X[4], 6, T[61])
54. II (D, A, B, C, X[3], 10, T[54])	62. II (D, A, B, C, X[11], 10, T[62])
55. II (C, D, A, B, X[10], 15, T[55])	63. II (C, D, A, B, X[2], 15, T[63])
56. II (B, C, D, A, X[1], 21, T[56])	64. II (B, C, D, A, X[9], 21, T[64])

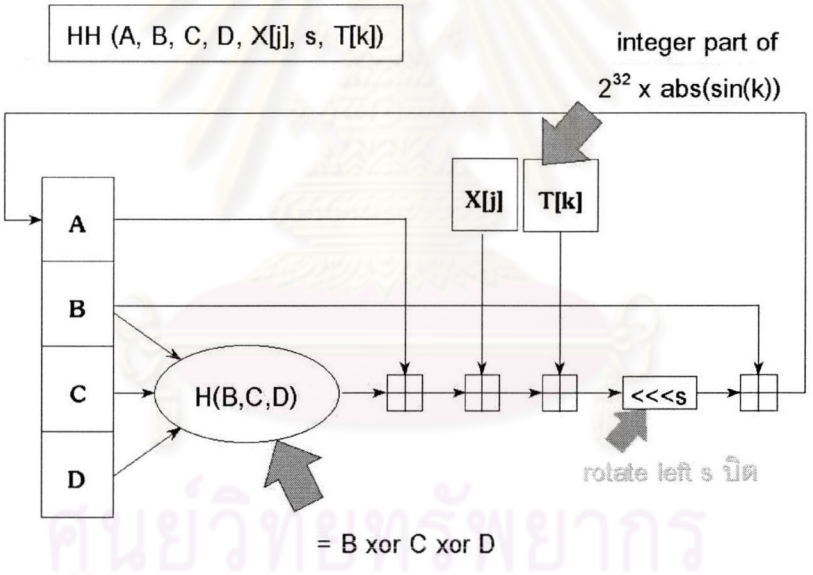
รูปที่ 2.19 การประมวลผลในรอบที่ 4



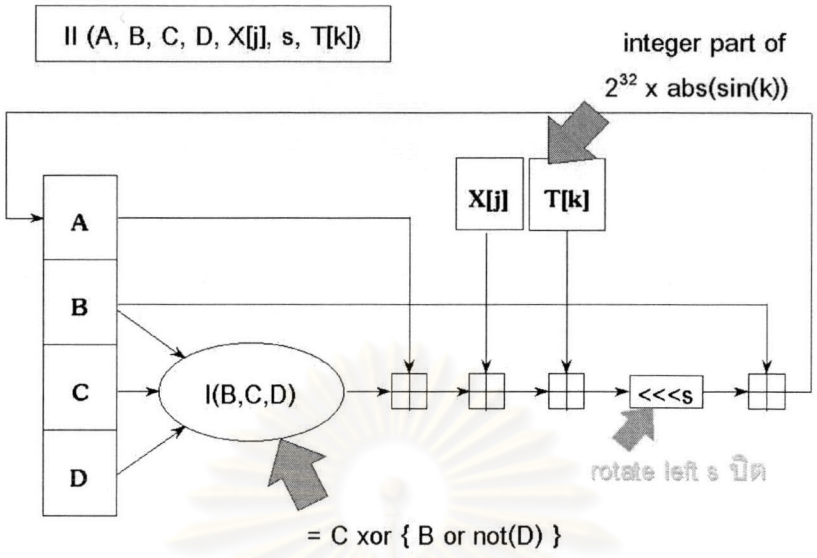
รูปที่ 2.20 การทำงานของฟังก์ชั่น FF ในการประมวลผลรอบที่ 1



รูปที่ 2.21 การทำงานของฟังก์ชั่น GG ในการประมวลผลรอบที่ 2



รูปที่ 2.22 การทำงานของฟังก์ชั่น HH ในการประมวลผลรอบที่ 3



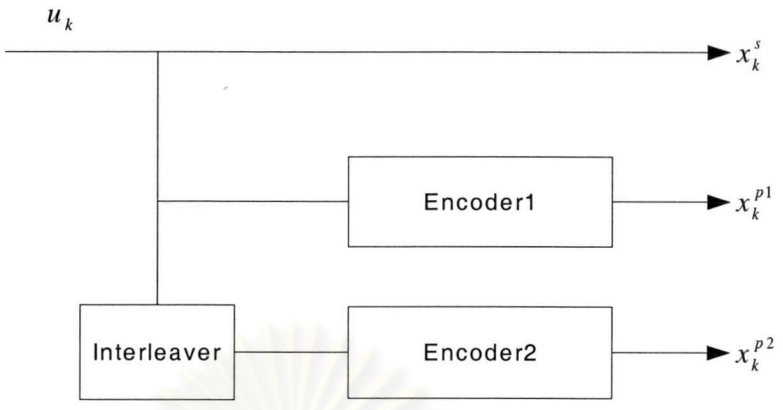
รูปที่ 2.23 การทำงานของฟังก์ชั่น II ในการประมวลผลรอบที่ 4

2.5 รหัสเทอร์โบ

รหัสเทอร์โบ [17-20] เป็นการเข้ารหัสแบบหนึ่งที่ได้รับความนิยมมากสำหรับการเข้ารหัสช่องสัญญาณ (Channel Coding) เนื่องจากการเข้ารหัสนี้ให้อัตราความผิดพลาดของบิตที่ต่ำ หลักการของการเข้ารหัสเทอร์โบนั้นเป็นแบบที่นำเครื่องเข้ารหัสมาต่อขนานกัน (parallel concatenation) ส่วนการถอดรหัสนั้นจะนำเอา Bahl algorithm มาใช้ในการถอดรหัสตัว RSC code โดยที่ผลลัพธ์ที่ออกจากตัวถอดรหัสจะเป็นแบบ soft output ซึ่งจะนำไปใช้ในการป้อนกลับเพื่อให้มีการคำนวณแบบวนซ้ำหลายๆรอบเพื่อแก้ไขและเพิ่มความเชื่อมั่นในการตัดสินใจของระดับสัญญาณ

2.5.1 การเข้ารหัสเทอร์โบ

การเข้ารหัสเทอร์โบนั้นจะนำเอาตัวเข้ารหัสแบบคอนโวลูชันที่มีการป้อนกลับ (RSC:Recursive Systematic Convolutional encoder) 2 ตัวมาต่อกันแบบขนาน ซึ่งโดยปรกติ RSC encoder ทั้งสองจะเหมือนกันแต่ RSC encoder ตัวที่ 2 จะรับเอาบิตข้อมูลที่ผ่านการสลับโดยตัววางสลับ (interleaver)



รูปที่ 2.24 ตัวเข้ารหัสเทอร์โบพื้นฐาน

2.5.2 ขั้นตอนการเข้ารหัส

ตัวอย่างของการเข้ารหัสเทอร์โบถูกแสดงตามรูป บล็อกของตัววางสลับแสดงด้วย α และผลลัพธ์ของตัววางสลับคือ $\overline{u}_i$ ฟังก์ชั่น α จะบอกถึงการวางสลับตามนี้

$$\overline{u}_{\alpha(i)} = u_i \tag{2.12}$$

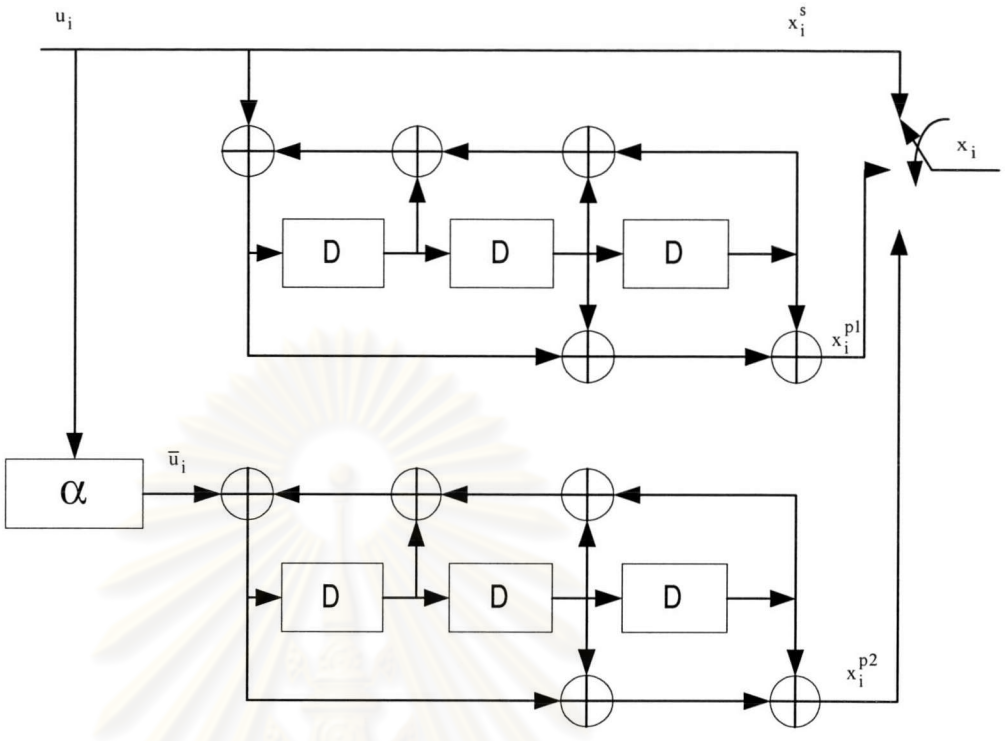
โดยที่ $i \in (0,...,L-1)$ และการวางสลับกลับ (deinterleaving) กำหนดได้ดังนี้

$$u_{\alpha^{-1}(i)} = \overline{u}_i \tag{2.13}$$

systematic output ของตัวเข้ารหัสเทอร์โบ x^s นั้นเอามาจาก RSC encoder ตัวบน และส่วนที่เป็น parity output x^{p1} และ x^{p2} เอามาจาก parity output ของ RSC encoder ตัวที่ 1 และ 2 ตามลำดับ ผลลัพธ์ทั้งสามสายนี้จะถูกมามัลติเพล็กซ์เป็นรหัส

$$x = (x_0^s, x_0^{p1}, x_0^{p2}, ..., x_{L-1}^s, x_{L-1}^{p1}, x_{L-1}^{p2}) \tag{2.14}$$

code rate ของรหัสเทอร์โบ (ไม่คำนึงถึง trellis termination) คือ 1/3 ซึ่งก็เหมือนกับรหัสคอนโวลูชัน โดยสามารถเพิ่มอัตราเข้ารหัสเทอร์โบได้โดยการทำ puncturing



รูปที่ 2.25 ตัวเข้ารหัสเทอร์โบ

2.5.3 การถอดรหัสเทอร์โบ

รหัสเทอร์โบนั้นจะใช้อัลกอริทึมในการถอดรหัสแบบ soft-decision ซึ่งเป็นไปได้เพราะตัวถอดรหัสเทอร์โบสร้างจากตัวถอดรหัสที่มีการวัดความน่าเชื่อถือของการใช้บิตร่วมกัน ซึ่งการที่มีการใช้ข้อมูลร่วมกันนั้นเป็นไปได้จากการวางสลับในตัวเข้ารหัสซึ่งจะสร้างสายของ parity สองสายที่เกี่ยวข้องกันอย่างหลวมๆ

ตัวถอดรหัสนั้นจะมีอัลกอริทึมในการถอดรหัสสองแบบคือ Soft Output Viterbi Algorithm (SOVA) และ Bahl, Cocke, Jelinek และ Rajiv (BCJR) แต่ว่าเนื่องจากต้องการให้อัตราความผิดพลาดต่ำวิธี BCJR จึงนิยมใช้ในรหัสเทอร์โบเพราะว่าใช้กฎการถอดรหัสแบบ MAP (Maximum a posteriori) ขณะที่ SOVA นั้นจะเป็นวิธีประมาณของ MAP และจะให้ผลของอัตราความผิดพลาดแย่กว่าแต่จะมีความซับซ้อนน้อยกว่าเมื่อเทียบกับ BCJR

2.5.4 BCJR Algorithm

พิจารณาตัวเข้ารหัสคอนโวลูชันอย่างมีระบบที่มีอัตราเข้ารหัส 1/2 ที่มีขนาดหน่วยความจำ v โดยมีลำดับบิตข้อมูลความยาว N เพื่อให้ได้รหัสแบบบล็อก (2N,N) ถ้าตัวเข้ารหัสนั้นให้สถานะเริ่มต้นเป็นศูนย์สำหรับการเริ่มต้นของบิตข้อมูลในแต่ละเฟรมรหัสแบบบล็อกก็จะเป็นเชิงเส้น เนื่องจากเป็นตัวเข้ารหัสแบบมีระบบผลลัพธ์ของตัวเข้ารหัสจะประกอบด้วยเฟรมข้อมูลด้านเข้า $u = x^s$ และลำดับของ parity x^P ทั้งบิตข้อมูลและลำดับของ parity จะเป็นเวกเตอร์ขนาด $(1 \times N)$ โดยใช้สัญลักษณ์ไบนารีในการส่ง

ถ้าเป็นการส่งสัญญาณแบบ BPSK บนช่องสัญญาณ AWGN สัญลักษณ์ที่ได้รับจากบิตข้อมูลด้านเข้าและ parity bit จะเป็นไปตามสมการ (2.15) และ (2.16) โดยตัวยกที่ใช้จะใช้แยกระหว่างสัญลักษณ์ด้านเข้า (y_k^s) และสัญลักษณ์ parity (y_k^p) สำหรับช่องสัญญาณ AGWN สัญญาณรบกวนที่เพิ่มเข้าไป (n_k^s และ n_k^p) คือตัวแปรสุ่มแบบเกาส์ที่มีค่าเฉลี่ยเป็นศูนย์และมีความแปรปรวน $N_0/2$ และจะเขียนรหัสที่ได้รับดังนี้ $Y = [y^s | y^p]$

$$y_k^s = \sqrt{E_s}(2 \cdot u_k - 1) + n_k^s \tag{2.15}$$

$$y_k^p = \sqrt{E_s}(2 \cdot x_k^p - 1) + n_k^p \tag{2.16}$$

สำหรับรหัสเดี่ยวๆอัลกอริทึมของ BCJR จะลดอัตราความผิดพลาดของสัญลักษณ์สำหรับการถอดรหัสเทรลิสและรหัสแบบบล็อก โดยหลังจากได้รับ Y จะเป็นหน้าที่ของตัวถอดรหัสที่จะตัดสินว่าบิตข้อมูลด้านเข้า u_k น่าจะเป็นอะไรจากสัญลักษณ์ที่ได้รับ เนื่องจากบิตข้อมูลด้านเข้าอยู่ในรูปของไบนารี (0 และ 1) ดังนั้นจึงสะดวกที่จะใช้ Log-likelihood ratio (LLR) ในการตัดสินบิต โดย LLR สำหรับสัญลักษณ์ด้านเข้าที่เวลา k กำหนดเป็น

$$L(k) = \log\left[\frac{P(u_k = 1|Y)}{P(u_k = 0|Y)}\right] \tag{2.17}$$

จากสมการที่ (2.17) $P(u_k = i|Y)$ คือ posteriori probability ของ $u_k = i$ เมื่อรู้ข้อมูลที่ได้รับ Y ตัวถอดรหัสจะทำการประมาณหา $\hat{u}_k$ ของบิตข้อมูลโดยจะใช้การเปรียบเทียบ $L(k)$ กับค่า threshold สำหรับช่องสัญญาณ AWGN จะทำการตัดสินบิตดังนี้

$$\hat{u}_k = \begin{cases} 1 & if \quad L(k) \geq 0 \\ 0 & if \quad L(k) \geq 0 \end{cases} \tag{2.18}$$

ถ้าตัวเข้ารหัสคอนโวลูชันมีจำนวนหน่วยความจำ v แผนภาพเทรลิสของมันจะมี q สถานะเมื่อ $q=2^v$ โดยสถานะเหล่านี้จะกำหนดตำแหน่งตั้งแต่ 0 ถึง q-1 เมื่อ S_k เป็นสถานะที่เวลา k สำหรับตัวเข้ารหัสแบบคอนโวลูชันแบบมีระบบที่มีอัตราเข้ารหัสเป็น 1/2 การเปลี่ยนสถานะแต่ละครั้งจะมีเอาต์พุต 2 ตัวเกี่ยวข้องโดยที่เอาต์พุตเหล่านี้จะเกี่ยวเนื่องกับบิตข้อมูลขาเข้าและค่า parity ของตัวเข้ารหัส โดยทั่วไป u_k และ x_k^p คือบิตที่เกี่ยวข้องเมื่อมีเปลี่ยนสถานะระหว่าง S_{k-1} ไป S_k จากอัลกอริทึมของ BJCR เขียน log-likelihood ratio ได้ว่า

$$L(k)=\frac{\sum_{m=0}^{(q-1)}\sum_{m'=0}^{(q-1)}\gamma^1(y_k^s,y_k^p,m',m)\cdot\alpha_{k-1}(m')\cdot\beta_k(m)}{\sum_{m=0}^{(q-1)}\sum_{m'=0}^{(q-1)}\gamma^0(y_k^s,y_k^p,m',m)\cdot\alpha_{k-1}(m')\cdot\beta_k(m)} \qquad (2.19)$$

โดยที่

$$\alpha_k(m)=\frac{\sum_{m'=0}^{(q-1)}\sum_{i=0}^{1}\gamma^i(y_k^s,y_k^p,m',m)\cdot\alpha_{k-1}(m')}{\sum_{m=0}^{(q-1)}\sum_{m'=0}^{(q-1)}\sum_{i=0}^{1}\gamma^i(y_k^s,y_k^p,m',m)\cdot\alpha_{k-1}(m')} \qquad (2.20)$$

$$\beta_k(m)=\frac{\sum_{m'=0}^{(q-1)}\sum_{i=0}^{1}\gamma^i(y_k^s,y_k^p,m,m')\cdot\beta_{k+1}(m')}{\sum_{m=0}^{(q-1)}\sum_{m'=0}^{(q-1)}\sum_{i=0}^{1}\gamma^i(y_k^s,y_k^p,m',m)\cdot\alpha_k(m')}$$

$$\gamma^i(r_k,r_k^p,m',m)=p(y_k|u_k=i,S_k=m,S_{k-1}=m')\cdot$$

$$p(y_k^p|u_k=i,S_k=m,S_{k-1}=m')\cdot$$

$$P(u_k=i|S_k=m,S_{k-1}=m')\cdot$$

$$P(S_k=m|S_{k-1}=m') \qquad (2.21)$$

ค่าของ $\alpha_k(m)$ แสดงถึงความเหมือนของสถานะ ที่เวลา k เมื่อคิดจากการกวาดไปข้างหน้าบนแผนภาพเทรลิส โดยจะหาจากการคำนวณแบบซ้ำ และค่าของมันที่เวลา k ใด ๆ คือฟังก์ชันเฉพาะสัญลักษณ์ที่ได้รับ (systematic และ partiy) ที่เวลาน้อยกว่าหรือเท่ากับ k เท่านั้น นอกจากนี้เนื่องจากอยู่ในรูปของการคำนวณแบบซ้ำ ค่าเริ่มต้น $\alpha_0(m)$ จึงขึ้นกับตัวแปรของการเข้ารหัส ถ้าให้ตัวเข้ารหัสเริ่มต้นที่สถานะศูนย์สำหรับแต่ละเฟรมดังนั้น $\alpha_0(0)=1$ และ $\alpha_0(0)=0$ เมื่อ $m\neq 0$

ค่า $\beta_k(m)$ จะแสดงถึงความเหมือนของสถานะโดยจะขึ้นกับการกวาดไปข้างหลังบนแผนภาพเทรลิส มันจะอยู่ในรูปแบบของการคำนวณซ้ำไปข้างหลังโดยเริ่มจากเวลา

$k = N$ โดยจะกำหนดค่าเริ่มต้นของ $\beta_k(m)$ เมื่อรู้สถานะสุดท้ายของตัวเข้ารหัส $S_N = m_N$ ได้ $\beta_N(m_N) = 1$ และ $\beta_N(m) = 0$ เมื่อ $m \neq m_N$

การคำนวณขั้นสุดท้ายของอัลกอริทึม BCJR คือหาค่าความน่าจะเป็นในการเปลี่ยนสถานะ $\gamma^i = (y_k^s, y_k^p, m', m)$ โดยที่ m' คือสถานะของตัวเข้ารหัสที่เวลา $k-1$ และ m คือสถานะที่เวลา k สัญลักษณ์ y_k^s และ y_k^p จะถูกสร้างขึ้นระหว่างการเปลี่ยนสถานะจากสถานะที่ m' ไปสถานะที่ m ต่อไปจะทำ $\gamma^i = (y_k^s, y_k^p, m', m)$ ให้อยู่ในรูปง่าย

ค่า $p(y_k^s | u_k = i, S_k = m, S_{k-1} = m')$ จะถูกลดรูปเป็น $p(y_k^s | u_k)$ เนื่องจาก y_k^s ขึ้นกับค่า u_k และสัญญาณรบกวนในช่องสัญญาณเท่านั้น และ $P(u_k = i | S_k = m, S_{k-1} = m') \cdot P(S_k = m | S_{k-1} = m')$ คือ $P(u_k = i)$ เมื่อการเปลี่ยนจากสถานะที่ m' ไปสถานะที่ m นั้นเป็นได้และเกี่ยวข้องกับ $u_k = i$ ไม่เช่นนั้นค่านี้ก็เป็นศูนย์ จะเขียน $\gamma^i(y_k^s, y_k^p, m', m)$ ใหม่ได้ดังนี้

$$\gamma(y_k^s, y_k^p, m', m) = p(y_k^s | u_k = i) \cdot P(u_k = i) \cdot$$

$$p(y_k^p | u_k = i, S_k = m, S_{k-1} = m') \cdot \delta(m' \xrightarrow{u_k = i} m) \qquad (2.22)$$

$$= p(y_k^s | u_k = i) \cdot P(u_k = i) \cdot \gamma'_i(y_k^p, m', m) \qquad (2.23)$$

โดยที่ $\delta(m' \xrightarrow{u_k = i} m)$ คือฟังก์ชันที่มีค่าเป็น 1 เมื่อ $u_k = i$ ทำให้เกิดการเปลี่ยนจากสถานะที่ m' ไปสถานะที่ m เมื่อแทนค่าและทำให้ $L(k)$ ง่ายขึ้นจะได้

$$L(k) = \log\left[\frac{p(y_k^s | u_k = 1)}{p(y_k^s | u_k = 0)}\right]$$

$$+ \log\left[\frac{(u_k = 1)}{(u_k = 0)}\right]$$

$$+ \log\left[\frac{\sum_{m=0}^{(q-1)} \sum_{m'=0}^{(q-1)} \gamma'_1(y_k^p, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}{\sum_{m=0}^{(q-1)} \sum_{m'=0}^{(q-1)} \gamma'_0(y_k^p, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}\right] \qquad (2.24)$$

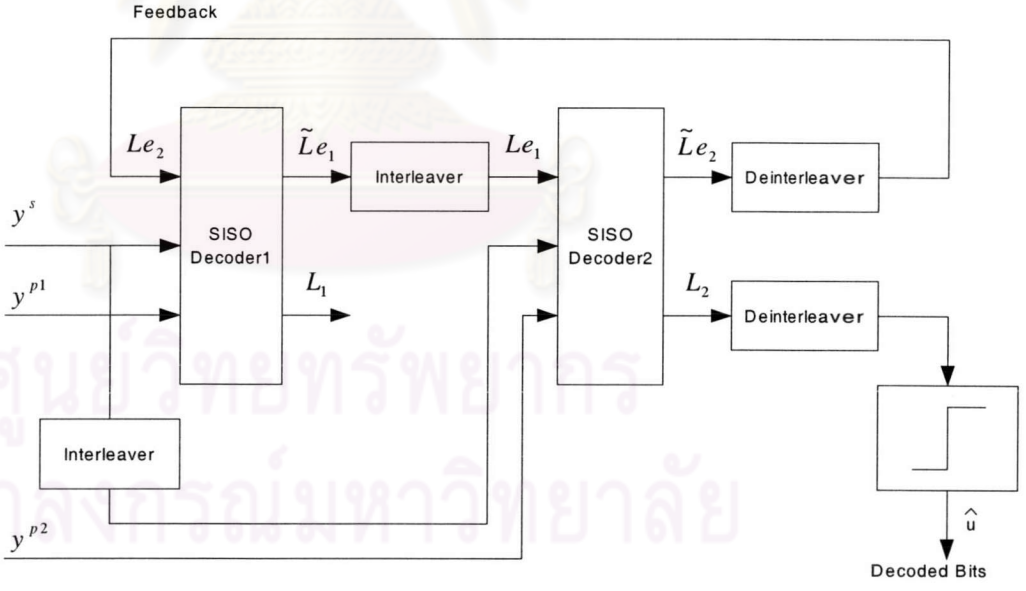
สามารถเขียน Log-likelihood ratio ได้เป็นสามส่วนดังนี้

$$L(k) = L_{systematic} + L_{a\,priori} + L_{extrinsic} \qquad (2.25)$$

$L_{systematic}$ นั้นขึ้นกับเฉพาะส่วนที่ได้รับมาจากสัญลักษณ์ที่เป็น systematic ที่เวลา k $L_{a\,priori}$ นั้นขึ้นกับ a priori information ของบิตข้อมูลเข้า u_k ส่วน $L_{extrinsic}$ คือความรู้ใหม่ที่ได้จากบิตข้อมูลเข้า u_k โดยจะขึ้นกับ parity และ systematic information ทั้งหมดยกเว้น systematic value ที่เวลา k โดยการใช้สมการนี้ของ log-likelihood ratio ทำให้เราสามารถถอดรหัสเทอร์โบได้

2.5.5 ขั้นตอนการถอดรหัส

จากข้างต้นเราจะใช้ตัวถอดรหัสที่ใช้ MAP และได้เห็นแล้วว่า log-likelihood ratio นั้นสามารถแยกออกมาได้เป็น 3 ส่วนจากสมการที่ (2.25) จากรูปที่ 2.26 **SISO Decoder 1** และ **SISO Decoder 2** คือตัวถอดรหัสที่ใช้อัลกอริทึมของ BCJR สำหรับการถอดรหัสที่เข้ารหัสด้วยตัวเข้ารหัสเทอร์โบ 2 ตัวแยกกัน L_1 และ L_2 คือค่า log-likelihood ratio ที่คำนวณได้โดยใช้อัลกอริทึม BJCR ตามสมการที่ (2.24) อย่างไรก็ดีกุญแจสำคัญในการคำนวณซ้ำไปมาของเครื่องถอดรหัสเทอร์โบก็คือการส่งความน่าจะเป็นของบิตข้อมูลซึ่งอยู่ในรูปของ extrinsic information ที่ได้จาก BJCR



รูปที่ 2.26 ตัวถอดรหัสเทอร์โบ

extrinsic information นี้จะสร้างโดยตัวถอดรหัสอีกตัวเพื่อใช้ในการคำนวณหา a priori probability ที่แต่ละบิต ตัวอย่างเช่น **Decoder 1** จะใช้ extrinsic information จาก **Decoder 2** เพื่อใช้คำนวณ a priori probability $P(u_k = i)$ สำหรับแต่ละบิต

$$P(u_k = 1) = \left(\frac{e^{L_{e2}(k)}}{1 + e^{L_{e2}(k)}} \right) \tag{2.26}$$

$$P(u_k = 0) = 1 - \left(\frac{e^{L_{e2}(k)}}{1 + e^{L_{e2}(k)}} \right) \tag{2.27}$$

โดยค่า a priori information ใหม่นี้จะถูกใช้ในอัลกอริทึม BJCR สำหรับการกำหนดค่าเริ่มต้นนั้นจะให้ $L_{e_2}(k) = 0$ ทุกๆค่า k เมื่อเริ่มทำการคำนวณครั้งแรก จุดมุ่งหมายในการคำนวณแบบวนซ้ำไปมาก็คือเพื่อกำจัดความผิดพลาดของบิต

2.6 งานวิจัย Digital Watermarking ที่ผ่านมา

คำว่า Digital Image Watermarking ได้เกิดขึ้นเป็นครั้งแรกเมื่อปี 1993 เมื่อ Tirkel [11] ได้เสนอเทคนิคขึ้นมา 2 แบบในการฝังข้อมูลลงในภาพ ซึ่งวิธีเหล่านี้จะอาศัยการเปลี่ยนแปลง LSB ของจุดภาพ การฝังลายน้ำใน spatial domain ได้เริ่มมีการพัฒนามาก่อน แต่เนื่องจากยังไม่ค่อยมีความทนทานมากนักจึงได้มีการพัฒนามาฝังข้อมูลใน frequency domain Frequency domain watermarking ได้เริ่มนำมาใช้โดย Cox[5] วิธีของ Cox จะใช้หลักการของสเปรดสเปกตรัมเพื่อที่จะฝังข้อมูลเพียงบิตเดียวในภาพ (ตัดสินว่ามีหรือไม่มีลายน้ำที่กำหนดไว้เท่านั้น) และต้องใช้ภาพต้นฉบับในขั้นตอนการตรวจจับลายน้ำ

งานวิจัยในการฝังลายน้ำส่วนมากจะใช้หลักการของสเปรดสเปกตรัมซึ่งคล้ายกับ spread sprectrum communication โดยที่สัญญาณ narrow band จะถูกส่งเป็นสัญญาณที่มีแบนด์วิดท์ขนาดใหญ่มากทำให้พลังงานของสัญญาณในแต่ละความถี่มีค่าน้อยมาก เช่นเดียวกันกับที่ลายน้ำจะถูก spread ไปในหลายความถี่ทำให้พลังงานในแต่ละความถี่มีค่าน้อยและทำให้ตำแหน่งในการฝังไม่เป็นที่สังเกตได้ สัญญาณลายน้ำที่จะถูกฝังลงไปจะเป็นเลขสุ่มเทียมมีพลังงานต่ำและจะถูกตรวจจับโดยอาศัยการหาค่าสหสัมพันธ์ของสัญญาณลายน้ำที่ตรวจจับได้กับสัญญาณลายน้ำที่ฝัง ถ้าค่าสหสัมพันธ์มีค่ามากกว่าค่าจุดเริ่มเปลี่ยน (Threshold) ที่กำหนดแสดงว่าสามารถถอดออกมาได้อย่างถูกต้อง

2.6.1 การฝังลายน้ำโดยวิธีของ Cox [5]

วิธีของ Cox เป็นประเภท Private watermarking Type I เนื่องจากต้องใช้รูปต้นฉบับในการตรวจสอบและบอกเพียงว่ารูปนี้ฝังลายน้ำหรือไม่เท่านั้น โดย Cox จะอาศัยหลักการของ Spread spectrum ในการใส่ลายน้ำโดยวิธีดังนี้

- การฝังลายน้ำ

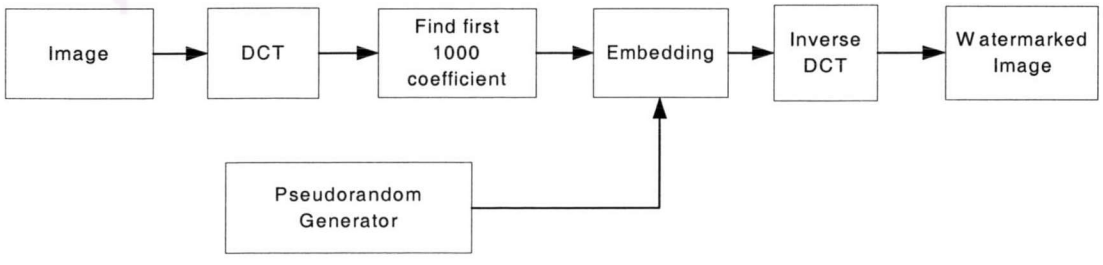
ลายน้ำ $X = x_1,...,x_n$ เป็นลำดับจำนวนสุ่มเทียมที่มีการแจกแจงปกติมีค่าเฉลี่ยเป็นศูนย์ ค่าความแปรปรวนเป็นหนึ่ง $N(0,1)$ ลายน้ำถูกฝังลงไปในรูปภาพ V ในส่วนที่มีความสำคัญต่อการมองเห็นมากที่สุด (most perceptive modes of the image) หรือก็คือค่าสัมประสิทธิ์การแปลงโคซายน์กลุ่มที่มีค่ามากที่สุดเพื่อให้มีความทนทานต่อกระบวนการบีบอัดแบบสูญเสีย (lossy compression) และวิธีการประมวลผลภาพแบบต่างๆ วิธีซึ่งเสนอโดย Cox ใส่ลายน้ำโดยทำการแปลงโคซายน์กับรูปต้นฉบับทั้งรูป แล้วเปลี่ยนแปลงค่าสัมประสิทธิ์ซึ่งมีค่ามากที่สุด 1000 ตัวแรก (ไม่รวม DC coefficient) ดังนี้

$$v'_i = v_i + \alpha x_i \tag{2.28}$$

$$v'_i = v_i(1 + \alpha x_i) \tag{2.29}$$

$$v'_i = v_i e^{\alpha x_i} \tag{2.30}$$

โดยที่ α คือความแรงในการฝังลายน้ำซึ่งสามารถปรับค่าได้เพื่อให้ได้ความทนทานและการมองไม่เห็นที่พอเหมาะ ถ้า α มีค่ามากลายน้ำจะมีความทนทานสูงแต่ในขณะเดียวกันก็จะทำให้เกิดการมองเห็นมากไปด้วยและ v_i คือส่วนประกอบของสเปกตรัมที่ผ่าน DCT ซึ่งโดยทั่วไปจะนิยมใช้สมการที่ (2.29) จากนั้นทำการแปลงกลับโคซายน์ก็จะได้รูปที่ฝังลายน้ำแล้ว V'

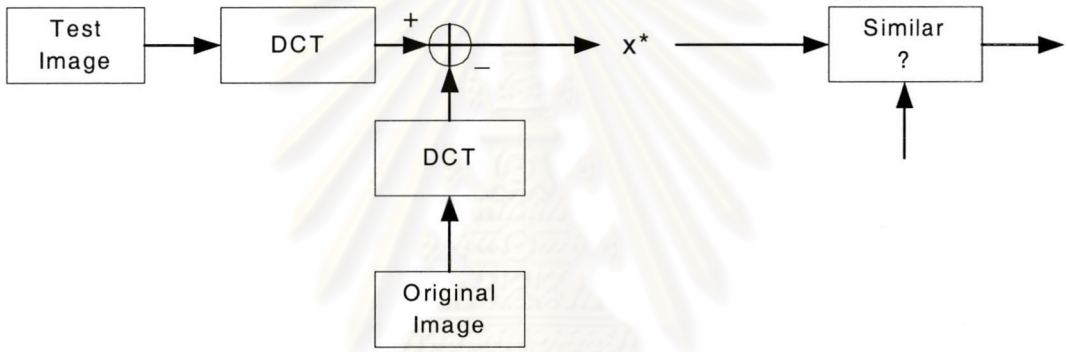


รูปที่ 2.27 การฝังลายน้ำโดยวิธีของ Cox

❑ การตรวจจับลายน้ำ

การตรวจจับจะต้องใช้รูปต้นฉบับ โดยนำมาลบออกจากรูปทดสอบแล้วทำการแปลงโคซายน์กับผลต่างนั้นซึ่งพิจารณาสมการจะได้ค่าผลต่างคือ $\alpha v_i x_i$ เมื่อหารด้วย $\alpha_i v_i$ (เนื่องจากมีรูปต้นฉบับ) ก็จะสามารถหาค่าประมาณของลายน้ำที่ตรวจจับได้ในรูปทดสอบคือ $X*$ ซึ่งนำมาหาค่าดัชนีความเหมือนกัน (similarity index) กับลายน้ำที่ฝังลงไปได้ดังนี้

$$sim(X, X*) = \frac{X * X}{\sqrt{X * X *}} \tag{2.31}$$



รูปที่ 2.28 การตรวจจับลายน้ำโดยวิธีของ Cox

2.6.2 การฝังลายน้ำโดยวิธีของ Piva [6]

วิธีของ Piva เป็นประเภท Semi-private watermarking เนื่องจากไม่ต้องใช้รูปต้นฉบับในการตรวจสอบและบอกเพียงว่ารูปนี้ฝังลายน้ำหรือไม่เท่านั้น โดยจะอาศัยหลักการของสเปรดสเปกตรัมในการใส่ลายน้ำเหมือน Cox แต่จะดัดแปลงดังนี้

❑ ลักษณะของลายน้ำ

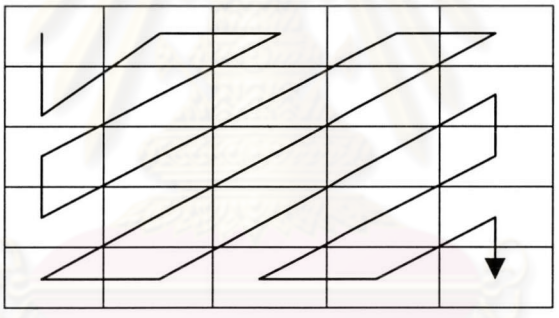
ลายน้ำ $X = \{x_1, x_2, ..., x_M\}$ เป็นลำดับจำนวนสุ่มเทียม (pseudorandom number sequence) ความยาว M สร้างโดย multiplicative congruential algorithm แต่ละค่า x_i เป็นเลขจำนวนจริงแบบสุ่ม (random real number) ที่มีการแจกแจงปกติ (normal distribution) มีค่าเฉลี่ยเป็นศูนย์และมีค่าความแปรปรวนเป็นหนึ่ง

❑ การฝังลายน้ำ

จากรูปต้นฉบับ gray scale image I ขนาด N x N นำมาทำการแปลงโคซายน์ทั้งรูปแล้วเรียงสัมประสิทธิ์ที่ได้แบบซิกแซ็ก ดังแสดงในรูปที่ 2.29 ฝังลายน้ำโดยการเปลี่ยนแปลงค่าสัมประสิทธิ์ตัวที่ L+1 ถึง L+M ซึ่งก็คือสัมประสิทธิ์ในกลุ่มแรก ๆ ที่มีความถี่ต่ำ เพื่อให้ลายน้ำทนทานต่อกระบวนการต่าง ๆ แต่ข้าม L ตัวแรกไปเพื่อไม่ให้เกิดการมองเห็นอย่างชัดเจน เวคเตอร์ T' = { t'_{L+1}, t'_{L+2}, ..., t'_{L+M} } ของสัมประสิทธิ์ที่ถูกเปลี่ยนแปลงด้วยลายน้ำคำนวณได้ตามสมการ (2.32) เวคเตอร์ T' นี้จะถูกใส่กลับเข้าไปในลำดับแบบซิกแซ็ก จากนั้นทำการแปลงกลับโคซายน์ก็จะได้รูปที่ถูกฝังด้วยลายน้ำ (Watermarked Image) I'

$$t'_{L+i} = t_{L+i} + \alpha |t_{L+i}| x_i \qquad (2.32)$$

i = 1, 2, ..., M α คือ ความเข้มของลายน้ำ



รูปที่ 2.29 การจัดเรียงสัมประสิทธิ์แบบซิกแซก

❑ การตรวจจับลายน้ำ

รูปทดสอบ I^* ขนาด N x N ทำการแปลงโคซายน์ทั้งรูปแล้วจัดเรียงสัมประสิทธิ์แบบซิกแซ็ก เวคเตอร์ $T^* = \{t^*_{L+1}, t_{L+2}*, ..., t^*_{L+M}\}$ เป็นเวคเตอร์ของสัมประสิทธิ์ตัวที่ $L+1$ ถึง $L+M$ ของรูปทดสอบ หาค่าสหสัมพันธ์ (correlation) z ระหว่างสัมประสิทธิ์ DCT ของรูปทดสอบกับลายน้ำที่จะนำมาตรวจสอบ Y ได้ตามสมการ (2.33) โดยค่า z จะใช้ตัดสินว่าในรูป I^* มีลายน้ำ Y อยู่หรือไม่โดยเปรียบเทียบกับค่าจุดเริ่มเปลี่ยน (threshold) T_z

$$z = \frac{Y.T^*}{M} = \frac{1}{M}\sum_{i=1}^{M} y_i t^*_{L+i} \tag{2.33}$$

- การหาค่าจุดเริ่มเปลี่ยน T_z

กรณีที่รูปทดสอบ I^* เหมือนกับรูปที่ถูกฝังด้วยลายน้ำ I' และเมื่อละจำนวนสัมประสิทธิ์ที่ถูกข้าม L จะได้ว่า

$$t^*_i = t'_i = t_i + \alpha|t_i|x_i \tag{2.34}$$

$$z = \frac{1}{M}\sum_{i=1}^{M}\left(t_i y_i + \alpha|t_i|x_i y_i\right) \tag{2.35}$$

สามารถหาค่าเฉลี่ยและค่าความแปรปรวนของ z ภายใต้สมมติฐานที่ว่า t_i และ x_i มีค่าเฉลี่ยเป็นศูนย์และเป็นอิสระต่อกัน ได้ดังนี้

$$\mu_z = \begin{cases} \alpha\mu_{|t|} & \text{if X = Y} \\ 0 & \text{if X} \neq \text{Y} \\ 0 & \text{if no mark is present} \end{cases} \tag{2.36}$$

$$\sigma_z^2 \approx \frac{\sigma_t^2}{M} \tag{2.37}$$

กรณีลายน้ำที่นำมาตรวจสอบไม่ตรงกับลายน้ำในรูปทดสอบ : z_1 $\mu_{z_1} = 0$

กรณีลายน้ำที่นำมาตรวจสอบตรงกับลายน้ำในรูปทดสอบ : z_2 $\mu_{z_2} = \alpha\mu_{|t|}$

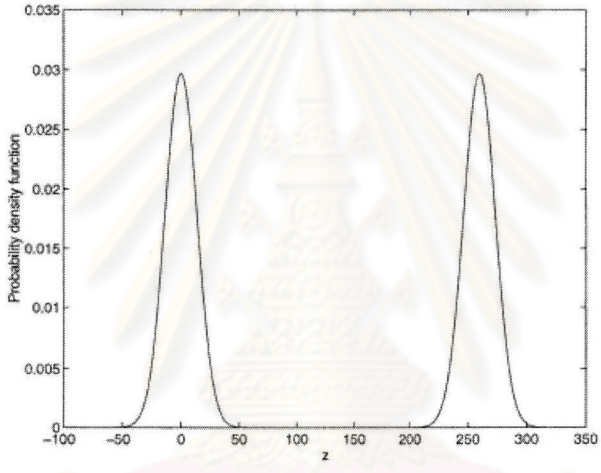
โดย $\sigma_{z_1}^2 = \sigma_{z_2}^2 \approx \frac{\sigma_t^2}{M}$ (2.38)

เขียนการแจกแจงของตัวแปรสุ่ม z_1 และ z_1 ได้ดังรูป 2.30 เพื่อให้ความน่าจะเป็นของการตรวจวัดความผิดพลาด (error probability) มีค่าต่ำระยะทางระหว่างการกระจายของตัวแปรทั้งสองหรือค่า $\frac{\mu_z}{\sigma_z} = \frac{\alpha\mu_{|t|}}{\frac{\sigma_t^2}{M}} = M\alpha\frac{\mu_{|t|}}{\sigma_t^2}$ ควรมีค่ามากและเนื่องจากเมื่อจำนวนสัมประสิทธิ์ที่ถูกข้าม L เพิ่มขึ้น $\mu_{|t|}$ และ $\sigma^2{}_t$ จะลดลงแต่ $\sigma^2{}_t$ จะลดลงด้วยอัตราที่เร็วกว่าดังนั้นจึงควรใช้ค่า L มากๆและจากสมการจะเห็นว่าค่า M ก็ควรมีค่ามากเช่นกัน สามารถหาค่าจุดเริ่มเปลี่ยน T_z ได้ดังนี้

$$T_z = \frac{\mu_{z_2}}{2} = \frac{\alpha\mu_{|t|}}{2} = \frac{\alpha}{2M}\sum_{i=1}^{M}|t'_i| \qquad (2.39)$$

แต่จากผลการทดสอบพบว่าถ้ารูปภาพถูกรบกวนด้วยวิธีการที่จงใจหรือไม่ก็ตาม จะทำให้ σ_{z_2} เพิ่มขึ้นอย่างมาก นั่นคือ z_1 และ z_2 ยังมีการกระจายแบบเดิมแต่ z_2 จะมีค่าความแปรปรวนสูงขึ้น ดังนั้นค่าจุดเริ่มเปลี่ยนที่ใช้จึงควรเข้าใกล้ศูนย์มากกว่าที่จะอยู่ตรงกลาง Piva ได้เสนอให้ใช้ค่าจุดเริ่มเปลี่ยนเป็นตามสมการ (2.40) ซึ่งให้ผลการทดสอบเป็นที่น่าพอใจ

$$T_z = \frac{\alpha}{3M}\sum_{i=1}^{M}|t^*{}_i| \qquad (2.40)$$



รูปที่ 2.30 การแจกแจงของตัวแปรสุ่ม z_1 และ z_2

- Visual Masking

หลังจากที่ได้รูปที่ถูกฝังด้วยลายน้ำ I' แล้วสามารถปรับให้ลายน้ำเข้ากับรูปที่ถูกฝังมากขึ้นเพื่อช่วยเพิ่มการมองไม่เห็น (invisibility) ได้โดยใช้ spatial masking characteristics ของ Human Visual System (HVS) โดยในขั้นตอนของการใส่ลายน้ำ รูปต้นฉบับ I และรูปที่ถูกฝังด้วยลายน้ำ I' จะถูกนำมาบวกกันทีละจุดภาพดังนี้

$$y''_{ij} = y_{ij}(1-\beta_{ij}) + \beta_{ij}y'_{ij} = y_{ij} + \beta_{ij}(y'_{ij} - y_{ij}) \qquad (2.41)$$

การหาค่า β_{ij} สำหรับจุดภาพ y_{ij} พิจารณาบล็อกขนาด R x R หาค่าความแปรปรวนของจุดภาพนั้นแล้วทำการนอร์มอลไลซ์ (normalized) ด้วยความแปรปรวนที่มากที่สุด β_{ij} คือค่าความแปรปรวนนอร์มอลไลซ์ (normalized variance) ของจุดภาพ y_{ij} บริเวณที่ทนต่อ

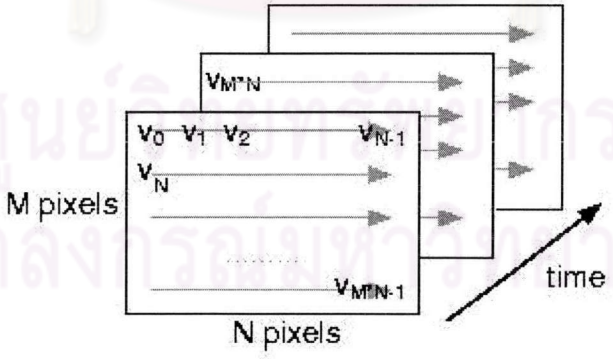
สัญญาณรบกวนได้มาก (บริเวณที่มีรายละเอียดสูง) $\beta_{ij} \approx 1$ ดังนั้น $y''_{ij} \approx y'_{ij}$ นั่นคือสามารถใส่ลายน้ำได้เต็มที่ ส่วนบริเวณที่ทนต่อสัญญาณรบกวนได้น้อย $\beta_{ij} \approx 0$ จะได้ $y''_{ij} \approx y_{ij}$ สามารถใส่ลายน้ำได้เพียงเล็กน้อยเท่านั้น จะเห็นได้ว่าเมื่อใช้วิธีนี้จะช่วยให้สามารถเพิ่มความเข้มของลายน้ำ α ได้โดยไม่ทำให้เกิดการมองเห็นมาก และทำให้ความพยายามที่จะเอาลายน้ำออกทำได้ยากขึ้นด้วย

2.6.3 การฝังลายน้ำโดยวิธีของ Hartung และ Girod [8]

งานวิจัยนี้ศึกษาการใส่ลายน้ำลงใน uncompressed และ compressed video โดยอาศัยหลักการ spread spectrum ในการส่งสัญญาณ narrow-band (watermark) ไปในสัญญาณที่มีแบนด์วิดท์กว้างโดยมีสัญญาณรบกวนคือรูปต้นฉบับหรือสัญญาณวิดีโอ ในที่นี้ขอกล่าวถึงเฉพาะการใส่ลายน้ำใน uncompressed video ซึ่งสามารถนำมาปรับใช้กับการใส่ลายน้ำในรูปภาพได้

- การใส่ลายน้ำ

สัญญาณวิดีโอเป็นสัญญาณที่มี 3 มิติ คือ 2 มิติทาง space และ 1 มิติทางเวลา พิจารณาสัญญาณวิดีโอจากการทำ line-scanning ดังในรูปที่ 2.31 ลายน้ำที่จะฝังเป็นลำดับของบิต a_j, $a_j \in \{-1,1\}$ $j \in N$ จากนั้นจะนำไป spread คือเรียงซ้ำไปจำนวนครั้งเท่ากับ cr (chip rate) ตามสมการที่ (2.42) ลายน้ำหนึ่งบิตจะถูกใส่เข้าไปในจุดภาพ cr จุดของสัญญาณวิดีโอ



รูปที่ 2.31 line scan ของสัญญาณวิดีโอ

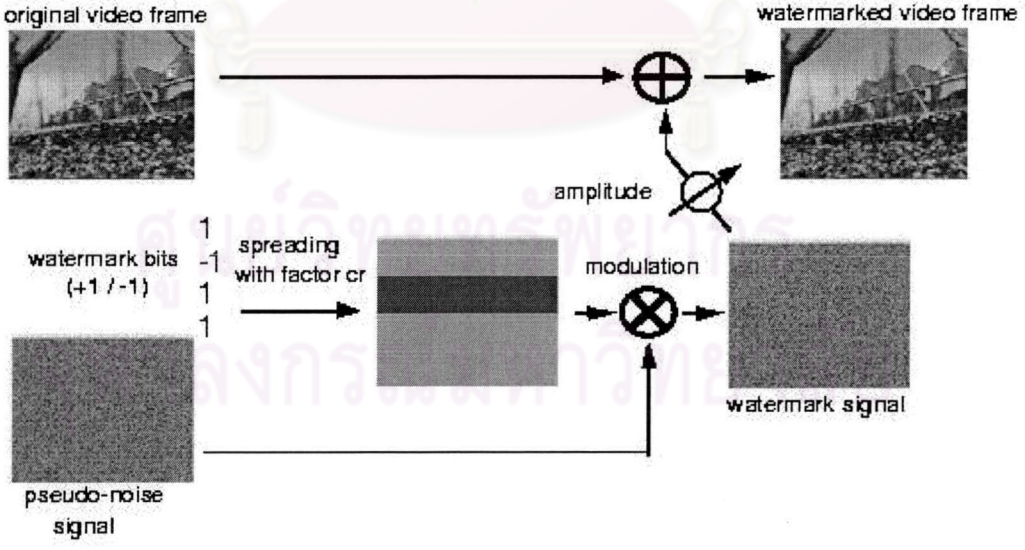
$$b_i = a_j, \qquad j.cr \le i < (j+1)cr \qquad i \in N \tag{2.42}$$

ลำดับของบิตในสมการ (2.42) จะถูกมอดูเลตด้วยสัญญาณไบนารี pseudo-noise sequence p_i, $p_i \in \{-1,1\}$ $i \in N$ เพื่อการ spread ทางความถี่ (frequency spreading) ซึ่งถูกสร้างได้หลายวิธี เช่น อาจสร้างจากตัวกำเนิดลำดับจำนวนสุ่มเทียม (pseudorandom number generator) และไม่เป็นสัญญาณไบนารีก็ได้ ในที่นี้ใช้สัญญาณไบนารีเพื่อความสะดวก pseudo-noise sequence นี้จะเปรียบเสมือนเป็นกุญแจ (key) ของการฝังและตรวจจับลายน้ำ และเนื่องจากแต่ละ sequence จะตั้งฉาก (orthogonal) กัน ดังนั้นจึงสามารถใส่ลายน้ำได้มากกว่าหนึ่งแบบในต้นฉบับเดียวโดยสามารถทำการตรวจจับได้อย่างเป็นอิสระต่อกัน

ลายน้ำ w หาได้ตามสมการ (2.43) โดย α คือความเข้มของลายน้ำ สามารถปรับได้เพื่อปรับระดับการมองเห็น (visibility) ของลายน้ำ ลายน้ำจะถูกฝังลงในสัญญาณวิดีโอโดยการเรียง w ให้เป็นเมตริกซ์แล้วบวกเข้ากับต้นฉบับใน spatial domain ตามสมการ (2.44)

$$w_i = \alpha_i . b_i . p_i \qquad i \in N \tag{2.43}$$

$$\tilde{v}_i = v_i + \alpha_i . b_i . p_i \qquad i \in N \tag{2.44}$$



รูปที่ 2.32 การฝังลายน้ำโดยวิธีของ Hartung และ Girod

- การตรวจจับลายน้ำ

วิธีของ Hartung และ Girod สามารถดึงบิตข้อมูลที่ฝังอยู่จริงออกมาได้โดยไม่ต้องใช้ต้นฉบับ สัญญาณวิดีโอ $\tilde{v}$ จะถูกนำไปผ่าน highpass filter ได้ $\overline{\tilde{v}}$ เพื่อกำจัดส่วนประกอบหลักคือส่วนต้นฉบับออก แล้วดีมอดูเลตโดยการคูณสัญญาณวิดีโอที่ผ่าน filter $\overline{\tilde{v}}$ ด้วยสัญญาณ pseudo-noise p_i เดิมที่ใช้ในการฝังลายน้ำ หาผลรวมในแต่ละช่วงการทำซ้ำของแต่ละบิต จะได้ผลบวกของค่าสหสัมพันธ์ (correlation sum) s_j สำหรับบิตที่ j

$$s_j = \sum_{i=j.cr}^{(j+1).cr-1} p_i.\overline{\tilde{v}}_i = \underbrace{\sum_{i=j.cr}^{(j+1).cr-1} p_i.\bar{v}_i}_{\Sigma_1} + \underbrace{\sum_{i=j.cr}^{(j+1).cr-1} \overline{p_i.\alpha_i.b_i}}_{\Sigma_2} \qquad (2.45)$$

ส่วน Σ_1 และ Σ_2 คือผลบวกของค่าสหสัมพันธ์สำหรับส่วนของสัญญาณวิดีโอที่ผ่าน filter และส่วนของลายน้ำที่ผ่าน filter ตามลำดับ หากส่วนของสัญญาณวิดีโอถูกกรองออกไปโดย filter จนหมด จะได้ว่า Σ_1 เป็นศูนย์ และหาก highpass filter ไม่มีผลกับลายน้ำส่วนที่เป็นสัญญาณ pseudo-noise นั่นคือ $\overline{p_i.\alpha_i.b_i} \approx p_i.\alpha_i.b_i$ จะเขียนสมการ (2.45) ได้เป็น

$$s_j = \Sigma_1 + \Sigma_2 \approx \sum_{i=j.cr}^{(j+1).cr-1} p_i^2.\alpha_i.b_i = a_j.\sigma_p^2.cr.mean(\alpha_i) \qquad (2.46)$$

โดยที่ σ_p^2 คือค่าความแปรปรวนของสัญญาณ pseudo-noise พบว่าเครื่องหมายของผลบวกของค่าสหสัมพันธ์คือบิตข้อมูลที่ต้องการนั่นเอง ดังแสดงในสมการ (2.47) หากผลบวกของค่าสหสัมพันธ์ระหว่างสัญญาณวิดีโอที่ต้องการทดสอบและผ่าน highpass filter แล้วกับสัญญาณ pseudo-noise เป็นบวก บิตข้อมูลนั้นคือ +1 และในทางกลับกันถ้าเป็นลบ บิตข้อมูลนั้นคือ –1 การตรวจจับลายน้ำให้ถูกต้องจำเป็นต้องรู้สัญญาณ pseudo-noise p_i ที่ใช้ในการฝังลายน้ำ และนอกจากนั้นยังต้องรู้ seed ที่ใช้สร้างสัญญาณนั้นด้วย

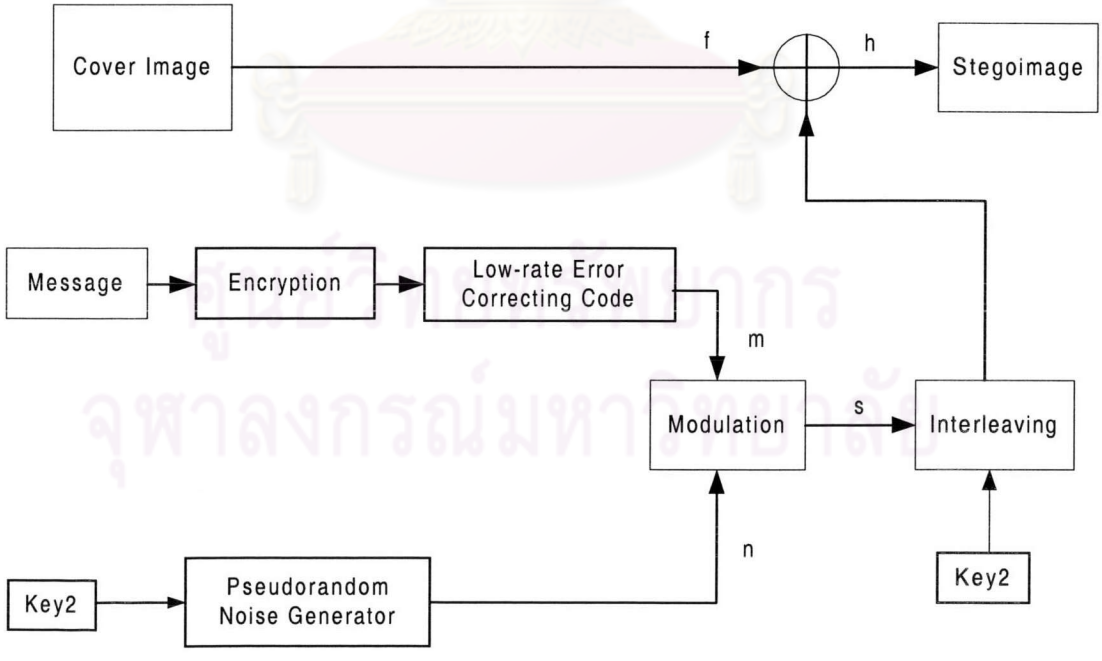
$$sign(s_j) = sign\left(a_j \cdot \underbrace{\sigma_p^2 \cdot cr \cdot mean(\alpha_i)}_{>0} \right) = sign(a_j) = a_j \qquad (2.47)$$

2.6.4 **การฝังลายน้ำโดยวิธีของ** Lisa [9]

วิธีของ Lisa เป็นประเภท Public watermarking เนื่องจากไม่ต้องใช้รูปต้นฉบับในการตรวจสอบและเป็นการฝังบิตข้อมูลลงไปจริงๆโดย Lisa จะอาศัยหลักการของสเปรดสเปกตรัมในการใส่ลายน้ำเหมือน Cox แต่จะทำใน Spatial Domain และใช้รหัสแก้ไขความผิดพลาดร่วมด้วย

- การซ่อนข้อมูล

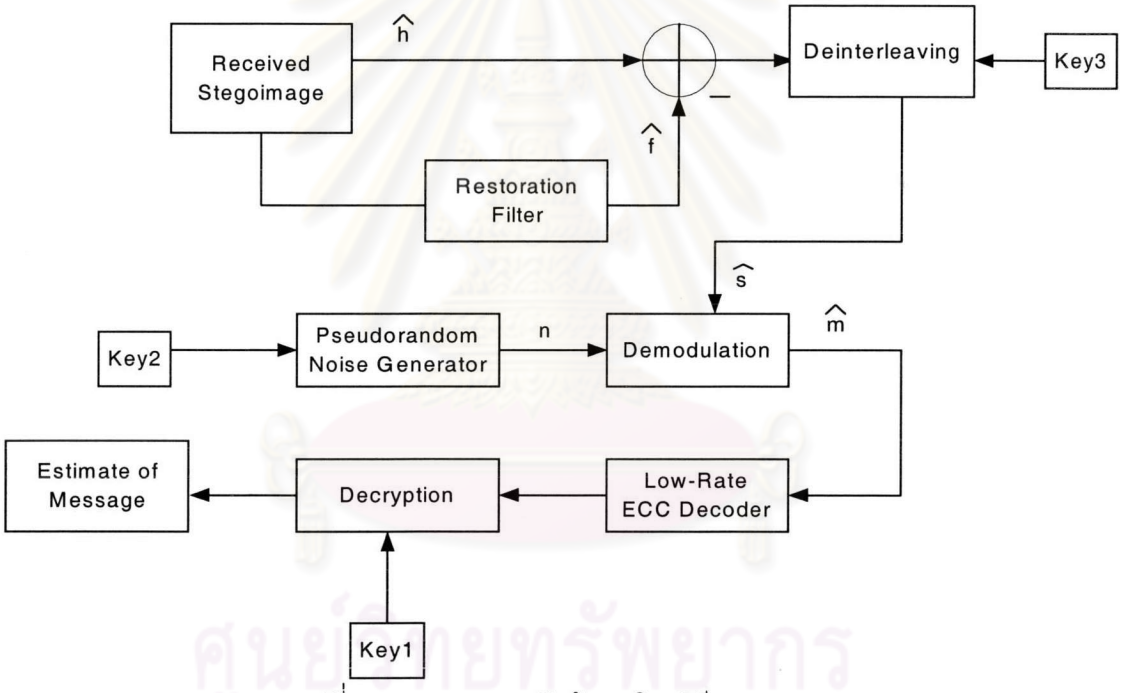
ในที่นี้ Lisa Marvel เรียกวิธีที่เสนอว่าการซ่อนข้อมูล (Data Hiding) เนื่องจากว่าคำนึงถึงเฉพาะปริมาณข้อมูลที่จะฝังลงไปให้ได้มากที่สุดโดยไม่คำนึงถึงความทนทาน ข้อมูลที่จะใส่ซึ่งจะเป็นบิตศูนย์หรือหนึ่งเรียงกันจำนวนหนึ่งจะถูกเข้ารหัสลับด้วยกุญแจตัวที่ 1 หลังจากนั้นจะนำมาเข้ารหัสด้วยรหัสแก้ไขความผิดพลาดก็จะได้ข้อมูลที่เข้ารหัสแล้ว m ในขณะเดียวกันก็จะทำการสร้างลำดับจำนวนสุ่มเทียม (pseudorandom number sequence) ได้ spread sequence n โดยใช้กุญแจตัวที่ 2 เสร็จแล้วนำ m และ n มามอดูเลตกันจากนั้นนำเอาค่าที่ผ่านการมอดูเลต s มาทำการวางสลับโดยใช้กุญแจตัวที่ 3 สัญญาณที่ได้นั้นก็จะนำมาบวกกับรูปต้นฉบับ f ก็จะได้ภาพที่ผ่านการใส่ลายน้ำมา h



รูปที่ 2.33 การซ่อนข้อมูลโดยวิธีของ Lisa

❑ การตรวจจับข้อมูล

การดีเทกต์ข้อมูลในที่นี้จะไม่ใช้ภาพต้นฉบับการถอดรหัสขั้นแรกก็จะต้องใช้เทคนิคของ image restoration มาทำการประมาณภาพต้นฉบับจากภาพที่ทำการฝังข้อมูลที่ได้รับมาซึ่งอาจผ่านการเปลี่ยนแปลงมาแล้ว ซึ่งก็ทำได้เช่นใช้ Adaptive Wiener Filter ความต่างของทั้งสองนี้จะนำมาทำวางสลับกลับ (deinterleaving) โดยใช้กุญแจตัวที่สามเพื่อหาค่าประมาณของสัญญาณที่ใส่เข้าไป จากนั้นจะทำการสร้างลำดับจำนวนสุ่มเทียม (pseudorandom number sequence) โดยใช้กุญแจตัวที่ 2 นำสัญญาณทั้งสองมาดีมอดูเลตก็จะได้สัญญาณประมาณของข่าวสาร นำรหัสแก้ความผิดพลาดมาถอดรหัสข่าวสารนี้เสร็จแล้วนำไปถอดรหัสลับโดยใช้กุญแจตัวที่ 1 ก็จะได้ข้อมูลที่ใส่เข้าไป



รูปที่ 2.34 การตรวจจับข้อมูลโดยวิธีของ Lisa

Image Restoration Filter ที่จะนำมาใช้ประกอบในขั้นตอนการถอดลายน้ำคือ Adaptive Wiener Filter (AW filter) โดยจะใช้ในการลดปริมาณของสัญญาณรบกวนที่บวกเข้าไปในภาพต้นฉบับ AW filter จะทำการรักษาสัญญาณไว้ขณะที่ทำการกำจัดสัญญาณรบกวนในภาพที่เสีย เนื่องจากภาพต้นฉบับและสัญญาณที่ใส่เข้าไปเป็นอิสระเชิงเส้นกัน ดังนั้นจะได้ความผิดพลาดกำลังสองเฉลี่ย (MSE) ต่ำสุดจากการฟิลเตอร์ภาพโดยใช้ AW filter ผลตอบสนองเชิงความถี่ของฟิลเตอร์นั้นขึ้นกับสเปกตรัมกำลังของภาพต้นฉบับและสัญญาณรบกวนตามสมการที่

(2.48) โดยที่ P_f คือสเปกตรัมกำลังของภาพต้นฉบับ f และ P_s คือสเปกตรัมกำลังของสัญญาณรบกวน s

$$H(\omega_1,\omega_2)=\frac{P_f(\omega_1,\omega_2)}{P_f(\omega_1,\omega_2)+P_s(\omega_1,\omega_2)} \tag{2.48}$$

สเปกตรัมกำลังของ AWGN ซึ่งมีค่าคงที่และเป็นอิสระของ ω_1 และ ω_2 จะรู้ได้ที่เครื่องรับ แม้ว่าคุณลักษณะของสัญญาณรบกวนที่บวกเข้าไปจะไม่เปลี่ยนแต่คุณลักษณะของภาพจะเปลี่ยนไปตามแต่ละบริเวณของภาพ เช่นภาพที่มีฉากหลังเรียบกับฉากหน้ามีรายละเอียดมาก สเปกตรัมกำลังงานของทั้งสองก็จะต่างกันเพื่อชดเชยการเปลี่ยนคุณลักษณะของภาพ AW filter จะเป็นฟิลเตอร์ที่มีพารามิเตอร์เปลี่ยนแปลงตามแต่ละส่วนของภาพโดยการเปลี่ยนนี้อาจจะเป็นจุดต่อจุดหรือเป็นบล็อกต่อบล็อกก็ได้

สเปกตรัมกำลังของรูปต้นฉบับนั้นจะไม่ทราบที่เครื่องรับต้องประมาณจากรูปที่ฝังลายน้ำที่รับมา $\hat{h}$ พิจารณาที่บริเวณท้องถิ่น (local region) ของภาพบริเวณหนึ่งถ้าสมมุติว่าสัญญาณภาพต้นฉบับ $f(n_1,n_2)$ ของบริเวณนั้นคงที่จะสามารถจำลองรูปแบบตามสมการที่ (2.49) โดยที่ m_f และ σ_f คือค่าเฉลี่ยบริเวณนั้นและค่าเบี่ยงเบนมาตรฐานของภาพต้นฉบับ w คือ white noise ที่มีค่าเฉลี่ยเป็นศูนย์และมีความแปรปรวนเป็นหนึ่ง

$$f(n_1,n_2)=m_f+\sigma_f w(n_1,n_2) \tag{2.49}$$

ในการประมาณค่าเหล่านี้จากภาพที่ใส่ลายน้ำจะพิจารณาว่าเมื่อค่าเฉลี่ยของสัญญาณรบกวนที่ใส่เป็นศูนย์ซึ่งเป็นกรณีที่ฝัง AWGN ลงไป m_f จะเหมือนกันค่าเฉลี่ยของบริเวณนี้ของภาพที่ใส่ลายน้ำ m_h และเนื่องจาก s เป็นการบวก σ_h^2 จะสามารถเขียนได้ตามสมการที่ (2.50) และค่าประมาณของ $\hat{\sigma}_f^2$ สามารถหาได้จากสมการที่ (2.51) โดยที่ $\sigma_{\hat{h}}^2$ คือความแปรปรวนบริเวณท้องถิ่นของภาพที่ได้รับมาตรวจสอบลายน้ำ ภายในบริเวณท้องถิ่นนี้ transfer function ของ AW Wiener filter จะเขียนได้ตามสมการที่ (2.52) และภาพที่กู้แล้ว $\hat{f}$ ก็จะหาได้ตามสมการที่ (2.53)

$$\sigma_h^2=\sigma_f^2+\sigma_s^2 \tag{2.50}$$

$$\hat{\sigma}_f^2(n,n_2)=\begin{cases}\sigma_{\hat{h}}^2(n_1,n_2)-\sigma_s^2, & if\ \sigma_{\hat{h}}^2(n_1,n_2)>\sigma_s^2\\ 0, & otherwise,\end{cases} \tag{2.51}$$

$$H(\omega_1,\omega_2) = \frac{P_f(\omega_1,\omega_2)}{P_f(\omega_1,\omega_2) + P_s(\omega_1,\omega_2)} = \frac{\hat{\sigma}_f^2}{\hat{\sigma}_f^2 + \sigma_s^2} \qquad (2.52)$$

$$\hat{f}(n_1,n_2) = m_{\hat{h}} + (\hat{h}(n_1,n_2) - m_{\hat{h}}) * \frac{\hat{\sigma}_f^2}{\hat{\sigma}_f^2 + \sigma_s^2}\delta(n_1,n_2) \qquad (2.53)$$

ภาพที่กู้ได้จะถูกเปลี่ยนแปลงตามความสัมพันธ์ระหว่าง $\hat{\sigma}_f^2$ ซึ่งประมาณได้จากบริเวณท้องถิ่นของภาพที่ใส่ลายน้ำและ σ_s^2 ถ้า σ_s^2 มากกว่า $\hat{\sigma}_f^2$ ก็จะมีการเปลี่ยนแปลงมาก แต่กลับกันถ้า $\hat{\sigma}_f^2$ มากกว่า σ_s^2 จะมีการเปลี่ยนแปลงน้อยมาก