โครงการ

การเรียนการสอนเพื่อเสริมประสบการณ์

ชื่อโครงการ การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH
Public Key Validation on SIDH Key Exchange Protocol

ชื่อนิสิต นายดิศรณ์ ฉลาดธัญญกิจ 5933625023

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์
สาขาวิชา วิทยาการคอมพิวเตอร์

ปีการศึกษา 2562

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH

นายดิศรณ์ ฉลาดธัญญกิจ

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต
สาขาวิทยาการคอมพิวเตอร์ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

Public Key Validation on SIDH Key Exchange Protocol

Disorn Chaladtanyakij

A Project Submitted in Partial Fulfillment of the Requirements
For the Degree of Bachelor of Science Program in Computer Science
Department of Mathematics and Computer Science
Faculty of Science
Chulalongkorn University
Academic Year 2019

หัวข้อโครงงาน	การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH
โดย	นายดิศรณ์ ฉลาดธัญญกิจ
สาขาวิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษาโครงงานหลัก	อาจารย์ ดร. วุฒิชัย จงจิตเมตต์

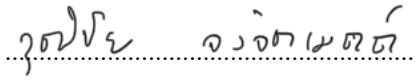
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยอนุมัติให้นับโครงงานฉบับนี้เป็นส่วนหนึ่ง ของการศึกษาตามหลักสูตรปริญญาบัณฑิต ในรายวิชา 2301499 โครงงานวิทยาศาสตร์ (Senior Project)

.....

(ศาสตราจารย์ ดร.กฤษณะ เนียมมณี)

หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

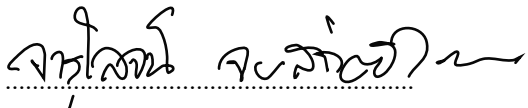
คณะกรรมการสอบโครงงาน



.....

(อาจารย์ ดร.วุฒิชัย จงจิตเมตต์)

อาจารย์ที่ปรึกษาโครงงานหลัก



.....

(ผู้ช่วยศาสตราจารย์ ดร.จารุโลจน์ จงสถิตย์วัฒนา)

กรรมการ

.....

(รองศาสตราจารย์ ดร.วิมลรัตน์ งามอร่ามวรางกูร)

กรรมการ

นายดิศรณ์ ฉลาดธัญญกิจ: การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH. (Public Key Validation on SIDH Key Exchange Protocol) อ.ที่ปรึกษาโครงงานหลัก อาจารย์ ดร.วุฒิชัย จงจิตเมตต์, 92 หน้า

ปัจจุบันโปรโตคอลการแลกเปลี่ยนกุญแจสาธารณะเพื่อใช้ในการสร้างความลับร่วมกันที่เรามักนำไปใช้งานในกระบวนการส่งข้อมูลด้วยการเข้ารหัสแบบสมมาตรต่ออีกทีนั้นมีหลายแบบ ซึ่งตัวโปรโตคอล SIDH นี้เป็นหนึ่งในโปรโตคอลที่เสนอขึ้นเพื่อใช้ในยุคของ post quantum cryptography เพื่อให้สามารถใช้ได้อย่างปลอดภัยในอนาคตแม้ว่าการสร้าง quantum computer จะสามารถทำได้สำเร็จจริง แต่ตัวโปรโตคอลนั้นยังสามารถพัฒนาได้มากกว่าที่เสนออยู่ในเวอร์ชันปัจจุบันที่แบ่งเป็นสองรูปแบบคือ SIDH ที่ใช้รูปแบบกุญแจสาธารณะทั่วไป กับอีกรูปแบบคือ SIDH ที่ใช้กุญแจสาธารณะในรูปแบบบีบอัด เพื่อแก้ไขข้อด้อยหนึ่งในโปรโตคอลส่วนมากที่กำลังแข่งขันกันอยู่ในปัจจุบัน ที่ตัวกุญแจสาธารณะมีขนาดใหญ่กว่าโปรโตคอลที่ใช้ในปัจจุบันมาก แต่การพัฒนากุญแจสาธารณะให้อยู่ในรูปแบบนี้ทำให้กระบวนการตรวจสอบกุญแจสาธารณะที่เดิมมีการใช้ในรูปแบบของกุญแจสาธารณะทั่วไปนั้นไม่สามารถนำมาใช้ได้ รวมถึงกระบวนการบีบอัดกุญแจทำให้เกิดค่าอื่นเพิ่มมาในส่วนของกุญแจสาธารณะ ทำให้ต้องมีกระบวนการตรวจสอบแบบใหม่ที่ใช้กับค่าเหล่านั้นได้ด้วย โดยตัวผู้พัฒนาเห็นว่ากระบวนการแลกเปลี่ยนกุญแจสาธารณะนี้ยังสามารถนำไปปรับเพื่อเพิ่มส่วนของการตรวจสอบกุญแจสาธารณะ เพื่อทำให้ความปลอดภัยของตัวโปรโตคอลมีมากขึ้นโดยที่อาจจะทำให้โปรโตคอลนั้นทำงานได้ช้าลงในกรณีที่ตัวกุญแจสาธารณะนั้นถูกต้องอยู่แล้ว แต่ก็อาจจะทำให้ทำงานเร็วขึ้นได้เช่นกันถ้าหากเจอกุญแจสาธารณะที่ผิดรูปแบบแล้วเราจบการทำงานของโปรโตคอลได้ก่อนที่จะทำสำเร็จ

ในส่วนของการวิเคราะห์กุญแจสาธารณะว่ามีสมบัติใดบ้างที่ใช้ตรวจสอบได้นั้น ส่วนหนึ่งนั้นจะอ้างอิงจากข้อกำหนดของตัวโปรโตคอลเอง และอีกส่วนคือการอ้างอิงย้อนไปถึงงานวิจัยก่อนหน้าที่เคยเสนอแนวทางการตรวจสอบกุญแจสาธารณะในรูปแบบทั่วไปก่อนที่จะถูกบีบอัด แล้วตรวจสอบว่ามีสมบัติทางคณิตศาสตร์อย่างใดบ้างที่สามารถแปลงมาให้ตรวจสอบได้แม้ตัวกุญแจสาธารณะจะถูกบีบอัดมาอยู่อีกรูปแบบหนึ่ง โดยจากผลการศึกษาทำให้เราสามารถคิดวิธีการตรวจสอบขึ้นมาได้สำเร็จ โดยที่ตัวโปรโตคอลนั้นทำงานช้าลงกว่าเดิมที่ไม่มีการตรวจสอบอยู่ที่ประมาณ 10 - 20% และเป็นผลลัพธ์ที่น่าพึงพอใจสำหรับงานวิจัยนี้

ภาควิชา: คณิตศาสตร์และวิทยาการคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา: วิทยาการคอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษาโครงงานหลัก .....

ปีการศึกษา: 2562

5933625023: MAJOR COMPUTER SCIENCE
KEYWORD: PUBLIC KEY CRYPTOGRAPHY / ELLIPTIC CURVE / ISOGENY

DISORN CHALADTANYAKIJ: PUBLIC KEY VALIDATION ON SIDH KEY EXCHANGE PROTOCOL ADVISOR: WUTICHAI CHONGCHITMATE, Ph.D, 92 pp.

Nowadays, there are many protocols for public key exchange for constructing shared secrets. One of the examples is SIDH key exchange protocol, which is nominated to be used in the post quantum cryptography era for security use. The current version of SIDH consists of 2 variants, the generic implementation and compressed public key implementation. The compressed version is created to fix the problem of many post quantum protocols, which is the large public key size compared to existing protocols. However the introduction of compressed public key made it unable to validate whether a public key is being sent correctly or without modification from a malicious party in the session. We concluded that it is possible to implement public key validation to the existing SIDH protocol, which might affect the performance to be slower than version without public key validation. We predict that the trade-off in performance and security will not slow down the protocol more than degree of 2 in terms of the metric we use (CPU clock cycles). We also anticipate that the SIKE protocol with public key validation will yield a faster result for overall running time after many iterations of key exchange if a certain percentage of such exchange consists of malicious, invalid public key that is going to cause an early exit of each key exchange session.

In the section of protocol analysis we look for properties of each public key parameter that might be possible to validate. One of the references is an old publication of public key validation for public key in generic form. Then we check what properties still hold once they are in compressed form. The result of such analysis made it possible for us to validate public keys in compressed form, with the increased CPU clock cycles of around 10 - 20% of the old implementation. This yields a satisfying result for our experiment.

Department: Mathematics and Computer Science Student's Signature Disorn Chaladtanyakij
Field of Study: Computer Science Advisor's Signature Wutichai Chongchitmate
Academic year: 2019

กิตติกรรมประกาศ

โครงงานวิจัยการศึกษาค้นคว้าในหัวข้อ การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH นี้ สามารถดำเนินการได้อย่างลุล่วงด้วยดีเพราะได้รับการอนุเคราะห์ และความช่วยเหลือจากคณาจารย์หลายท่าน ดังนี้

ขอขอบพระคุณ อาจารย์ ดร.วุฒิชัย จงจิตเมตต์ อาจารย์ที่ปรึกษาโครงงานที่คอยให้คำปรึกษาตั้งแต่ช่วงการวางแผนหัวข้อโครงงาน ข้อเสนอแนะระหว่างที่ได้เริ่มทำโครงงาน ทำให้โครงงานนี้มีความคืบหน้าตลอดระยะเวลาที่ทำงานอยู่เสมอ

ขอขอบพระคุณ คณะกรรมการคุมสอบโครงงาน ได้แก่ ผู้ช่วยศาสตรจารย์ ดร.จารุโลจน์ จงสถิตย์วัฒนา และรองศาสตรจารย์ ดร.วิมลรัตน์ งามอร่ามวรางกูร ที่ให้คำแนะนำกำหนดขอบเขตโครงงานให้ชัดเจนมากยิ่งขึ้น ช่วยลดภาระในการทำโครงงานวิจัยนี้อย่างมาก

ขอขอบพระคุณ ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ ที่ได้สนับสนุนแหล่งความรู้สำหรับนิสิตในการดำเนินโครงงานวิจัย รวมถึงงบประมาณที่ใช้ในการดำเนินโครงงานวิจัยนี้

สุดท้ายนี้ขอขอบพระคุณทุกท่านที่ไม่ได้กล่าวนามไว้ในข้างต้น ที่ให้การสนับสนุนในด้านอื่น ที่คอยผลักดันให้โครงงานสำเร็จลุล่วงไปได้ด้วยดี ขอขอบคุณผู้มีส่วนเกี่ยวข้องทุกท่านไว้ ณ โอกาสนี้

ดิศรณ์ ฉลาดธัญญกิจ

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุผล

กระบวนการแลกเปลี่ยนกุญแจสาธารณะนั้นมีที่มาจากการแลกเปลี่ยนข้อความ หรือการสร้างกุญแจการเข้ารหัสด้วยโปรโตคอลการแลกเปลี่ยนกุญแจแบบไม่สมมาตร โดยมีผู้ติดต่อสองฝ่ายได้แก่ $Alice$ และ Bob โดยที่ทั้งสองฝ่ายต้องการจะสื่อสารข้อมูลถึงกันและกันโดยจะใช้สิ่งที่เรียกว่า “กุญแจสาธารณะ $(Public\ key, pk)$” ที่สร้างมาจาก “กุญแจลับ $(Secret\ key, sk)$” ของผู้ติดต่อแต่ละฝ่าย จากนั้นผู้ติดต่อแต่ละฝ่ายจะต้องนำกุญแจสาธารณะที่ได้รับมาจากอีกฝ่ายนั้นไปสร้างเป็นสิ่งที่เรียกว่า “ความลับร่วมกัน $(Shared\ secret, ss)$” ซึ่งผู้ติดต่อแต่ละฝ่ายที่สามารถสร้างความลับร่วมกันนี้ได้ก็จะนำความลับร่วมกันนี้ไปใช้ในกระบวนการเข้ารหัสข้อความด้วยกุญแจแบบสมมาตร

ในปัจจุบันนี้มีโปรโตคอลที่ใช้ในการสร้างความลับร่วมกันนี้อยู่หลายแบบตัวอย่างเช่น“ Diffie-Hellman key exchange” อย่างไรก็ตามตัวโปรโตคอลเหล่านี้ยังคงประสบปัญหาอยู่ที่ว่าหากตัว กุญแจสาธารณะที่ทั้งสองฝ่ายทำการแลกเปลี่ยนกันนั้นถูกปรับแต่งให้กลายเป็นกุญแจสาธารณะที่ไม่พึงประสงค์ กุญแจสาธารณะที่ถูกแก้ไขไปนั้นจะส่งผลต่อความปลอดภัยของโปรโตคอลที่เราใช้งานอยู่ได้ หรือในอีกกรณีหนึ่งคือตัวกุญแจสาธารณะไม่ได้ถูกแก้ไขโดยผู้ไม่ประสงค์ดี แต่ตัวผู้ใช้งานโปรโตคอลเองนั้นไม่ได้ทำงานตามข้อกำหนดของโปรโตคอลที่ตั้งไว้ ส่งผลให้ตัวโปรโตคอลมีความปลอดภัยที่ลดลงเช่นเดียวกัน เพื่อป้องกันไม่ให้เหตุการณ์เหล่านี้เกิดขึ้น ผู้ติดต่อทั้งสองฝ่ายจึงต้องมีกระบวนการที่เรียกว่า “การตรวจสอบกุ ญแจสาธารณะ (Public key validation)” เพื่อให้มั่นใจได้ว่าตัวกุญแจสาธารณะที่ได้รับมาจากอีกฝ่ายนั้นเป็นไปตามข้อกำหนดที่ได้เลือกไว้ และไม่ได้ถูกปรับแต่งมาโดยผู้ดักฟังที่ไม่พึงประสงค์

ตัวโปรโตคอลที่งานนี้สนใจนั้นมีชื่อว่า “Supersingular isogeny Diffie-Hellman key exchange (SIDH)” เป็นโปรโตคอลที่กำลังมีการพัฒนาอยู่เพื่อให้สามารถใช้ได้ในระบบการสร้างความลับร่วมกันในยุคของ Post-Quantum Cryptography กล่าวคือ โปรโตคอลนี้จะยังไม่สูญเสียความปลอยภัยไปแม้ว่าผู้ที่ประสงค์ร้ายจะครอบครอง Quantum computer อยู่ โดยความยากในการแก้ไขปัญหาเพื่อให้สามารถทำลายความปลอดภัยของ SIDH ได้นั้นอยู่ที่ $O(p^{1/4})$ สำหรับ Classical computer และ $O(p^{1/6})$ [3] สำหรับ Quantum computer เมื่อให้ p แทนจำนวน bit ของจำนวนเฉพาะที่ตัวโปรโตคอลนั้นเลือกใช้ (ปัจจุบันมี 434, 503, 610 และ 751 bit ให้เลือกใช้ในขณะนี้) โดยในงานวิจัยของเรานี้จะสนใจ p ที่มีขนาด 751 bit

ตัวโปรโตคอล SIDH นั้นถูกเสนอครั้งแรกในปี 2011 ที่ในขณะนั้นสามารถใช้ได้กับกุญแจสาธารณะที่สร้างมาอย่างถูกต้องเท่านั้น จนกระทั่งในปี 2016 ที่ Microsoft Researcher [5] ได้พัฒนาต่อให้สามารถทำงานได้เร็วยิ่งขึ้นกว่าเดิม รวมไปถึงวิธีการตรวจสอบกุญแจสาธารณะในรูปแบบทั่วไปอีกด้วย ถัดมาภายในปี 2016 - 2017 จะเป็นเรื่องของการพัฒนากุญแจสาธารณะให้บีบอัดลงมาเหลือขนาดเพียงครึ่งหนึ่งของขนาดเดิม ซึ่งทำให้วิธีตรวจสอบกุญแจสาธารณะแบบเดิมนั้นไม่สามารถทำได้บนกุญแจสาธารณะแบบบีบอัด

ถัดมาตัวโปรโตคอล SIDH ได้มีการพัฒนาใหม่ให้อยู่ในรูปแบบของ SIKE คือเพิ่มในส่วนของการทำ key encapsulation และ key encryption ช่วยป้องกันกุญแจสาธารณะได้เพียงส่วนหนึ่ง เป็นวิธีการตรวจสอบกุญแจสาธารณะแบบอ้อม คือถ้าหากมีฝ่ายใดฝ่ายนึงไม่ทำตามโปรโตคอลที่กำหนด ตัวความลับร่วมกันที่คำนวณได้ก็จะมีค่าต่างกันส่งผลให้ไม่สามารถติดต่อกันได้ รวมถึงแก้ไขปัญหาของการใช้กุญแจเดิมในการเข้ารหัสด้วยจากการใช้ hash function

อย่างไรก็ตามตัว SIKE นี้ยังไม่สามารถบอกได้ว่าเป็นการตรวจสอบกุญแจสาธารณะอย่างแท้จริง เพราะตัวโปรโตคอลยังต้องคำนวณทุกขั้นตอนตามปกติแม้ว่าจะได้รับกุญแจสาธารณะที่ไม่ถูกต้อง โดยส่วนนี้เองจะเกิดปัญหาขึ้นได้ในรูปแบบของกุญแจสาธารณะที่มีการบีบอัดเพราะส่วนหนึ่งของกระบวนการที่เราถอดรหัสกุญแจบีบอัดให้กลับไปอยู่ในรูปเดิมนั้นเสียเวลาในการคำนวณอย่างมาก ซึ่งถ้าเราปล่อยให้ตัวโปรโตคอลเราทำงานโดยไม่ตรวจสอบเลยว่ากุญแจสาธารณะที่ได้รับมานั้นถูกต้องหรือไม่จะเป็นการสิ้นเปลืองเวลาและทรัพยากรของทางฝั่งผู้ถอดรหัสกุญแจอย่างมาก ด้วยเหตุนี้เองเราจึงสรุปว่า กระบวนการตรวจสอบกุญแจสาธารณะนั้นยังมีความจำเป็นอยู่ แม้ว่าเราจะเลือกใช้โปรโตคอล SIKE โดยการตรวจสอบนั้นต้องตรวจโดย $(bit, t_1, t_2, t_3, A, s, r)$ ที่ได้รับเท่านั้นก่อนที่จะไปเข้ากระบวนการถอดรหัส และเราก็ต้องตรวจสอบผลกระทบที่เกิดขึ้นจากการเพิ่มวิธีการตรวจสอบนี้ด้วย

1.2 วัตถุประสงค์

เพื่อออกแบบอัลกอริทึมสำหรับการตรวจสอบกุญแจสาธารณะที่ได้รับมาจากผู้ติดต่ออีกฝ่ายบนโปรโตคอลการแลกเปลี่ยนกุญแจสาธารณะแบบ SIDH/SIKE จากนั้นตรวจสอบผลกระทบที่เกิดขึ้นทางด้านประสิทธิภาพเมื่อเทียบกับกระบวนการแลกเปลี่ยนกุญแจสาธารณะที่ไม่มีส่วนการตรวจสอบกุญแจสาธารณะ

1.3 ขอบเขตของโครงงาน

อัลกอริทึมจะต้องสามารถตรวจสอบกุญแจสาธารณะในรูปแบบบีบอัด $(bit, t_1, t_2, t_3, A, s, r)$ ว่าถูกต้องตามข้อกำหนดของโปรโตคอลหรือไม่ ที่เกิดขึ้น 2 ครั้งสำหรับทั้งโปรโตคอล SIKE คือช่วงที่มีการทำ

encapsulation ที่ Bob ได้รับกุญแจสาธารณะ pk_A จาก $Alice$ และอีกครั้งหนึ่งคือตอนทำ decapsulation ที่ $Alice$ ได้รับกุญแจสาธารณะ pk_B จาก Bob โดยเราคาดว่าจะสามารถทำได้ภายใต้ข้อกำหนดดังนี้

1. งานของเราจะเริ่มมาจากตัวโปรโตคอล SIKEp751_Compressed ที่มีการเขียนไว้อยู่แล้วโดยมีกระบวนการทางคณิตศาสตร์, การตั้งค่าโปรโตคอลที่เกี่ยวข้องกับตัวโปรโตคอลนี้อยู่ก่อนแล้ว แต่ไม่มีการทำส่วนการตรวจสอบกุญแจสาธารณะ
2. เราจะทดสอบตัวโปรโตคอลนี้บนระบบปฏิบัติการ Linux, macOS
3. การจะใช้ตัวโปรโตคอลนี้ต้องมี CMake เวอร์ชัน 3.5 หรือมากกว่า และ GMP เวอร์ชัน 6.1.2 หรือมากกว่า และ C99 Compiler เช่น Clang, GCC
4. ผลกระทบจากอัลกอรึทึมที่เราใช้ต้องไม่ทำให้มาตราการวัดผลที่เราใช้ประเมินการทำงานของโปรโตคอลนี้คือจำนวนรอบของ CPU เพิ่มไปเกิน 2 เท่าเมื่อเทียบจากค่าเดิมที่ไม่มีการตรวจสอบกุญแจสาธารณะ
5. ความแม่นยำในการตรวจสอบกุญแจสาธารณะแบบที่มีการบีบอัด ควรมีความใกล้เคียงกับการตรวจสอบบนกุญแจสาธารณะที่ไม่มีการบีบอัด

1.4 ขั้นตอนการดำเนินงาน

1.4.1 แผนการศึกษา

1. ค้นหาและศึกษางานวิจัยที่เกี่ยวข้องกับการพัฒนาของตัวโปรโตคอล SIDH/SIKE โดยเริ่มจากศึกษางานวิจัยที่เสนอแนวทางการประยุกต์ปัญหาของ isogeny walk problem มาใช้เป็นโปรโตคอล SIDH ของ Jao, David และ Luca de Feo และงานวิจัยอื่นที่พัฒนาให้ลักษณะของกุญแจสาธารณะมาอยู่ในรูปแบบปัจจุบัน
2. ศึกษาค้นคว้าโครงสร้างทางคณิตศาสตร์ที่เกี่ยวของกับ SIDH จากบทเรียนที่เกี่ยวข้องกับ elliptic curve และ isogeny
3. นำตัวโปรโตคอล SIKE ที่มีอยู่มาแยกส่วนจากเดิมที่ทดสอบทั้ง SIKE ให้ทดสอบแค่เพียงในส่วนของ SIDH แยกออกมา จากเพิ่มส่วนการตรวจสอบกุญแจสาธารณะ
4. ออกแบบวิธีการตรวจสอบกุญแจสาธารณะในรูปแบบบีบอัด
5. นำวิธีที่ออกแบบมาตรวจสอบประสิทธิภาพการทำงานของโปรโตคอลว่าเป็นไปตามเป้าหมายที่กำหนดไว้

6. สรุปผลการดำเนินงาน
7. จัดทำเอกสาร

1.4.2 ตารางระยะเวลาการทำงาน

Activity	2562					2563			
	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย.
1. ศึกษางานวิจัยที่เกี่ยวข้อง									
2. ศึกษาหลักทางคณิตศาสตร์ที่เกี่ยวข้องกับ SIDH									
3. ทดสอบวิธีการตรวจสอบแบบดั้งเดิม									
4. พิจารณาโครงสร้างทางพีชคณิตและออกแบบวิธีการตรวจสอบกุญแจสาธารณะ									
5. นำวิธีที่ออกแบบมาไปทดสอบจริง									
6. ตรวจสอบผลกระทบที่เกิดขึ้น									
7. เตรียมเอกสาร									

ตารางที่ 1.1 ขั้นตอนการดำเนินงาน

1.5 ประโยชน์ที่ได้รับ

1.5.1 ประโยชน์ต่อตัวนิสิต

1. ได้ศึกษาเกี่ยวกับพีชคณิตการคำนวณ
2. ได้ความรู้มากขึ้นเกี่ยวกับการดำเนินการทางคณิตศาสตร์ในภาษา C
3. ได้รับประสบการณ์การทำงานบน cryptographic protocol
4. ได้รับประสบการณ์การทำงานร่วมกับบุคคลอื่นในแวดวง cryptography

1.5.2 ประโยชน์ของตัวงาน

1. ได้วิธีการตรวจสอบกุญแจสาธารณะสำหรับโปรโตคอล SIDH ในรูปแบบของกุญแจที่มีการบีบอัด
2. อาจจะทำให้เกิดโปรโตคอลอื่นที่มีพื้นฐานมาจาก SIDH แต่ไม่ได้พึ่งวิธีการแบบ SIKE แต่มีความปลอดภัยคล้ายกันอยู่โดยที่ยังตรวจสอบกุญแจสาธารณะได้

1.6 โครงสร้างของรายงาน

บทที่ 2 จะกล่าวถึงความรู้พื้นฐานที่ควรมีและงานวิจัยที่เกี่ยวข้องกับโครงงานนี้

บทที่ 3 จะกล่าวถึงวิธีการศึกษาค้นคว้า ว่าทำไมจึงควรมีการตรวจสอบที่ขั้นตอนใด รวมไปถึงวิธีการออกแบบการทดสอบและประเมินผล

บทที่ 4 จะกล่าวถึงผลลัพธ์ที่ได้จากการทดสอบ

บทที่ 5 จะกล่าวถึงข้อสรุปและข้อเสนอแนะที่ได้จากการทำโครงงาน

บทที่ 2
ความรู้และงานวิจัยที่เกี่ยวข้อง

2.1 Public key cryptography

ในส่วนแรกที่เราต้องกล่าวถึงในงานวิจัยนี้คือกระบวนการของ Public key cryptography (วิทยาการเข้ารหัสโดยกุญแจสาธารณะ, วิทยาการเข้ารหัสแบบอสมมาตร) ที่มีพื้นฐานมาจากการที่ผู้ติดต่อแต่ละฝ่ายจะมีส่วนที่เป็นกุญแจสาธารณะที่พร้อมจะกระจายให้ผู้ติดต่ออื่นรับทราบได้ กับอีกส่วนหนึ่งคือกุญแจลับที่มีเพียงตนเองเท่านั้นที่ทราบ โดยตัววิธีที่ใช้ในการสร้างจะสร้างสองสิ่งนี้มาเป็นคู่กัน เราอาจจะมองได้สองแบบคือ การที่มีอัลกอริทึม $KeyGen \rightarrow (sk, pk)$ รับค่า random seed ไปแล้วสร้างคู่ของกุญแจลับกับกุญแจสาธารณะมาให้ หรืออีกกรณีหนึ่งคือผู้ใช้เลือกกุญแจลับขึ้นมาเองตามรูปแบบของโปรโตคอล แล้วนำไปสร้างเป็นกุญแจสาธารณะที่ขึ้นกับกุญแจลับนั้น

กระบวนการเข้ารหัสแบบอสมมาตรนี้สามารถนำไปประยุกต์ใช้ได้กับหลายอย่าง เช่นการกระจายกุญแจ (Key distribution), การลงลายเซ็นดิจิตอล (Digital signature), การเข้ารหัสและถอดรหัสข้อความ แต่ในส่วนของงานวิจัยนี้เราจะเน้นไปที่สองวิธีการหลักดังนี้

2.1.1 Public key exchange

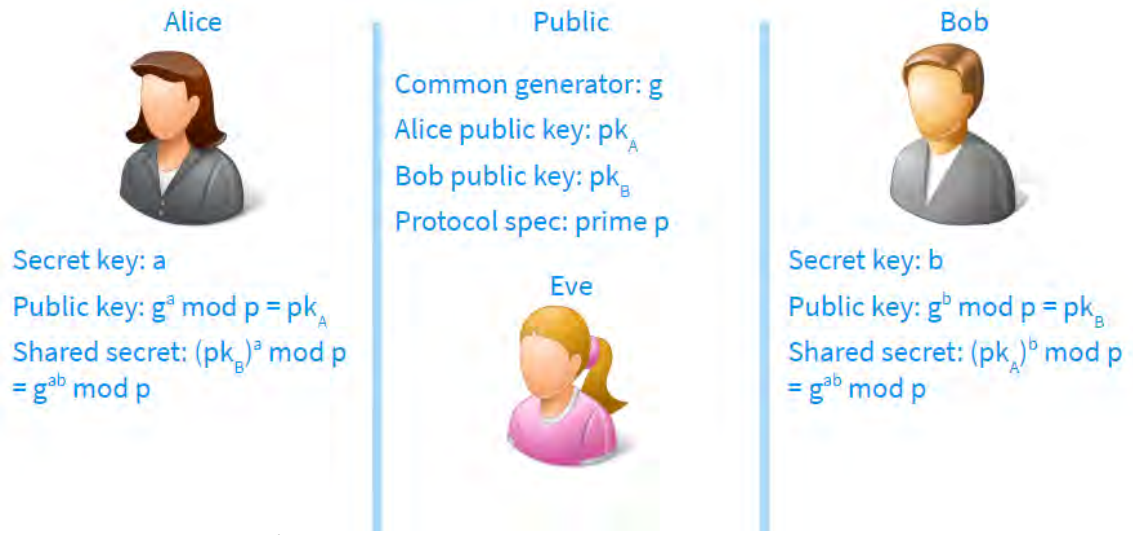
การแลกเปลี่ยนกุญแจสาธารณะเป็นกระบวนการที่ผู้ติดต่อสองฝ่าย ในส่วนนี้จะแทนด้วย $Alice$ (A) และ Bob (B) ที่ต้องการสร้างความลับร่วมกัน โดยความลับร่วมกันนี้เองที่จะนำไปใช้ในกระบวนการเข้ารหัสแบบสมมาตร เนื่องจากในปัจจุบันการส่งข้อความผ่านการเข้ารหัสแบบอสมมาตรนั้นยังมีข้อจำกัดอยู่มาก เช่นส่งได้เฉพาะข้อความขนาดสั้น หรือใช้เวลานานในการประมวลผลเพื่อเข้ารหัสและถอดรหัสข้อความ

พื้นฐานของตัวโปรโตคอลนี้คือ $Alice, Bob$ ต่างจะมีกุญแจลับของตนเองคือ sk_A, sk_B และในตัวโปรโตคอลก็จะกำหนดข้อมูลเริ่มต้น เรียกว่า generator g พร้อมกับวิธีการทำงานของโปรโตคอล ทั้งสองฝ่ายจะนำกุญแจลับไปสร้างเป็นกุญแจสาธารณะตามวิธีที่โปรโตคอลกำหนดกลายเป็น pk_A, pk_B ที่แลกเปลี่ยนกันและกันและผู้ดักฟังอื่นก็สามารถรับรู้ข้อมูลนี้ได้ เมื่อ $Alice$ ได้รับข้อมูล pk_B มาแล้ว ก็นำไปคำนวณเป็นความลับร่วมกัน $(shared\ secret, ss)$ เช่นเดียวกับ Bob ที่ได้รับข้อมูล pk_A มาแล้วนำไปคำนวณเป็น $shared\ secret$ โดยตัวโปรโตคอลเองนั้นจะยืนยันได้ว่า $shared\ secret$ ที่ $Alice$ และ Bob คำนวณได้นั้นจะมีค่าเท่ากัน

ตัวอย่างของกระบวนการแลกเปลี่ยนกุญแจสาธารณะที่เป็นพื้นฐานของหลาย โปรโตคอล รวมไปถึงโปรโตคอลที่เราทำงานวิจัยนี้อยู่ด้วยมีชื่อว่า Diffie-Hellman key exchange ที่อาศัยสมบัติของ group ภายใต้การคูณของ $mod\ p$

1. ตัวโปรโตคอลจะบอกข้อมูลของค่าจำนวนเฉพาะ p และค่าเริ่มต้น g ให้ทุกฝ่ายรับทราบ
2. $Alice$ เลือกจำนวนเต็มที่อยู่ในช่วง $[1, p-1]$ เก็บเป็น a ใช้เป็นกุญแจลับ sk_A
3. Bob เลือกจำนวนเต็มที่อยู่ในช่วง $[1, p-1]$ เก็บเป็น b ใช้เป็นกุญแจลับ sk_B
4. $Alice$ คำนวณ $g^a\ mod\ p$ เก็บเป็นกุญแจสาธารณะ pk_A แล้วส่งค่านี้ให้ Bob รับทราบ
5. Bob คำนวณ $g^b\ mod\ p$ เก็บเป็นกุญแจสาธารณะ pk_B แล้วส่งค่านี้ให้ $Alice$ รับทราบ
6. $Alice$ ได้รับ pk_B คำนวณ $pk_B^a\ mod\ p$ กลายเป็นความลับร่วมกัน
7. Bob ได้รับ pk_A คำนวณ $pk_A^b\ mod\ p$ กลายเป็นความลับร่วมกัน

ตามที่เห็นได้ในภาพที่ 2.1



ภาพที่ 2.1 ตัวอย่างของโปรโตคอล Diffie-Hellman key exchange

ตัวโปรโตคอลนี้อาศัยสมบัติของปัญหาที่โด่งดังทางคณิตศาสตร์คือ discrete logarithm ที่บอกว่าการทราบว่าค่า $pk_A = g^a\ mod\ p$ ไม่สามารถทำให้ผู้ดักฟังสามารถหาค่า a กลับมาได้ เช่นเดียวกับการที่ทราบว่าค่า $pk_B = g^b\ mod\ p$ ก็ไม่สามารถทำให้หาค่า b กลับมาได้ หรือการที่ทราบทั้งสองค่า pk_A, pk_B แต่ไม่ทราบค่า a, b ก็ไม่ได้ทำให้สามารถนำไปสร้างความลับร่วมกันที่ตรงกับที่ $Alice, Bob$ ครอบครองอยู่ได้

2.1.2 Public key encryption

การเข้ารหัสด้วยกุญแจสาธารณะเป็นกระบวนการที่ผู้ติดต่อสองฝ่ายที่เราแทนด้วย $Alice$ และ Bob ต้องการส่งข้อความไปหากันและกันในรูปแบบของข้อความที่ถูกเข้ารหัส $(ciphertext, ct)$ ส่วนมากนั้นมักจะนำวิธีนี้ไปใช้ในการส่งข้อความขนาดสั้น

พื้นฐานการทำงานของตัวโปรโตคอลนี้คือจะต้องมีฟังก์ชันที่ชื่อว่า $Gen(\,)$ ที่ทำหน้าที่ในการสร้างคู่ของกุญแจลับและกุญแจสาธารณะ, $Enc(\,)$ ที่ใช้ในการเข้ารหัสข้อความ และ $Dec(\,)$ที่ใช้ในการถอดรหัสข้อความที่ถูกเข้ารหัส

การทำงานของการเข้ารหัสด้วยกุญแจสาธารณะนั้นโดยมากจะเหมือนกันหมด ต่างกันแค่ขึ้นอยู่กับว่าจะเลือกใช้ฟังก์ชันอะไรเท่านั้นมาใช้เข้ารหัสและถอดรหัส โดยเราสามารถอธิบายขั้นตอนการทำงานได้ดังนี้

1. $Alice$ ใช้ $Gen(\,) \rightarrow (sk_A, pk_A)$ และประกาศ pk_A ให้ทุกฝ่ายรับทราบ
2. Bob ใช้ $Gen(\,) \rightarrow (sk_B, pk_B)$ และประกาศ pk_B ให้ทุกฝ่ายรับทราบ
3. $Alice$ ทำการเลือกข้อความที่จะส่งเป็น m_A จากนั้นนำไปเข้ารหัสด้วยฟังก์ชัน$Enc(m_A, pk_B) = ct_A$ แล้วส่งไปให้ Bob
4. Bob ทำการเลือกข้อความที่จะส่งเป็น m_B จากนั้นนำไปเข้ารหัสด้วยฟังก์ชัน $Enc(m_B, pk_A) = ct_B$ แล้วส่งไปให้ $Alice$
5. $Alice$ อ่านข้อความจาก Bob โดยที่ $m_B = Dec(ct_B, sk_A)$
6. Bob อ่านข้อความจาก $Alice$ โดยที่ $m_A = Dec(ct_A, sk_B)$

ตามที่เห็นได้ในภาพที่ 2.2

ภาพที่ 2.2 ตัวอย่างของกระบวนการ Public key encryption

2.2 Elliptic curve

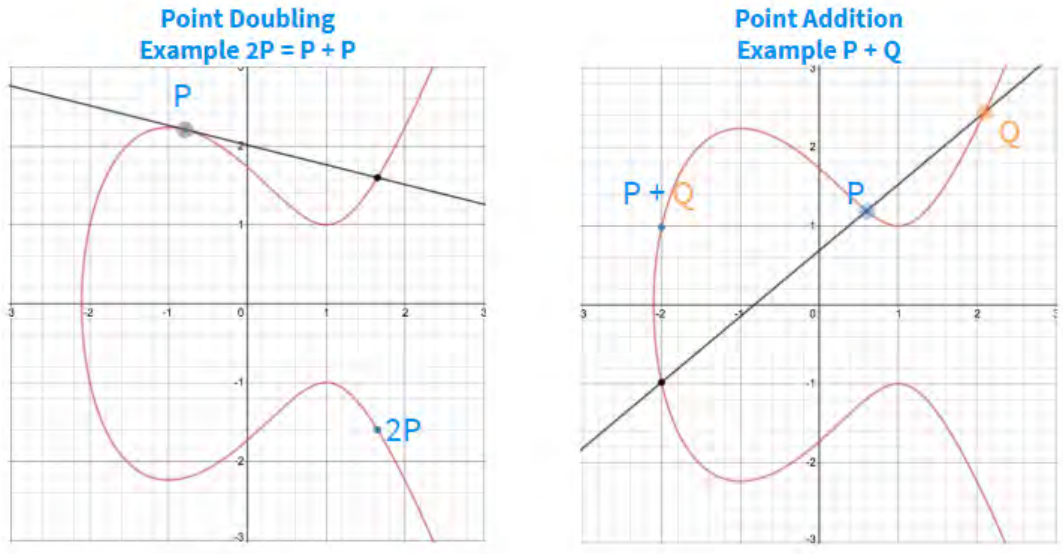
นิยามของ elliptic curve นั้นมีหลายแบบ ในส่วนนี้จะพูดถึงนิยามหนึ่งที่ใช้ในการอธิบายรูปแบบสมการ และกระบวนการทางคณิตศาสตร์ที่เกิดขึ้นภายใต้นิยามของ elliptic curve

เรานิยามให้ $E: y^2 = x^3 + ax + b$ แทน elliptic curve (เรียกรูปแบบนี้ว่า short weierstrass form) และให้จุด $P_1 = (x_1, y_1)$ และจุด $P_2 = (x_2, y_2)$ เป็นจุดบน E และไม่ใช่จุดที่อนันต์ โดยเราจะแทนจุดที่อนันต์ด้วย O เราสามารถนิยามกระบวนการบวก (+) บน elliptic curve ได้ดังนี้ [8]

- $P + O = O + P = P$ สำหรับทุกจุด $P \in E$
- ถ้า $x_1 = x_2$ และ $y_1 = -y_2$ แล้ว $P_1 + P_2 = O$
- ในกรณีอื่นให้ทำการคำนวณค่า $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ ถ้าหาก $P \neq Q$ หรือ $\lambda = \frac{3x_1^2 + a}{2y_1}$ ถ้าหาก $P = Q$ และคำนวณจุด $P_1 + P_2 = P_3 = (x_3, y_3)$ ได้จาก $x_3 = \lambda^2 - x_1 - x_2$ และ $y_3 = -\lambda x_3 - y_1 + \lambda x_1$
- กระบวนการข้างต้นนี้ทำให้เกิด Abelian group

กระบวนการอีกอย่างหนึ่งคือ การคูณจุด P บน elliptic curve ด้วยจำนวนเต็ม n เขียนแทนด้วย $[n]P$ คือแทนการ + ด้วยจุดตัวเอง n ครั้ง เช่นเราเขียนได้ว่า $[3]P = P + P + P$

ภาพตัวอย่างด้านล่างเป็นตัวอย่างของกระบวนการบวกและการคูณด้วย 2 กับจุด P บน elliptic curve E ที่นิยามเหนือจำนวนจริง R



ภาพที่ 2.3 ตัวอย่างของกระบวนการบวกระหว่างจุดและการคูณด้วยจำนวนเต็มกับจุดบน elliptic curve

ที่มา: https://www.desmos.com/calculator/ialhd71we3 - Elliptic Curve Points

สำหรับ elliptic curve E ที่นิยามเหนือ field k เรามักจะเขียนเซตกลุ่มของจุด P ที่อยู่บน elliptic curve E ด้วย $E(k)$

2.2.1 Elliptic curve over finite field

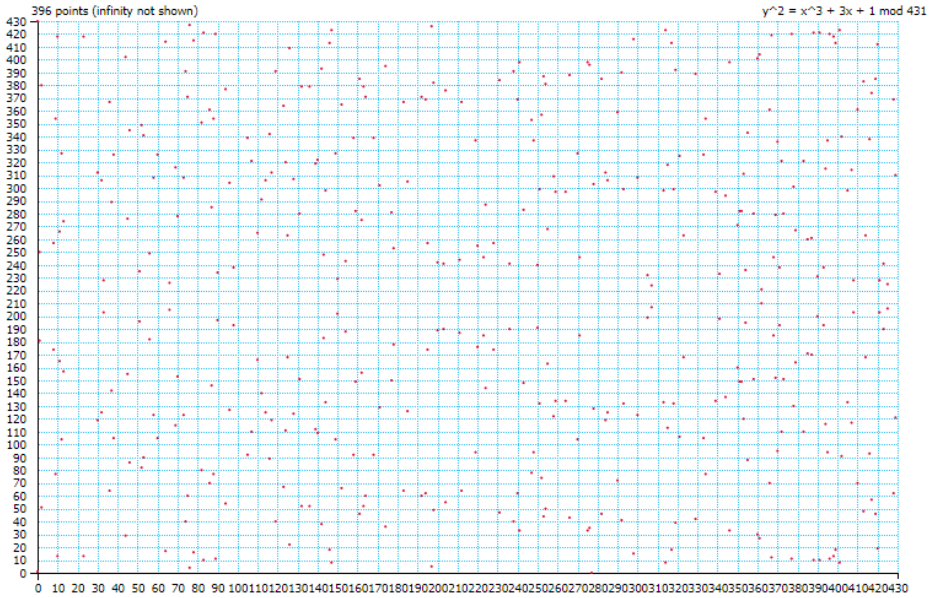
ในส่วนนี้จะพูดถึง elliptic curve E ที่นิยามเหนือ finite field F_q โดยให้ $q = p^a$ เมื่อให้ p เป็นจำนวนเฉพาะที่มีค่ามาก (ในที่นี้คือ $p > 3$) และ $a \in N$, elliptic curve ที่นิยามเหนือ F_q ในรูปแบบ short weierstrass และถูกนิยามโดยจำนวนเต็ม $A, B \in F_q$ จะเขียนเป็นเซตของจุดได้ดังนี้คือ

$$E(F_q) = \{(x, y) \in (F_q)^2 : y^2 = x^3 + Ax + B\} \cup \{O\}$$

เมื่อกำหนดให้ O คือจุดที่อนันต์ที่มีค่า $(x : y : z) = (0 : 1 : 0)$ บน projective curve $E : y^2 = x^3 + Axz^2 + Bz^3$ (บางครั้งเราจะใช้รูปแบบนี้เพื่อให้คำนวณได้เร็วขึ้น [5])

และในบางครั้งเราจะพูดถึงเซต $E(F_{\overline{q}})$ สำหรับทุกจุดของ E เหนือ algebraic closure $F_{\overline{q}}$ ของ F_q โดยที่กระบวนการทางคณิตศาสตร์การบวกและการคูณยังทำเหมือนเดิมเช่นเดียวกับ curve ทั่วไป

สมบัติสำคัญที่เราสนใจบน elliptic curve ที่นิยามเหนือ finite field นั้นมีหลายอย่าง ล้วนแต่เป็นสิ่งที่สำคัญสำหรับโปรโตคอลที่มีการใช้งานจริงในระบบ cryptography ในปัจจุบันเช่น ค่า j-invariant ของ E ที่เขียนแทนด้วย $j(E)$, order ของจุด P ที่อยู่บน E, cardinality, characteristic, endomorphism และ subfield curve ของตัว curve โดยในส่วนนี้จะขอเลือกนำมาอธิบายแค่ในส่วนที่เกี่ยวข้องกับตัวงานที่เราทำงานวิจัยเท่านั้น



ภาพที่ 2.4 ตัวอย่างของ elliptic curve ที่อยู่เหนือ F_{431}

ที่มา: https://graui.de/code/elliptic2/

2.2.2 Ordinary and Supersingular elliptic curve

เราจะเริ่มจากนิยามของ n-torsion group ของ elliptic curve ที่นิยามเหนือ finite field k กล่าวคือ สำหรับจำนวนเต็มบวก $n \in Z^+$ เราจะเรียกเซต n-torsion group ของ elliptic curve E เขียนแทนด้วย $E[n]$ ได้โดย $E[n] = \{P \in E(K) : [n]P = O)\}$ (O ในที่นี้คือจุดที่อนันต์ และ K คือ algebraic closure ของ k)

แต่เราจะมีอีกนิยามหนึ่งคือ ถ้าให้ k แทน characteristic ของ finite field K แล้ว $E[n]$ จะสามารถเขียนได้อีกแบบคือ [4]

$$E[n] \cong (Z/nZ)^2 \text{ ถ้า } k \nmid n$$

หรือถ้าหาก characteristic k ของ field K มีค่าเท่ากับจำนวนเฉพาะ $p > 0$ แล้วจาก [4] และ [8] จะได้ว่า

- $E[p^i] \cong Z/p^iZ$ สำหรับทุกค่า $i > 0$ ถ้า E เป็น ordinary curve
- $E[p^i] \cong \{O\}$ สำหรับทุกค่า $i > 0$ ถ้า E เป็น supersingular curve

อีกนิยามหนึ่งที่สนใจคือ สำหรับ elliptic curve E ที่นิยามเหนือ F_q ที่ $q = p^a$ เราให้นิยาม cardinality (จำนวนสมาชิก) ของเซต X ใด ๆ เขียนแทนด้วย $\#X$ จะได้ว่า $\#E(F_q) = q + 1 - t$

สำหรับจำนวนเต็ม $t \in Z^+$ โดยที่ $|t| \leq 2\sqrt{q}$ [2] และอีกอย่างหนึ่งคือเราจะเรียก elliptic curve E นั้นว่าเป็น supersingular curve ถ้าหาก $t \mid p$ (t หาร p ลงตัว) และเป็น ordinary curve ในกรณีที่ $t \nmid p$

สิ่งที่ได้ตามมาจากนิยามนี้คือ elliptic curve E จะถือว่าเป็น supersingular ถ้าหาก $\#E(F_q) \equiv 1 \; mod \; p$ และการเป็น supersingular curve จะบอกต่อได้อีกด้วยว่า $\#E(F_{q^n}) \equiv 1 \; mod \; p$ สำหรับทุกค่า $n \in N$ [4]

อีกนิยามหนึ่งคือจาก n-torsion group $E[n]$ เราจะบอกได้ว่าถ้าหาก $n \nmid p$ แล้วจะได้ว่า $\#E[n] = n^2$ รวมไปถึงอีกข้อสรุปคือ $E[n]$ จะเป็นผลลัพธ์จาก product ของ cyclic subgroup 2 กลุ่มที่มี order n และข้อสรุปสุดท้ายคือ

$E[p] = \{O\}$ ถ้า E เป็น supersingular curve แต่ถ้าหาก E เป็น ordinary curve จะได้ว่า $\#E[p] = p$ [4] (การที่ $E[p] = \{O\}$ บอกได้ว่าทุกจุด $P \in E$ ไม่มีจุดใดที่ order ของจุดคือ p)

2.2.3 Isomorphism and j-invariant

ก่อนจะพูดถึง isogeny จะต้องทำความรู้จักกับ morphism, isomorphism ในนิยามของ elliptic curve ก่อน โดยมีนิยามดังนี้ [2]

morphism ของ elliptic curve $f: E \to E'$ เป็นฟังก์ชันที่อธิบายได้โดยอัตราส่วนของพหุนามที่ส่งจุดบน E ไปยัง E' ในขณะที่ isomorphism f คือ morphism f' ที่เข้าเงื่อนไข $f(O_E) = O_{E'}$ (ส่งจุดที่อนันต์บน E ไปยังจุดอนันต์บน E') และ inverse ของ f' ก็ต้องเป็น morphism เช่นเดียวกัน ทำให้เรานิยามต่อมาได้ว่า isomorphism เป็น bijection $E(F_{\overline{q}}) \to E'(F_{\overline{q}})$ ที่ไม่แน่นอนเสมอไปว่าจะเป็นการส่งค่าจาก $E(F_q) \to E'(F_q)$

ในกรณีที่เรามี E นิยามเหนือ F_q และบอกว่า $\#E(F_q) = q + 1 - t$ เราจะมี elliptic curve E' อีกอันหนึ่งที่นิยามเหนือ F_q เรียกว่า quadratic twist ของ E แล้วเราจะสามารถบอกได้ว่า $\#E'(F_q) = q + 1 + t$

นิยามถัดมาที่เราสนใจคือการคำนวณค่า j-invariant สำหรับ elliptic curve $E_{A,B}: y^2 = x^3 + Ax + B$ ที่นิยามได้ว่า

$$j(E_{A,B}) = \frac{1728 \times 4A^3}{4A^3 + 27B^2}$$

แล้วเราจะบอกว่าการจะมี isomorphism $f: E \to E'$ ได้ก็ต่อเมื่อ $j(E) = j(E')$ เท่านั้น

สุดท้ายก่อนที่จะพูดถึงเรื่อง isogeny คือการที่ทุก supersingular curve E ที่นิยามเหนือ F_p นั้นจะ isomorphic กับ supersingular curve ที่นิยามบน F_{p^2} ทำให้เราบอกได้ว่าค่า j-invariant นั้นจะต้องอยู่ใน F_{p^2} และจากทฤษฎีนั้นได้กล่าวไว้ว่าเรามีทั้งหมดราว $\left\lfloor \frac{p}{12} \right\rfloor$ isomorphism class (j-invariant) ของ

supersingular curve [2] และมีราว $O\left(\sqrt{plog(p)}\right)$ ที่มีค่า j-invariant อยู่ใน F_p ทำให้เราต้องเลือก p ที่มีค่ามากในการนำไปใช้ในระบบ cryptography ในปัจจุบัน กับอีกข้อสรุปก็คือคือค่าของ $\#E\left(F_{p^2}\right)$ สำหรับ supersingular curve E นั้นจะมีเพียงแค่ $(p+1)^2$ หรือไม่ก็ $(p-1)^2$ เท่านั้น

2.2.4 Isogeny

Isogenies เป็น group homomorphisms ระหว่าง elliptic curve ที่แตกต่างกัน โดยมีหน้าที่สำคัญในหลายอย่างทั้งทางด้านทฤษฎีและในด้านการนำไปใช้สร้างโปรโตคอลเพราะเป็นสมบัติที่ทำให้เราบอกความเกี่ยวข้องของ curve ที่แตกต่างกันได้ เพราะการที่ curve ที่แตกต่างกันนั้นมี homomorphism กันจะรักษาสมบัติทางโครงสร้างของ elliptic curve ให้เหมือนกันได้ และการที่ตัวโปรโตคอลที่เราจะพูดถึงนี้นั่นเริ่มต้นจาก curve เริ่มต้นอันหนึ่งคือ E_0 ที่มีสมบัติชนิดหนึ่งอยู่คือความเป็น supersingular, จำนวนสมาชิก และการนิยามอยู่เหนือ F_{p^2} แล้วทำการคำนวณหา isogeny จาก curve นั้นกลายเป็นกุญแจสาธารณะ E_A, E_B ทำให้เราสามารถตรวจสอบกุญแจสาธารณะโดยอาศัยสมบัติเหล่านั้นได้นี้เอง โดยเราจะกำหนดนิยามของ isogeny ได้ดังนี้ [2]

ให้ E_1, E_2 เป็น elliptic curve ที่นิยามเหนือ field K, isogeny จะบอกว่าคือ non-constant homomorphism $\phi: E_1(K) \to E_2(K)$ ที่อธิบายได้โดย rational function หรือเขียนอีกแบบคือ $\phi(P+Q) = \phi(P) + \phi(Q)$ แล้วสำหรับทุก จุด $P, Q \in E_1(K)$ เราจะมี rational function R_1, R_2 ที่ถ้าหาก $\phi(x_1, y_1) = (x_2, y_2)$ แล้ว $x_2 = R_1(x_1, y_1)$, $y_2 = R_2(x_1, y_1)$ สำหรับทุกค่า $(x_1, y_1) \in E_1(K)$ เราสามารถเขียน isogeny ϕ ได้เป็น

$$(x_2, y_2) = \phi(x_1, y_1) = \left(r_1(x_1), y_1 r_2(x_1)\right)$$

เมื่อให้ r_1, r_2 เป็น rational function ที่ถ้าหากว่าสัมประสิทธิ์หน้า r_1, r_2 นั้นเป็นสมาชิกของ K เราจะบอกว่า ϕ นิยามเหนือ K แล้วเขียน

$$r_1(x) = \frac{p(x)}{q(x)}$$

โดยกำหนดให้พหุนาม p, q ไม่มีตัวประกอบร่วมกัน และจะนิยาม degree ของ isogeny ได้ว่า $deg(\phi) = Max\{deg\left(p(x)\right), deg\left(q(x)\right)\}$ และในกรณีที่หากว่าอนุพันธ์อันดับ 1 ของ $r_1 = r_1'(x)$ นั้นไม่เท่ากับ 0 เราจะบอกว่า isogeny ϕ นั้น separable (แยกจากกันได้) [2]

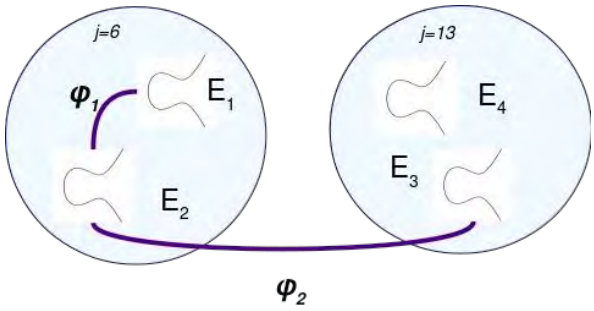
ทำให้ได้นิยามว่าถ้า ϕ แยกจากกันได้แล้ว $deg(\phi) = \#Ker(\phi)$ แต่ถ้าแยกจากกันไม่ได้แล้ว $deg(\phi) > \#Ker(\phi)$ และในกรณีทั่วไปนั้น kernel (Ker) ของ isogeny จะเป็น finite subgroup ของ $E_1(K)$

ตัวอย่างของ isogeny ที่เห็นได้ชัดคือ การคูณด้วยจำนวนเต็ม n บน elliptic curve ที่เราเคยให้นิยามไปแล้วว่า $[n]P = P + P + \cdots + P$ (n ครั้ง) ซึ่งส่ง O ไปยังตัวเอง, เป็น group homomorphism และอธิบายได้ด้วย rational function ที่มาจาก group law โดยที่ตัว kernel ก็จะเป็น $E[n]$ ตามที่ได้เคยนิยามไว้ในส่วนก่อนหน้า

ส่วนสุดท้ายที่จะเกี่ยวกับ isogeny คือ endomorphism ring โดยเราจะบอกว่า endomorphism ring ของ E คือเซตของ isogeny จาก E ที่กลับมายังตัวเอง รวมถึงเราจะมีนิยาม Zero map $0: E \rightarrow E$ ที่อธิบายโดย $0(P) = O$ สำหรับทุกจุด $P \in E$ แล้วเราจะเขียน endomorphism ring ได้ดังนี้

$$End(E) = \{\phi: E \rightarrow E\} \cup \{0\}$$

ส่วนนี้เองเป็นส่วนสุดท้ายที่ทำให้นำไปใช้สร้างโปรโตคอล SIDH ได้ โดยได้มีการนิยามการรวมกันของ isogeny สองอันคือ $(\phi_1 + \phi_2)(P) = \phi_1(P) + \phi_2(P)$



ภาพที่ 2.5 ตัวอย่างของ isogeny ϕ_1 และ ϕ_2 โดยที่ ϕ_1 เป็น isomorphism
ที่มา: Cloudflare Blog - Towards Post-Quantum Cryptography in TLS

2.2.5 Weil pairing

สำหรับจุดบน elliptic curve E ที่นิยามบน finite field K ถ้าเรากำหนดให้ K เป็น field ที่มี characteristic p และ $\overline{K}$ ที่เป็น algebraic closure ของ K และ n ที่เป็นจำนวนเต็มบวก เรามีนิยามของ n-torsion group ว่า

$$E[n](\overline{K}) \cong Z_n \times Z_n$$

เมื่อให้ Z_n แทน cyclic group ที่มี order n เราจะบอกว่า Weil pairing คือ non-degenerate inner product ที่นิยามบนจุดของ $E[n](\overline{K})$ และนิยามได้ดังนี้

Weil pairing บน elliptic curve E ที่นิยามเหนือ field K จะเป็นกลุ่มของการส่งค่า e_n แต่ละสมาชิกที่นิยามเหนือ K กับจำนวนเต็ม n ที่เป็น coprime กับ p (p เป็น characteristic ของ K) โดยจาก [1] จะบอกได้ว่า

$$e_n: E[n] \times E[n] \to \mu_n$$

หรือก็คือเป็นฟังก์ชันสำหรับส่งค่าของจุด P, Q ที่เป็นสมาชิกใน n-torsion group ของ elliptic curve $E[n] = \{P \in E(K): [n]P = O\}$ ไปยังค่าใน K ในรูปแบบของ primitive nth-root of unity โดยที่ $\mu_n = \{x \in K \mid x^n = 1\}$ และเรากำหนดให้ $e_n(P, Q)^n = 1$ (เราอาจจะมอง field $K = F_{p^n}$ ก็ได้เช่นเดียวกัน)

ในส่วนของ Weil pairing เองก็จะมีสมบัติของความเป็น bilinearity, alternating และ non-degenerate ทำให้สำหรับจุด $P, Q \in E[m]$ เราจะได้ว่า

- $e(P_1 + P_2, Q) = e(P_1, Q)\, e(P_2, Q)$
- $e(P, Q_1 + Q_2) = e(P, Q_1)\, e(P, Q_2)$
- $\forall P \in E[m] \setminus \{O\}, \exists Q \in E[m], e(P, Q) \neq 1$
- $e(P, P) = 1$
- $e(P, Q) = e(Q, P)^{-1}$
- $e(P, Q) = 1$ สำหรับทุกค่า Q ก็ต่อเมื่อ $P = O$
- $e(Q, P) = 1$ สำหรับทุกค่า P ก็ต่อเมื่อ $Q = O$
- ถ้าจุด P, Q อยู่ใน $E[mn]$ ค่า n-th power ของ Weil pairing $e_{mn}(P, Q)$ จะหาได้จาก $e_{mn}(P, Q)^n = e_m([n]P, [n]Q)$

ส่วนถัดมาที่จะพูดถึงคือ divisors และ Weil functions เพื่อใช้ในการคำนวณค่าของ Weil pairing e_n ที่เดิมแล้ว Weil pairing ถูกนิยามบน divisor classes ที่มี degree 0 เนื่องจากทุก divisor class บน Curve E จะต้องมีรูปแบบเฉพาะที่เขียนได้ในลักษณะของ $[P] - [O]$ เราเปลี่ยนนิยามนั้นมาใช้กับจุดบน curve ได้ดังนี้

ให้ C เป็น curve และ D เป็น divisor ที่ $supp(D) \cap supp\big(div(f)\big) = \emptyset$ เรานิยามให้

$$f(d) \coloneqq \prod_{P \in C} f(P)^{v_P(D)}$$

และอีกนิยามที่ใช้บอกการคำนวณค่าของ Weil pairing คือ ให้ $n > 1$ เป็นจำนวนเต็ม และ D_1, D_2 เป็น divisor บน elliptic curve E ที่ $nD_1, nD_2 \sim 0$ ทำให้เรามีฟังก์ชัน f_1, f_2 ที่ทำให้ $nD_i = div(f_i)$ สำหรับ $i = 1,2$ เราจึงนิยามการคำนวณค่าของ Weil pairing ได้ดังนี้

$$e_n(D_1, D_2) = \frac{f_1(D_2)}{f_2(D_1)}$$

ค่าของ Weil pairing นี้จะขึ้นกับ divisor classes D_1, D_2 เท่านั้น โดยตัวนิยามนี้ทำให้เราสามารถคิดอัลกอริทึมที่สามารถคำนวณค่าของ Weil pairing ได้ [1]

ด้วยสมบัติของ Weil pairing หลายอย่าง รวมไปถึงการคำนวณค่าของ Weil pairing ที่สามารถทำได้โดยมีอัลกอริทึมที่มีชื่อว่า Miller's Weil pairing algorithm ทำให้มีการนำมาใช้ประยุกต์ในการทำโปรโตคอลที่เกี่ยวข้องกับ elliptic curve เช่นการตรวจสอบ order ของจุดบน elliptic curve การตรวจสอบความ independence ของจุดที่ต่างกัน หรือการเปลี่ยนปัญหา discrete logarithm จาก group หนึ่งไปยังอีก group หนึ่งที่ทำได้ง่ายกว่า เช่นจาก group บน elliptic curve ไปยัง group บน finite field ซึ่ง Weil pairing มีการใช้อย่างต่อเนื่องทั้งในการศึกษาสมบัติของ elliptic curve และการนำไปใช้งานในโปรโตคอลในช่วงแรก แต่ในปัจจุบันนั้นได้มีการพัฒนาให้อยู่ในรูปแบบของการคำนวณ Tate pairing $t(P, Q)$ ที่ใช้งานจริงในโปรโตคอลที่เรากำลังจะพูดถึง โดยที่ยังมีการคงสมบัติในเรื่องของความเป็น bilinear และ non-degenerate เอาไว้แต่จะต่างตรงที่ค่าของ $t(P, P)$ ไม่เท่ากับ 1 ในทุกกรณีซึ่งต่างจาก Weil pairing นั่นเอง ทำให้การใช้งานโดนส่วนมากที่เดิมใช้ Weil pairing นั้นเราสามารถนำ Tate pairing ไปใช้แทนได้ โดยทั่วไปแล้วเรามักจะบอกได้ว่าการคำนวณ Weil pairing ก็จะเหมือนกับการคำนวณ Tate pairing 2 รอบ (รายละเอียดเพิ่มเติมเรื่องความแตกต่างสามารถอ่านได้จาก [18])

2.3 Post Quantum Cryptography

โปรโตคอลที่เราใช้อยู่ในปัจจุบันนี้ เช่น RSA, ECDH ล้วนมีที่มาจากปัญหาทางคณิตศาสตร์ที่พิสูจน์ได้ว่าสามารถหาคำตอบได้ยาก ที่กำหนดว่าจะต้องหาคำตอบได้ใน exponential time แต่เรื่องนั้นเป็นจริงเฉพาะในกรณีที่เราพูดถึง Classical computer ที่ใช้ระบบ bit แบบ 0, 1 แต่ในอนาคตอันใกล้นี้ปัญหาเหล่านั้นจะกลายเป็นปัญหาที่ไม่ได้ยากอีกต่อไป โดยที่ในอนาคตเราอาจจะมีอัลกอริทึมที่สามารถแก้ไขปัญหานั้นได้ใน polynomial time เมื่อเราใช้ Quantum computer ในการแก้ไขปัญหาเหล่านั้น เช่น Shor's Algorithm ในการทำลายความปลอดภัยของระบบ RSA โดยอ้างอิงจากความยากในการแยกตัวประกอบที่มาจากผลคูณของจำนวนเฉพาะที่มีขนาดใหญ่มาก ทำให้ในปัจจุบันองค์กร NIST (National institute of standard and technology) เริ่มค้นหาโปรโตคอลที่สามารถใช้งานคงความปลอดภัยได้ แม้ว่า Quantum computer จะสามารถสร้างขึ้นมาได้สำเร็จจริงในอนาคต ซึ่งหนึ่งในโปรโตคอลที่กำลังแข่งขันกันอยู่ในตอนนี้ในส่วนของ Public key encryption, Key-establishment คือ Supersingular isogeny Diffie-Hellman key exchange (SIDH)/Supersingular isogeny key encapsulation (SIKE)

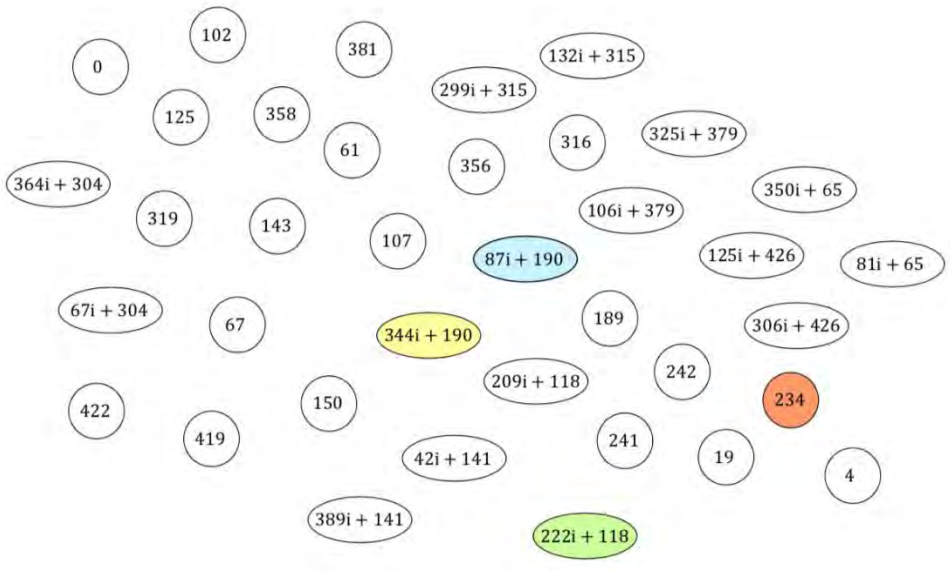
2.4 Supersingular isogeny Diffie-Hellman key exchange (SIDH)

ตัวโปรโตคอลนี้ต้องการที่จะทำสิ่งที่ Diffie-Hellman key exchange เคยทำ กล่าวคือ $Alice$ และ Bob ต้องการสร้างความลับร่วมกัน โดยอาศัยการสร้างกุญแจสาธารณะแล้วแลกเปลี่ยนกุญแจสาธารณะนั้น โดยสิ่งที่จะต่างจาก Diffie-Hellman ทั่วไปก็คือ วิธีการทำงานของโปรโตคอล, ลักษณะของกุญแจสาธารณะ และปัญหาทางคณิตศาสตร์ที่แก้ไขได้ยากอันเป็นพื้นฐานของตัวโปรโตคอลนี้

ตัวโปรโตคอลนี้จะทำงานอยู่บนเซตของ supersingular elliptic curve ที่นิยามบน quadratic extension ของ prime field F_p ที่มีจำนวนเฉพาะ $p = l_A^{e_A} l_B^{e_B} f \pm 1$ ขนาดใหญ่โดยที่ $p \equiv 3 \, mod \, 4$ และโดยส่วนมากเราจะใช้วิธีการบ่งบอก extension field นั้นอย่างง่าย โดยเขียนให้ $F_{p^2} = F_p(i)$ เมื่อให้ $i^2 + 1 = 0$ โดยที่สมาชิกของ F_{p^2} จะเขียนในรูป $a + bi$ ที่ $a, b \in F_p$

จากจำนวนทั้งหมด p^2 ค่าใน F_{p^2} เราจะสนใจ subset ขนาด $\left\lfloor \frac{p}{12} \right\rfloor + z$, $z \in \{0, 1, 2\}$ ที่มีขนาดเท่ากับจำนวนค่าของ supersingular j-invariant ใน F_{p^2}

จากสมบัติที่เคยกล่าวไว้ในส่วนก่อนหน้า elliptic curve E แต่ละอันจะมีค่า j-invariant ได้เพียงค่าเดียวเท่านั้น และ E_1, E_2 จะเป็น isomorphic ต่อกันก็ต่อเมื่อมีค่า j-invariant เท่ากัน ในส่วนนี้จะยกตัวอย่างเมื่อให้ $p = 431$ และ j-invariant ทั้งหมดที่เป็นไปได้ดังในภาพที่ 2.6



ภาพที่ 2.6 เซตของ 37 supersingular j-invariant ที่เป็นไปได้ใน F_{431^2}

ที่มา: [16] หัวข้อที่ 2: The set of supersingular j-invariants Fig. 1

สำหรับ elliptic curve ที่ใช้ใน SIDH เรามักจะเขียนในรูปแบบของ montgomery form แทนที่จะใช้ short weierstrass form คือจะเขียน E ในรูปของ $E_{a,b}: by^2 = x^3 + ax^2 + x$และเรามีลักษณะของ montgomery curve ที่สำคัญคือ เราจะใช้เพียงค่า a ที่เป็นสมาชิกใน F_{p^2} เท่านั้นในการบ่งบอกถึง elliptic curve ที่แตกต่างกัน และเราสามารถคำนวณ j-invariant ของ E ในรูปแบบนี้ได้โดย

$$j(E_{a,b}) = \frac{256(a^2-3)^3}{a^2-4}$$

เห็นได้ว่าค่า j-invariant ของ elliptic curve ในรูปแบบของ montgomery curve นั้นไม่ขึ้นกับค่า b เลย ตัวอย่างเช่นถ้าเรากำหนดให้ $a_1 = 208i + 161,\ a_2 = 172i + 162$ เราจะสามารถคำนวณค่า j-invariant ได้ว่า $j(E_{a_1}) = j(E_{a_2}) = 364i + 304$ แปลว่า curve E_{a_1} เป็น isomorphic กับ E_{a_2} เราจะยกตัวอย่างได้ว่า isomorphism ψ สำหรับ E_{a_1}, E_{a_2} คือ

$$E_{a_1} \to E_{a_2}: (x, y) \to \big((66i + 182)x + (300i + 109),\ (122i + 159)y\big)$$

และ ψ^{-1}

$$E_{a_2} \to E_{a_1}: (x, y) \to \big((156i + 40)x + (304i + 202),\ (419i + 270)y\big)$$

เราสามารถหา $\psi\left(O_{E_{a_1}}\right) = O_{E_{a_2}}$ และ $\psi\left(O_{E_{a_2}}\right) = O_{E_{a_1}}$ ซึ่งตรงกับนิยามที่เคยให้ไว้ในส่วนก่อนหน้า

ในส่วนของวิธีการทำงานของโปรโตคอล SIDH นั้นเราจะสามารถอธิบายได้ดังนี้

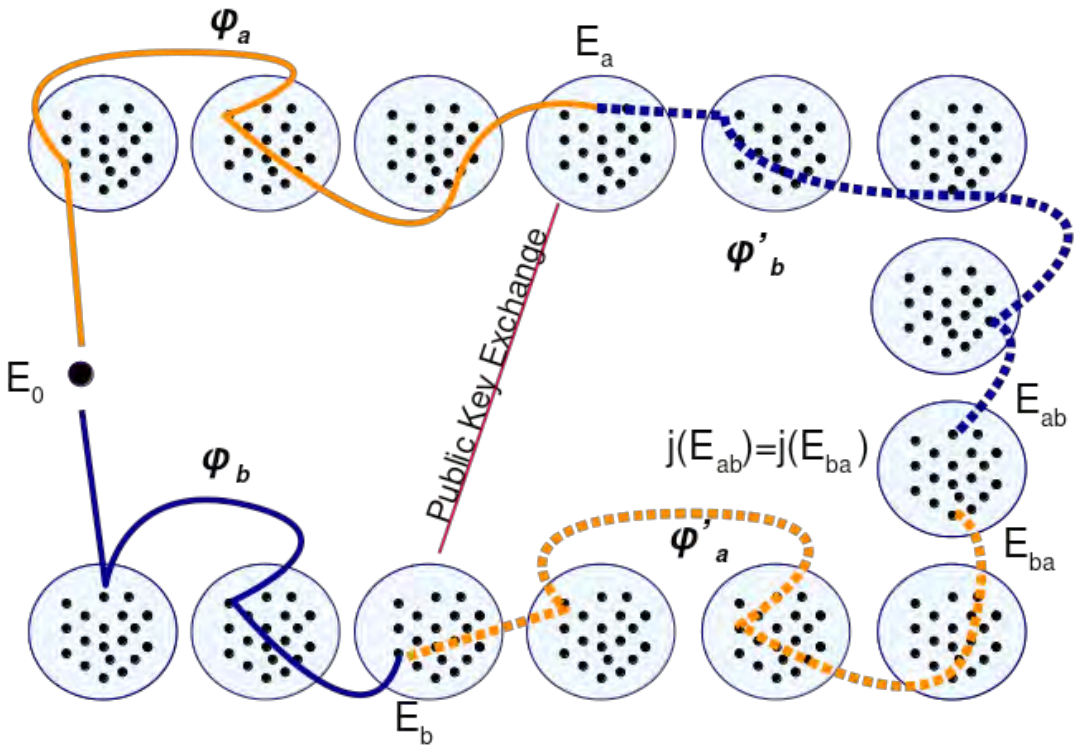
เราจะมี parameter สาธารณะของโปรโตคอล ได้แก่จำนวนเฉพาะ $p = l_A^{e_A} l_B^{e_B} f - 1$ ในที่นี้จะใช้ค่า $l_A = 2, l_B = 3$ ในส่วนของค่า e_A, e_B นั้นขึ้นอยู่กับว่าต้องการความปลอดภัยที่ระดับใด เช่นในงานนี้สนใจที่ความปลอดภัยระดับของจำนวนเฉพาะ p ที่มีความยาว 751 bit มีความปลอดภัยเทียบเท่ากับระบบ AES256 เราจะต้องกำหนด $e_A = 372, e_B = 239$ และมี f เป็น cofactor ของ p โดยในที่นี้จะใช้ $f = 1$

จากนั้นเราจะมี E ที่นิยามเหนือ F_{p^2} ที่มี cardinality $\left(l_A^{e_A} l_b^{e_B} f\right)^2$ และนิยาม $E\left[l_A^{e_A}\right]$ ที่มี $l_A^{e_A}(l_A + 1)$ cyclic subgroup เช่นเดียวกับ $E\left[l_B^{e_B}\right]$ ที่มี $l_B^{e_B - 1}(l_B + 1)$ cyclic subgroup แล้วตัวโปรโตคอลจะมีขั้นตอนการทำงานดังนี้ [17]

- ทั้ง $Alice$ และ Bob เริ่มที่ elliptic curve E
- $Alice$ เลือกกุญแจลับ sk_A แล้วคำนวณหา isogeny ϕ_A ที่เดินตาม isogeny ที่มี degree l_A แล้วไปจบที่ curve E_A ($E/\langle S\rangle$ เมื่อให้ S เป็น kernel ของ isogeny ϕ degree $l_A^{e_A}$)
- Bob เลือกกุญแจลับ sk_B แล้วคำนวณหา isogeny ϕ_B ที่เดินตาม isogeny ที่มี degree l_B แล้วไปจบที่ curve E_B ($E/\langle R\rangle$ เมื่อให้ R เป็น kernel ของ isogeny ϕ degree $l_B^{e_B}$)
  (ถ้าให้ $\phi: E_1 \to E_2$ จะได้ว่า $Kernel(\phi) = \{P \in E_1 : \phi(P) = O\}$)

- $Alice, Bob$ ส่ง E_A, E_B ให้ทุกฝ่ายรับทราบ
- $Alice$ นำ E_B ที่ได้รับจาก Bob มาคำนวณหา isogeny$\phi_A(E_B)$ ซ้ำอีกครั้งหนึ่ง ทำให้ไปจบที่ curve $E_{A,B}(E/\langle S, R\rangle)$
- Bob นำ E_A ที่ได้รับจาก $Alice$ มาคำนวณหา isogeny $\phi_B(E_A)$ ซ้ำอีกครั้งหนึ่ง ทำให้ไปจบที่ curve $E_{B,A}(E/\langle R, S\rangle)$
- $Alice$ คำนวณ j-invariant, $j(E_{A,B}) = ss$ ได้เป็น $shared\ secret$
- Bob คำนวณ j-invariant, $j(E_{B,A}) = ss$ ได้เป็น $shared\ secret$
- จบการทำงานของโปรโตคอล

เราอาจจะเปรียบเทียบกลับไปยังภาพที่ 2.6 ได้ว่าเรากำหนดให้ curve เริ่มต้นนั้นมีค่า j-invariant อยู่ในกรอบสีฟ้า $(87i + 190)$ แล้วทางฝั่ง $Alice$ ทำการคำนวณ isogeny ไปจบที่ curve หนึ่งที่มีค่า j-invariant อยู่ในกรอบสีเหลือง $(344i + 190)$ เช่นเดียวกับ Bob ที่คำนวณ isogeny ไปจบที่ curve หนึ่งที่มีค่า j-invariant อยู่ในกรอบสีเขียว $(222i + 118)$ แล้วหลังจากทั้งสองฝ่ายแลกเปลี่ยนกุญแจสาธารณะ (curve E_A, E_B) ก็ทำการคำนวณ isogeny ต่อไปจาก curve นั้น แล้วไปจบที่ curve หนึ่งซึ่งมีค่า j-invariant อยู่ในกรอบสีส้ม (234) แล้วเราใช้ค่านี้แทนความลับร่วมกันนี้เอง



ภาพที่ 2.7 ตัวอย่างของการเดินทางบน isogeny path ระหว่าง elliptic curve ที่แตกต่างกัน
ที่มา: Cloudflare Blog - Towards Post-Quantum Cryptography in TLS

ถ้าดูตามภาพที่ 2.7 นี้จะเห็นได้ว่าจุดสีดำนั้นแทน elliptic curve ในแต่ละ supersingular j-invariant โดยที่เรามีหลาย curve ที่มีค่าของ j-invariant เท่ากัน ในขณะที่กลุ่มก้อนสีฟ้าคือค่า j-invariant ที่แตกต่างกันแต่ละค่า แม้ว่า $E_{A,B}$ ที่ $Alice$ หามานั้นอาจจะไม่ใช่ curve อันเดียวกันกับที่ Bob หามาได้ คือ $E_{B,A}$ แต่ทั้งสอง curve จะมีค่า j-invariant เท่ากัน แล้วเราจะใช้ค่านี้เองในการใช้เป็น $shared\ secret$ ที่เราต้องการ

ทั้งหมดที่อธิบายนี้เป็นเพียงทฤษฎีที่เสนอกันในช่วงแรกเท่านั้น แต่ในทางปฏิบัติจริงรวมถึงที่กำลังใช้งานอยู่บนโปรโตคอลที่เราสนใจ แทนที่จะส่งข้อมูลของทั้ง curve E_A, E_B ในช่วงของการแลกเปลี่ยนกุญแจสาธารณะที่การคำนวณหา isogeny นั้นทำได้ช้าและเปลือง bandwidth ที่ต้องส่งข้อมูล ทำให้ในส่วนของการใช้งานจริงเราจะเริ่มจาก parameter สาธารณะดังนี้ [17]

- Elliptic curve $E_0: by^2 = x^3 + ax^2 + cx$ (ปัจจุบันเริ่มต้นที่ $a = 6,\ b = 1,\ c = 1$)
- จำนวนเฉพาะ $\mathrm{p} = 2^{372}3^{239} - 1$

- จุดเริ่มต้นของ $Alice$: $P_A = [3^{239}](i+c, \sqrt{f(i+c)})$, $Q_A = [3^{239}](i+c, \sqrt{f(i+c)})$ ซึ่ง $\mathrm{P_A}, \mathrm{Q_A} \in \mathrm{E_0}(\mathrm{F_{p^2}})$ มี order 2^{372} และจุด P_A, Q_A สร้าง basis สำหรับ $\mathrm{E_0}[2^{372}]$ โดยที่ c เป็นจำนวนเต็มบวกที่น้อยที่สุดที่ทำให้ $[2^{372-1}]P_A = (-3 \pm 2\sqrt{2}, 0)$, $[2^{372-1}]Q_A = (0,0)$
- จุดเริ่มต้นของ Bob: $P_B = [2^{372}](c, \sqrt{f(c)})$, $Q_B = [2^{372}](c, \sqrt{f(c)})$ ซึ่งกำหนด $P_B \in E_0(F_{p^2}) \setminus E_0(F_p), Q_B \in E_0(F_p)$ มี order 3^{239} และจุด P_B, Q_B สร้าง basis สำหรับ $\mathrm{E_0}[3^{239}]$ และ $f(c)$ ของจุด P_B เป็น square ใน F_P ในขณะที่ $f(c)$ ของจุด Q_B ไม่เป็น square ใน F_p (a เป็น square ใน F_p ก็ต่อเมื่อ $\exists b \in F_p, b^2 = a$)
- แต่ละฝ่ายจะใช้ ϕ บนจุด P, Q เริ่มต้นของฝ่ายตรงข้ามแทนในการสร้างกุญแจสาธารณะ เช่น
- $Alice$ หา $\phi_A(P_B), \phi_A(Q_B)$ กลายเป็นจุด P, Q ใหม่ แล้วเก็บเฉพาะ x-coordinate ของจุด $P, Q, P-Q$ เป็น x_P, x_Q, x_{P-Q}
- Bob หา $\phi_B(P_A), \phi_B(Q_A)$ กลายเป็นจุด P, Q ใหม่ แล้วเก็บเฉพาะ x-coordinate ของจุด $P, Q, P-Q$ เป็น x_P, x_Q, x_{P-Q}

การส่งเพียง tuple สามตัวนี้ที่เป็นสมาชิกใน F_{p^2} จะสามารถทำให้ดำเนินการทางคณิตศาสตร์ได้เร็วขึ้นโดยการคำนวณแค่บน x-coordinate และหาค่า curve parameter A กลับมาได้ด้วยไม่ต้องแนบมาด้วยในกุญแจสาธารณะ โดยปัจจุบันวิธีนี้เราถือว่าเป็น generic implementation ของ SIDH

2.5 Supersingular isogeny key encapsulation (SIKE)

ปัญหาเดิมของโปรโตคอล SIDH คือต้องทำงานร่วมกับ $Alice, Bob$ ที่ขึ้นตรงกับโปรโตคอล ไม่มีฝ่ายใดฝ่ายหนึ่งที่มีเจตนาแอบแฝง (เช่น Bob แอบทำงานให้ Eve เพื่อต้องการค้นหากุญแจลับของ $Alice$) เช่น การใช้กุญแจลับตัวเดียวกันในการติดต่อทุกรอบจะสามารถทำให้เราสามารถทราบข้อมูลในแต่ละ bit ของกุญแจลับของ $Alice$ ได้ โดยในส่วนนี้จะขอละส่วนของวิธีการที่ใช้ในการโจมตีรูปแบบนั้นไป แต่จะเน้นมาที่ว่าจากการที่ได้มีการเสนอโปรโตคอล SIKE ขึ้นมาแทน SIDH ทำให้ช่วยแก้ปัญหาในด้านนั้นไปได้ส่วนหนึ่ง โดยตัวโปรโตคอลมีวิธีการทำงานดังนี้

เราจะใช้วิธีการของ public key encryption ที่ได้กล่าวถึงไว้ในตอนแรก คือการที่มีฟังก์ชัน $Gen(), Enc(), Dec()$ ที่ใช้ในการสร้างกุญแจ, การเข้ารหัสข้อความ, การถอดรหัสข้อความ และตัวอย่างสำหรับในโปรโตคอล SIKE นั้นมีการทำงานดังนี้ [17]

$Gen(\,)$

- สร้างคู่ของกุญแจลับและกุญแจสาธารณะสำหรับ $Alice$: $(sk_A,\ pk_A)$ โดย pk_A มาจากการคำนวณ isogeny ด้วย sk_A และจุดเริ่มต้นที่กำหนดไว้
- $Alice$ ส่งค่า pk_A ให้ Bob แล้วเก็บค่า sk_A

$Enc(pk_A, m)$

- Bob สร้าง sk_B แล้วคำนวณหา isogeny ด้วย sk_B และจุดเริ่มต้นที่กำหนดไว้ เก็บไว้เป็น c_0
- Bob คำนวณหา j-invariant จาก pk_A ที่ได้รับ และ sk_B ของตนเอง
- Bob hash ค่า j-invariant ด้วยฟังก์ชัน F เก็บไว้เป็นค่า h
- คำนวณ $c_1 = h \oplus m$
- คืนค่าคู่ของ (c_0, c_1)

$Dec\big(sk_A, (c_0, c_1)\big)$

- $Alice$ คำนวณหา j-invariant ด้วย c_0 และ sk_A ของตนเอง
- $Alice$ hash ค่า j-invariant ด้วยฟังก์ชัน F เก็บไว้เป็น h
- $Alice$ คำนวณ $m = h \oplus c_1$
- คืนค่า m กลับมา

และอีกส่วนหนึ่งที่ทำให้ SIKE ทำงานได้คือ key encapsulation mechanism ที่ประกอบด้วย 3 ฟังก์ชันได้แก่ $KeyGen(\,), Encaps(\,), Decaps(\,)$ มีการทำงานดังนี้

$KeyGen(\,)$

- $Alice$ สร้างคู่ของกุญแจลับและกุญแจสาธารณะ sk_A, pk_A โดย pk_A มาจากการคำนวณ isogeny ด้วย sk_A และจุดเริ่มต้นที่กำหนดไว้
- $Alice$ สุ่มค่า s ขึ้นมาโดยการใช้ฟังก์ชัน $PRNG$ (Pseudo random number generator) ขนาด n bit
- $Alice$ ส่งค่า pk_A ให้ Bob แล้วเก็บค่า sk_A, s

$Encaps(pk_A)$

- Bob รับค่า pk_A
- Bob สุ่ม message m ขนาด n bit จากฟังก์ชัน $PRNG$

- Bob คำนวนหาค่า $r = G(m||\, pk_A)$ เมื่อให้ G เป็น hash function และ || แทนการ concatenate กันของข้อความ
- Bob หาค่า c_0, c_1 โดยให้ input คือ pk_A กับ $(m; r)$ แทน message สำหรับฟังก์ชัน $Enc(pk_A, m)$
- Bob คำนวณหาค่า $K = H(m||(c_0, c_1))$ เมื่อให้ H เป็น hash function
- Bob ส่งค่า c_0, c_1 แล้วเก็บค่า K โดยที่ K คือ $shared\ secret$

$Decaps(s, sk_A, pk_A, (c_0, c_1)$

- $Alice$ คำนวณ $m' = Dec\big(sk_A, (c_0, c_1)\big)$
- $Alice$ คำนวณ $r' = G(m||\, pk_A)$
- $Alice$ คำนวณ isogeny ด้วยค่า r' แล้วเก็บเป็น c_0'
- $Alice$ ตรวจสอบค่า c_0' เทียบกับ c_0 ที่ได้รับจาก Bob แล้วเตรียมคำนวณค่า K
- ถ้า $c_0' = c_0, K = H(m'||(c_0, c_1))$
- ถ้า $c_0' \neq c_0, K = H(s||(c_0, c_1))$

การที่เรามีฟังก์ชันในกระบวนการ public key encryption และ key encapsulation mechanism เหล่านี้ขึ้น ทำให้เรามั่นใจได้ว่า $Alice, Bob$ จะต้องได้ $shared\ secret$ ที่ตรงกันอย่างแน่นอนถ้าทั้งสองฝ่ายทำตามโปรโตคอลที่กำหนดไว้ โดยเทคนิคนี้เราเรียกได้ว่าเป็นการเพิ่ม indirect public key validation เข้าไป หรือก็คือเราไม่ได้ตรวจสอบกุญแจสาธารณะโดยตรงว่าทำตามเงื่อนไขที่กำหนดไว้หรือไม่ แต่เราเพิ่มกระบวนการในโปรโตคอลเข้าไป ที่ถ้าหากมีฝ่ายใดฝ่ายหนึ่งไม่ทำตามโปรโตคอลจะไม่ทำให้เกิดความสูญเสียความปลอดภัยเกิดขึ้นในตัวโปรโตคอลนั้น อันเป็นที่มาของ SIKE ที่ใช้อยู่ในปัจจุบัน

2.6 SIDH public key compression

ในส่วนนี้จะพูดถึงการบีบอัดกุญแจสาธารณะบนโปรโตคอล SIDH จากเดิมที่เคยบอกว่า กุญแจสาธารณะของ SIDH นั้นในปัจจุบันจะอยู่ในรูปแบบของ (x_P, x_Q, x_{P-Q}) เมื่อให้ x_P แทน x-coordinate สำหรับจุด P บน elliptic curve ที่เป็นสมาชิกของ F_{p^2} โดยที่สมาชิกใน F_{p^2} จะเขียนในรูปแบบของ $a + bi,\ a, b \in F_p$

เราพบว่าการส่งข้อมูลกุญแจสาธารณะในรูปแบบนี้นั้นเป็นการสิ้นเปลือง bandwidth อย่างมาก และเป็นลักษณะส่วนใหญ่ของโปรโตคอลที่กำลังแข่งขันกันอยู่ในยุคของ Post quantum cryptography ที่โดยทั่วไปแล้วกุญแจสาธารณะนั้นมีขนาดใหญ่ (ใหญ่กว่าราว 5-10 เท่าหากเทียบโปรโตคอลที่ใช้ในปัจจุบัน

เช่น ECDH, RSA ในกรณีอย่างดีที่สุด และในบางโปรโตคอลอาจใหญ่มากกว่าเดิมถึง 50 - 70 เท่า) ด้วยเหตุนี้เองจึงมีการวิจัยเกิดขึ้นที่จะเปลี่ยนลักษณะของกุญแจสาธารณะให้มีขนาดลดลง ทำให้มีขนาดเหลือเพียงประมาณครึ่งหนึ่งของรูปแบบกุญแจสาธารณะทั่วไป ตัวอย่างสำหรับในส่วนนี้จะอธิบายผ่านมุมมองการบีบอัดกุญแจสาธารณะของ $Alice\ pk_A$

เดิมทีแล้วกุญแจสาธารณะของ $Alice$ จะประกอบด้วยค่า $a \in F_{p^2}$ ที่ใช้บ่งบอก elliptic curve $E_a: y^2 = x^3 + ax^2 + x$ ในรูปของ montgomery form ที่นิยามเหนือ F_{p^2} และจุดอีกสองจุดคือ $\phi_A(P_B), \phi_A(Q_B)$ ที่อยู่บน curve E_a และใช้เฉพาะค่าในส่วนของ x-coordinate ที่เป็นสมาชิกใน F_{p^2} เช่นเดียวกัน

สิ่งที่จะเปลี่ยนแปลงไปคือแทนที่จะส่งจุด $\phi_A(P_B), \phi_A(Q_B)$ ให้ส่ง $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ที่เป็นสมาชิกของ $Z/l_B^{e_B}Z$ แทน และใช้ในการบ่งบอกจุด $\phi_A(P_B), \phi_A(Q_B)$ ได้ โดยค่า α, β นี้จะต้องขึ้นกับ basis point R_1, R_2 สำหรับ $E\left[l_B^{e_B}\right]$ โดยทางฝ่าย Bob ก็ต้องสามารถหาค่า R_1, R_2 นี้ได้เช่นเดียวกันโดยใช้ข้อมูลที่ได้รับจาก $Alice$ โดยที่ตัวจุด R_1, R_2 นี้ในขณะที่เราสร้างก็ต้องมีความ independence กันด้วยเราถึงจะเลือกนำมาใช้ และกำหนดให้ $\left[l_A^{e_A}\right]R_1, \left[l_A^{e_A}\right]R_2$ ต้องมี order ของจุดคือ $l_B^{e_B}$
ในส่วนของวิธีการจะที่สร้าง basis R_1, R_2 ในส่วนนี้จะเป็นตัวอย่างสำหรับ $E\left[l_A^{e_A}\right]$ เราทำได้โดย [6]

- สุ่มจุด $P \in E\left(F_{p^2}\right)$
- คำนวณ $P' = \left[l_B^{e_B}\right]P$ และเราต้องการให้ order ของจุด P' หาร $l_B^{e_B}$ ลงตัว
- ตรวจสอบ order ของจุด P' โดยการตรวจสอบว่า $\left[l_A^{e_A}\right]P' = O$ หรือไม่
- ถ้าตรวจสอบแล้วผ่านแปลว่าเราได้ R_1 ที่ต้องการแล้วคือ P'
- ทำแบบนี้เช่นเดียวกันโดยเลือกจุดใหม่เป็น Q แล้วหา Q'
- ตรวจสอบความ independence ของ R_1 กับ Q' ด้วยการใช้ Weil pairing ของ R_1, Q' ถ้าเราคำนวณค่า $e(R_1, Q') \neq 1$ แปลว่าเราหา Q' ที่ต้องการเจอแล้วก็ให้ใช้ $R_2 = Q'$

เมื่อเราได้ R_1, R_2 มาแล้วสิ่งที่ต้องทำถัดมาคือการแก้ปัญหา discrete logarithm ด้วยอัลกอริทึมของ Pohlig-Hellman เพื่อหาค่า $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ โดยใช้ Weil pairing มาช่วยดังนี้
ตัวอย่างการคำนวณค่า α_P, β_P ของ R_1, R_2 สำหรับจุด $P: \psi_A\left(\phi_A(P_B)\right) = \alpha_P R_1 + \beta_P R_2$

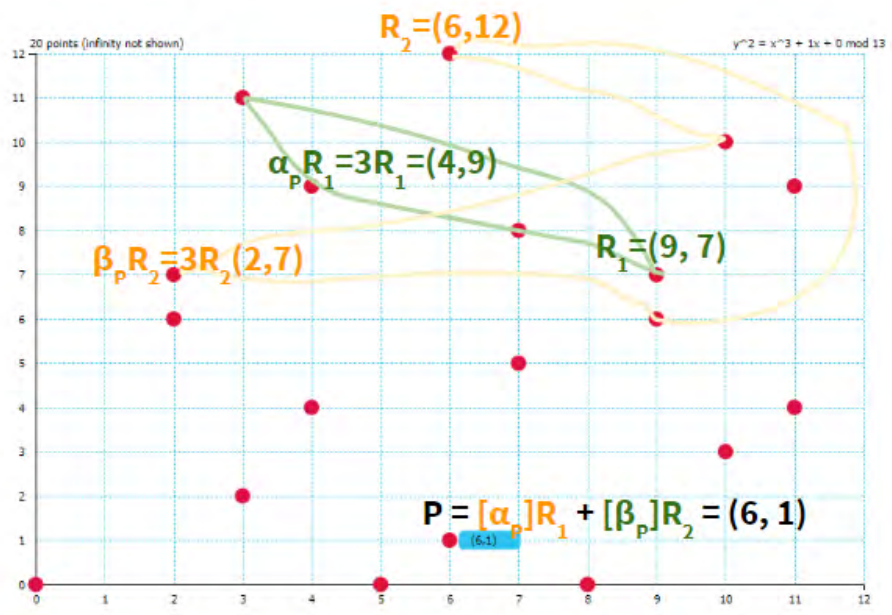
$$\begin{aligned} e\left(R_1, \psi_A\left(\phi_A(P_B)\right)\right) &= e(R_1, \alpha_P R_1 + \beta_P R_2) \\ &= e(R_1, \alpha_P R_1)\, e(R_1, \beta_P R_2) \\ &= e(R_1, R_1)^{\alpha_P}\, e(R_1, R_2)^{\beta_P} \\ &= e(R_1, R_2)^{\beta_P} \end{aligned}$$

ทำให้เราหาค่า β_P มาได้ จากนั้นเราก็นำไปหา α_P ต่อโดยการหา

$$e\left(R_2, \psi_A\left(\phi_A(P_B)\right) - \beta_P R_2\right) = e(R_2, \alpha_P R_1) = e(R_2, R_1)^{\alpha_P}$$

อย่างไรก็ตามวิธีที่ได้กล่าวไว้ข้างต้นนี้เป็นเพียงแค่ naive approach ที่เสนอมาในงานครั้งแรกสุด นั้นมีการใช้เวลาในการสร้าง basis ที่มาก รวมไปถึงการแก้ไขปัญหา discrete logarithm เพื่อหา α, β ก็ทำได้ช้าเช่นเดียวกัน โดยในปัจจุบันนี้มีวิธีสร้าง basis R_1, R_2 สำหรับ $E[2^{e_A}]$ ที่เร็วขึ้นมาก (จะอธิบายในส่วนถัดไป) ในขณะที่ basis R_1, R_2 สำหรับ $E[3^{e_B}]$ ยังใช้เทคนิคประมาณนี้อยู่สำหรับโปรโตคอลที่วิจัยในตอนนี้ (มีการพัฒนาขึ้นมาเล็กน้อยโดยการเพิ่มข้อมูลเพื่อชี้แนะแนวทางการหา basis ให้เร็วขึ้นได้) แต่เนื่องจากสิ่งที่เราสนใจคือลักษณะทาง algebraic structure ของ R_1, R_2 ยังคงเหมือนกันอยู่ในทั้งวิธีเดิมและวิธีใหม่ เช่นเดียวกับการแก้ไขปัญหา discrete logarithm ที่ในตอนนี้ใช้ Tate pairing แทน Weil pairing ทำให้คำนวณได้เร็วยิ่งขึ้นกว่าเดิม แต่ยังใช้หลักการพื้นฐานคล้ายกันในเรื่องของความเกี่ยวข้องของตัว basis R_1, R_2 กับค่า α, β

ทำให้ตอนนี้เราจะได้ว่าจากเดิมในการส่งจุด P, Q เราจะส่ง $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ไปแทน แล้วเราจะสามารถเขียนได้ว่า $P = \alpha_P R_1 + \beta_P R_2,\ Q = \alpha_Q R_1 + \beta_Q R_2$



ภาพที่ 2.8 ตัวอย่างของจุด $P = \alpha_P R_1 + \beta_P R_2$ ในการบีบอัดกุญแจสาธารณะ

ที่มา: https://graui.de/code/elliptic2/

โดยที่ในคราวนี้จะต้องส่ง Curve parameter A เพิ่มไปด้วยเพราะมีผลในการสร้าง basis เนื่องจากเราไม่ต้องการคำนวณ α, β สำหรับจุด $P - Q$ ซึ่งเดิมมีการใช้ในรูปแบบของ generic implementation

แลกกับการที่ไม่ต้องส่งค่า A เพราะการแลกเปลี่ยนทางด้านประสิทธิภาพเทียบกับเวลาแล้วไม่คุ้มค่าในรูปแบบนี้

อีกเรื่องหนึ่งที่จะพูดถึงคือเราสามารถลดขนาดของกุญแจสาธารณะลงไปได้อีก [9] เนื่องจากเดิมเป้าหมายเราคือต้องการคำนวณ $\langle P + lmQ \rangle$ สำหรับ $t \in \{2,3\}$ และกุญแจลับคือ m เราสังเกตลักษณะเพิ่มขึ้นมาได้ว่า

สำหรับจุด P ที่มี order $n \in \{2^{e_A}, 3^{e_B}\}$ ดังนั้นเราจะได้ว่า $\alpha_P \in Z_n^*$ หรือ $\beta_P \in Z_n^*$ จากนั้นเราอาศัยสมบัติของ subgroup ที่สร้างมาจากเซตเพิ่มเติมได้ว่า

$$\langle P + lmQ \rangle = \langle \alpha_P^{-1} P + lm\alpha_P^{-1} Q \rangle \; if \; \alpha_P \in Z_n^*$$

หรือ

$$\langle P + lmQ \rangle = \langle \beta_P^{-1} P + lm\beta_P^{-1} Q \rangle \; if \; \beta_P \in Z_n^*$$

นั่นก็คือเราไม่จำเป็นต้องคำนวณจุด P, Q กลับมาเพื่อการคำนวณหา $\langle P + lmQ \rangle$ แต่เราคำนวณเพียงแค่

$$(\alpha_P^{-1} P, \alpha_P^{-1} Q) = \left(R_1 + \alpha_P^{-1}\beta_P R_2, \alpha_P^{-1}\alpha_Q R_1 + \alpha_P^{-1}\beta_Q R_2\right)$$

หรือ

$$(\beta_P^{-1} P, \beta_P^{-1} Q) = \left(\beta_P^{-1}\alpha_P R_1 + R_2, \beta_P^{-1}\alpha_Q R_1 + \beta_P^{-1}\beta_Q R_2\right)$$

ทำให้เราลดกุญแจสาธารณะในส่วนที่ใช้บอกจุด P, Q จากเดิมที่เป็น 4 ค่า เหลือเพียง 3 ค่า ใน Z_n คือ

$$\begin{aligned}(t_1,\ t_2,\ t_3) &= \left(\alpha_P^{-1}\beta_P, \alpha_P^{-1}\alpha_Q, \alpha_P^{-1}\beta_Q\right) if \; \alpha_P \in Z_n^* \\ &= \left(\beta_P^{-1}\alpha_P, \beta_P^{-1}\alpha_Q, \beta_P^{-1}\beta_Q\right) if \; \beta_P \in Z_n^*\end{aligned}$$

แล้วเราเพิ่มข้อมูลไปค่าหนึ่งโดยใช้ $bit \in \{0,1\}$ เพื่อใช้ในการบอกว่า (t_1, t_2, t_3) อยู่ในรูปแบบใดเท่านั้น ทำให้สุดท้ายแล้วเราใช้ค่าในการบอกจุด P, Q เพียง (bit, t_1, t_2, t_3) โดยเรากำหนดให้ $bit \in Z_2, (t_1, t_2, t_3) \in (Z_n)^3$ เมื่อ $n = l^e$

ในส่วนถัดมานี้จะพูดถึงวิธีการสร้าง basis R_1, R_2 ที่ทำได้เร็วขึ้น โดยแลกกับการที่ต้องเพิ่มข้อมูลในชุดกุญแจสาธารณะเข้าไป และมีวิธีการที่แตกต่างกันเล็กน้อยจึงจะอธิบายแยกจากกัน [12]

- การคำนวณ basis R_1, R_2 ของ $E[2^{e_A}]$ ที่ทำโดย Bob ขณะที่จะส่ง pk_B

เราจะใช้วิธีที่เรียกว่า Entangled basis generation โดยการเลือกตารางที่เก็บค่า QNR หรือ QR (Quadratic non residue, Quadratic residue) ที่ประกาศไว้เป็นสาธารณะล่วงหน้าก่อนการเริ่มทำงานของโปรโตคอล

$$v = \frac{1}{1 + ur^2}, v \in F_{p^2}, u = u_0^2, u, u_0 \in F_{p^2} \setminus F_p$$

โดยการเลือกตารางนั้นขึ้นกับว่า A เป็น QR หรือ QNR (เช่นถ้า A เป็น QR ให้เราเลือกค่าในตาราง QNR) และเราจะมีค่า $r \in Z_{256}$ ใช้ในการบอกตำแหน่ง index ภายในตารางนั้นเอง

โดยสุดท้ายแล้วเราจะได้ basis R_1, R_2 ในรูปของ $R_1 = (x, y), R_2 = (u_0 rx, ur^2 y)$ มาพร้อมกันจากค่า r ที่เราจะใช้ส่งไปในกุญแจสาธารณะนั่นเอง

- การคำนวณ basis R_1, R_2 ของ $E[3^{e_B}]$ ที่ทำโดย $Alice$ ขณะที่จะส่ง pk_A

เราจะใช้วิธีที่เรียกว่า shared elligator ที่ใช้ในการสร้างจุดบน elliptic curve ที่ independent ซึ่งกันและกัน ([12] หัวข้อที่ 4: On basis generation for $E[3^{3_B}]$) โดยจะมีตัวแปร shared elligator counter เพื่อช่วยในการให้ฝั่งผู้รับกุญแจสาธารณะสามารถสร้าง basis ได้เร็วขึ้นนั่นเอง ทำให้เราได้ค่า r_1, r_2 ขึ้นมา โดยฝั่งนี้เองก็มีตารางสาธารณะเช่นเดียวกัน (ต่างจากตารางของการสร้าง $E[2^{e_A}]$) ที่มีการเก็บค่า

$$v = \frac{1}{1 + Ur^2}, v \in F_{p^2}, U = 4 + i$$

สำหรับค่า index $r \in Z_{256}$ ที่แตกต่างกัน โดยเราเก็บค่านี้แยกกันไว้สำหรับ basis R_1, R_2

สุดท้ายแล้ว กุญแจสาธารณะในรูปแบบที่ถูกบีบอัดในปัจจุบันเลยเขียนได้ในรูปของ

$pk_B = (bit, t_1, t_2, t_3, A, s, r)$ สำหรับ $E[2^{e_A}]$ (สำหรับ Bob ที่จะส่งไปให้ $Alice$) โดยเราเพิ่ม $s \in \{0, 1\}$ ขึ้นมาเพื่อใช้บอกว่า A เป็น QR หรือ QNR และใช้ค่า r บอกตำแหน่งในตาราง และ

$pk_A = (bit, t_1, t_2, t_3, A, r_1, r_2)$ สำหรับ $E[3^{e_B}]$ (สำหรับ $Alice$ ที่จะส่งไปให้ Bob) โดยที่ r_1, r_2 ใช้บอก elligator counter สำหรับ basis R_1, R_2 แยกจากกัน

ทำให้เราสรุปได้ว่าโดยรวมแล้วกุญแจสาธารณะจะต้องประกอบด้วยข้อมูลที่มีชนิดดังนี้

- Z_2 สำหรับ bit
- $(Z_{l^e})^3$ สำหรับ (t_1, t_2, t_3)
- F_{p^2} สำหรับ A
- $(Z_{256})^2$ สำหรับ (s, r) หรือ (r_1, r_2)

ทั้งหมดนี้ทำให้เราสามารถลดขนาดของกุญแจสาธารณะลงไปได้ราว 41% ถ้าเทียบกับรูปแบบทั่วไปที่ส่งจุดในรูป x-coordinate ของจุด $P, Q, P - Q$ หรือก็คือ x_P, x_Q, x_{P-Q} ที่รวมแล้วใช้ $\left(F_{p^2}\right)^3$ และเป็นรูปแบบของกุญแจสาธารณะที่เราสนใจในงานนี้เอง

บทที่ 3
วิธีการศึกษาค้นคว้า

ในบทนี้จะกล่าวถึงขั้นตอนการตรวจสอบกุญแจสาธารณะนั้นควรจะเกิดขึ้นในจังหวะใดของการแลกเปลี่ยนข้อมูล วิธีการศึกษาการทำงานของตัวโปรโตคอลจริงที่ใช้งานอยู่จริงและทฤษฎีต่างหลายอย่างที่สามารถนำมาใช้ยืนยันได้ว่าตัวกุญแจสาธารณะนั้นถูกส่งมาอย่างถูกต้อง รวมไปถึงวิธีการตั้งค่าประเมินผลลัพธ์ของอัลกอริทึมที่เลือกใช้ในการตรวจสอบกุญแจสาธารณะนั้น

ในส่วน code ของทั้งตัวโปรโตคอล SIKE นั้นจะนำมาพัฒนาต่อจาก [19] ในส่วนของ optimized implementation สำหรับชุดของจำนวนเฉพาะ p751 $(p = 2^{372}3^{239} - 1)$ ในรูปแบบของกุญแจสาธารณะที่ถูกบีบอัด (SIKEp751_compressed) ซึ่งในตอนแรกจะมีทั้งกระบวนการทำงานของ SIDH, SIKE เขียนไว้ ได้แก่กระบวนการทำ key generation, shared secret computation, encapsulation, decapsulation (sidh.c และ sike.c) รวมไปถึงกระบวนการทางคณิตศาสตร์บน elliptic curve และการกำหนดโครงสร้างของข้อมูล elliptic curve หลายส่วนที่ใช้ library GMP ของภาษา C มาช่วยในการคำนวณ (fpx.c) นอกจากนี้เองจะมีส่วนของการประเมินผลรอบการใช้ CPU ของกระบวนการภายในโปรโตคอล SIKE ได้แก่ key generation, encapsulation, decapsulation ไว้แล้วเช่นเดียวกัน (test_sike.c)

ในส่วนของสิ่งที่จะเพิ่มไปในงานนี้จะประกอบไปด้วยกระบวนการตรวจสอบกุญแจสาธารณะ ที่เขียนอยู่ในไฟล์ validation.c และตัว API อยู่ใน header file validation.h สำหรับการเรียกใช้งานกระบวนการเหล่านั้นเอง โดยใน validation ก็จะใช้ส่วนของกระบวนการทางคณิตศาสตร์บน elliptic curve ที่ได้มีการเขียนอยู่ก่อนแล้วมาช่วยในการทำงานด้วยเช่นกัน นอกจากนั้นจะเป็นการเพิ่มกระบวนการทดสอบโปรโตคอลแยกมาในส่วนของ test_sidh.c รวมไปถึงการเพิ่มการทดลองของทั้งระบบ SIKE ให้รองรับการวัดผลเมื่อกุญแจสาธารณะถูกแก้ไขด้วยเพื่อประเมินผลการทำงานว่าควรที่จะเพิ่มการตรวจสอบกุญแจสาธารณะหรือไม่ ซึ่งตัววัดรอบ CPU ในการทดสอบทั่วไปจะใช้แบบเดียวกับที่ได้มีการเขียนอยู่ก่อนหน้าในส่วนของ test_sike.c เพียงแต่ปรับให้เหลือแค่กระบวนการภายใน SIDH เท่านั้น แต่สำหรับส่วนของการทดสอบกุญแจสาธารณะที่ผิดรูปแบบบน SIKE การวัดรอบ CPU นั้นจะใช้เป็นจำนวนรอบทั้งหมดที่ใช้แทน ซึ่งต่างจากเดิมที่เคยมีอยู่ก่อนหน้าในการทดสอบอื่น

สุดท้ายหลังจากเรามีแนวคิดแล้วว่าในงานนี้จะต้องใช้ code จากส่วนใดบ้าง รวมไปถึงการวางแผนแนวทางการทำงานว่าจะต้องเขียน code เองที่ส่วนใดบ้าง เราก็ได้เริ่มกระบวนการศึกษาทางทฤษฎีก่อนว่า

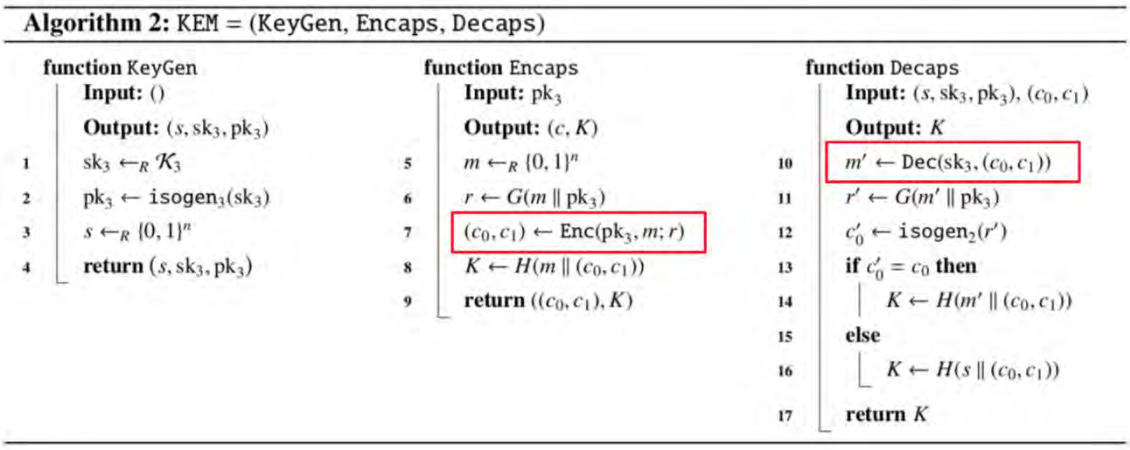
จากขั้นตอนการทำงานของโปรโตคอลนั้นควรจะเพิ่มส่วนของการตรวจสอบกุญแจสาธารณะไว้ที่ขั้นตอนใดนั้นเอง พร้อมกับศึกษาควบคู่ไปกับตัว code จริงที่มีอยู่ว่าตัวทฤษฎี นั้นเทียบเท่ากับส่วนใดภายในตัว code แล้วอาศัยสมบัติทางคณิตศาสตร์ของตัว supersingular elliptic curve, isogeny จากที่ได้กล่าวไว้ในบท 2 รวมไปถึงจาก [5] ที่เสนอการตรวจสอบกุญแจสาธารณะก่อนถูกบีบอัดไว้เป็นแนวทางด้วยว่าจะสามารถปรับกระบวนการและสมบัติเหล่านั้นมาใช้กับกุญแจที่ถูกบีบอัดได้อย่างไร กลายเป็นขั้นตอนการดำเนินงานในบทที่ 3 นี้เอง

3.1 การวิเคราะห์ขั้นตอนการทำงานของตัวโปรโตคอล

สิ่งที่เราต้องการทำคือ การตรวจสอบกุญแจสาธารณะที่แลกเปลี่ยนกันว่ามีคุณสมบัติถูกต้องตามที่ได้กล่าวไว้ในบทที่ 2 ที่ถ้าหากเราพิจารณาการทำงานของตัวโปรโตคอลแล้วจะพบว่าจังหวะที่ต้องตรวจสอบนั้นไม่ตรงกันสำหรับของ $Alice$ และ Bob โดยเราเริ่มมองจากระบบโดยรวมขนาดใหญ่สุดคือส่วนของ public key encapsulation เราจะสามารถสรุปการทำงานอย่างคร่าว ๆ ได้ดังนี้

- $KeyGen$: สร้าง sk_A, pk_A สำหรับ $Alice$ และ s เพื่อเตรียมตัวให้ $Alice$ ทำ indirect key validation
- $Encaps$: Bob รับ pk_A แล้วนำไปสร้าง $shared\ secret$ พร้อมเตรียมการสำหรับการทำ indirect key validation เพื่อให้มั่นใจได้ว่า $Alice$ และ Bob จะต้องได้ $shared\ secret$ อันเดียวกันจากนั้นส่ง c_0, c_1 ให้ $Alice$
- $Decaps$: $Alice$ ใช้ sk_A, pk_A และข้อมูลส่วนอื่นที่ได้รับมาจาก Bob นำมาคำนวณให้ได้ $shared\ secret$ อันเดียวกันกับที่ Bob คำนวณได้

เมื่อเรามองลึกลงไปอีกจะเห็นว่าส่วนที่เกี่ยวของกับกุญแจสาธารณะของฝ่ายตรงข้ามจะเกิดขึ้นที่ $Encaps()$ ที่ Bob ต้องตรวจสอบว่า $Alice\ pk_A$ ที่ส่งมาให้นั้นเป็นไปอย่างถูกต้องหรือไม่ และอีกส่วนหนึ่งคือขั้นตอน $Decaps()$ ที่ $Alice$ ก็ต้องตรวจสอบว่า c_0 ที่ Bob ส่งมาให้นั้นเป็นไปอย่างถูกต้องหรือไม่เช่นเดียวกัน



ภาพที่ 3.1 กระบวนการ Key encapsulation ของโปรโตคอล SIKE

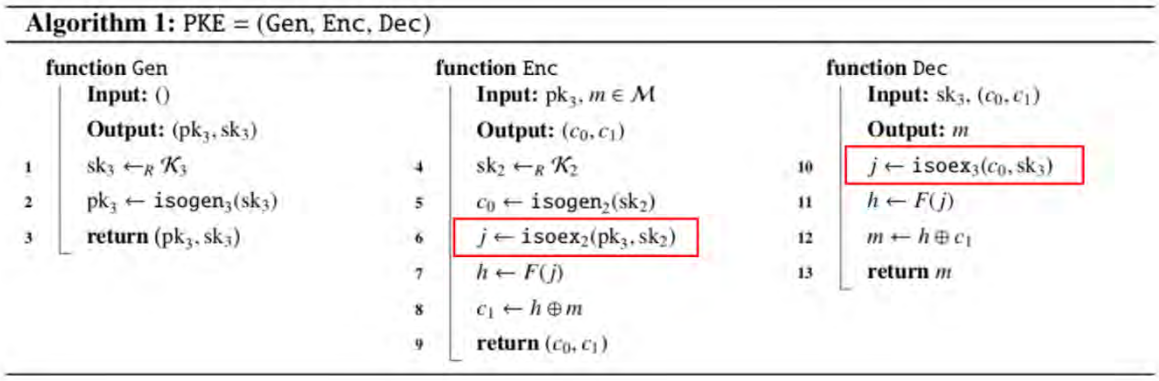
ที่มา: [17] หัวข้อ 1.3.10 - Key encapsulation mechanism

เราจะเห็นได้ว่าในกระบวนการย่อยของฟังก์ชันนั้นส่วนของ $Encaps()$ จะมีการกระทำที่เกี่ยวกับ pk_A อยู่สองครั้งคือการหาค่า r โดย hash function G กับการนำไปเข้าส่วนของ encryption ที่เป็นฟังก์ชันในส่วนของ public key encryption โดยเราสนใจในส่วนหลังมากกว่า

ในขณะที่ฟังก์ชัน $Decaps()$ นั้นไม่ได้มีการนำค่า c_0 ไปทำอะไรก่อนเลย แต่นำไปเข้าในส่วนของฟังก์ชัน decryption ในกระบวนการ public key encryption เช่นเดียวกัน เราจึงเบนความสนใจจากระบบทั้งระบบ SIKE ให้เหลือแค่ส่วนที่เป็น public key encryption ได้ เพราะส่วนที่มีความเกี่ยวข้องกับกุญแจสาธารณะนั้นอยู่แค่ในส่วนของ public key encryption

เราสามารถสรุปกระบวนการทำงานของฟังก์ชันภายในส่วนของ public key encryption ได้ดังนี้

- Gen: สร้าง sk_A, pk_A สำหรับ $Alice$
- Enc: Bob รับ pk_A แล้วนำมาสร้าง $sk_B, c_0 = pk_B$ และคำนวณหา j-invariant จาก sk_B, pk_A พร้อมกับเตรียมค่า c_1 จาก message m ที่ส่งมาให้
- Dec: $Alice$ รับค่า c_0, c_1 คำนวณหา j-invariant จากค่า sk_A, c_0 และจากนั้นนำไปผ่าน hash function F แล้วเก็บเป็นค่า h แล้วคืนค่าข้อความ m กลับมาจากค่า $h \oplus c_1$



ภาพที่ 3.2 กระบวนการ Public key encryption ของโปรโตคอล SIKE

ที่มา: [17] หัวข้อ 1.3.9 - Public-key encryption

(isoex ในส่วนนี้คือฟังก์ชันที่ใช้ในการคำนวณ shared secret)

เราจะเห็นได้ว่าสิ่งที่เกิดขึ้นและเกี่ยวข้องกับการตรวจสอบกุญแจสาธารณะนั้นไม่ได้เกี่ยวกับส่วน public key encryption เช่นเดียวกัน เพราะตัวกุญแจสาธารณะ pk_A, pk_B ไม่ได้เคยถูกปรับแต่งเพื่อให้นำมาใช้ได้เข้ากับกระบวนการของ public key encryption หรือ key encapsulation mechanism เลย ทำให้เราสนแค่ส่วนที่เป็น SIDH ที่ต้องมีฟังก์ชันทำหน้าที่คำนวณหาค่า j-invariant

ดังนั้นสิ่งที่ต้องพิจารณาสุดท้ายแล้วก็คือ ในช่วงของการเข้าไปทำงานในฟังก์ชัน isoex นั้น จะต้องเกิดอะไรขึ้นบ้างกับกุญแจสาธารณะที่แลกเปลี่ยนกัน เพราะว่าเราต้องการปฏิเสธกุญแจสาธารณะที่ไม่ถูกต้องก่อนที่จะไปเริ่มคำนวณกระบวนการในขั้นตอนถัดมา เช่นการหา isogeny, การสร้าง basis R_1, R_2 หรือแม้กระทั่งจุดที่ไม่ได้อยู่บน curve E ที่เกิดขึ้นภายในตัวฟังก์ชันนั้น

เราจะเริ่มจากส่วนของฟังก์ชัน $Encaps()$ หรือก็คือ Bob ที่ได้รับ pk_A จาก $Alice$ โดยดูได้จากภาพที่ 3.3

```c
int crypto_kem_enc(unsigned char *ct, unsigned char *ss, const unsigned char *pk)
{ // SIKE's encapsulation
  // Input:   public key pk              (CRYPTO_PUBLICKEYBYTES bytes)
  // Outputs: shared secret ss           (CRYPTO_BYTES bytes)
  //          ciphertext message ct      (CRYPTO_CIPHERTEXTBYTES = COMPRESSED_CHUNK_CT + MSG_BYTES bytes)
    unsigned char ephemeralsk[SECRETKEY_B_BYTES] = {0};
    unsigned char jinvariant[FP2_ENCODED_BYTES] = {0};
    unsigned char h[MSG_BYTES];
    unsigned char temp[CRYPTO_CIPHERTEXTBYTES + MSG_BYTES] = {0};

    // Generate ephemeralsk <- G(m||pk) mod oB
    randombytes(temp, MSG_BYTES);

    memcpy(&temp[MSG_BYTES], pk, CRYPTO_PUBLICKEYBYTES);

    shake256(ephemeralsk, SECRETKEY_B_BYTES, temp, MSG_BYTES + CRYPTO_PUBLICKEYBYTES);
    FormatPrivKey_B(ephemeralsk);

    // Encrypt
    EphemeralKeyGeneration_Compressed_B(ephemeralsk, ct);

    EphemeralSecretAgreement_Compressed_B(ephemeralsk, pk, jinvariant);

    shake256(h, MSG_BYTES, jinvariant, FP2_ENCODED_BYTES);

    for (int i = 0; i < MSG_BYTES; i++) ct[i + COMPRESSED_CHUNK_CT] = temp[i] ^ h[i];

    // Generate shared secret ss <- H(m||ct)
    memcpy(&temp[MSG_BYTES], ct, CRYPTO_CIPHERTEXTBYTES);
    shake256(ss, CRYPTO_BYTES, temp, CRYPTO_CIPHERTEXTBYTES + MSG_BYTES);

    return 0;
}
```

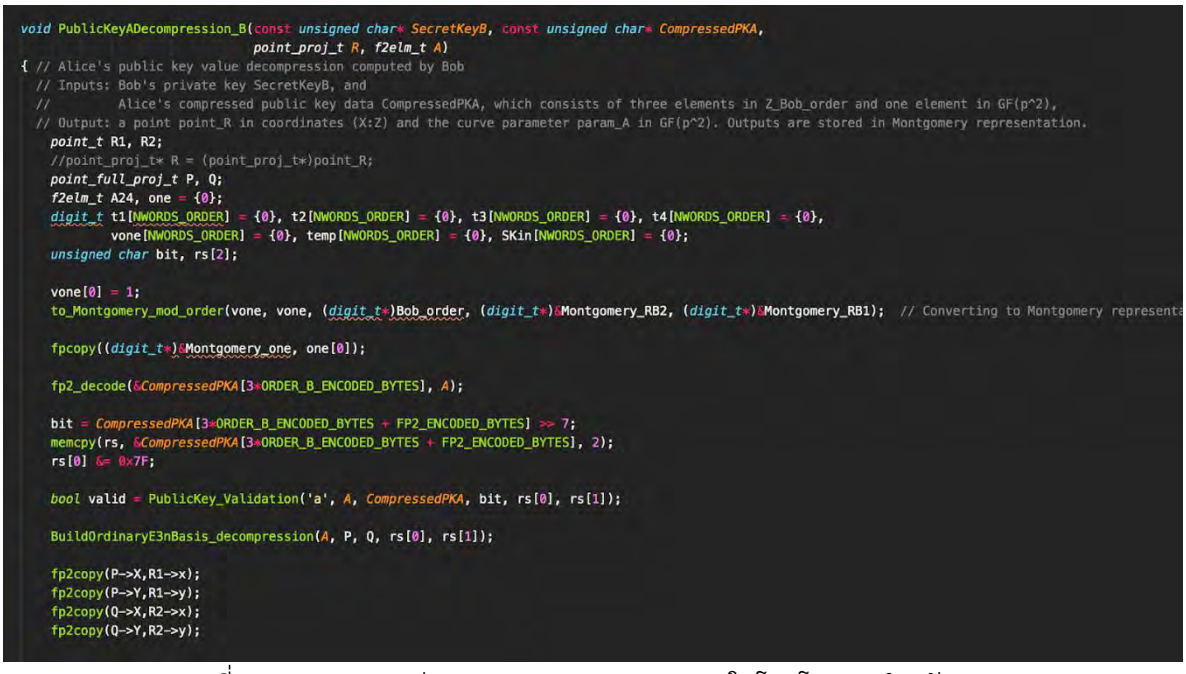
ภาพที่ 3.3 Code ของส่วน Encapsulation ในโปรโตคอล

เห็นได้ว่าฟังก์ชันที่เราสนใจคือ EphemeralSecretAgreement_Compressed_B ที่รับ sk_B กับ pk_A เข้าไป โดยตัวฟังก์ชันนั้นจะสามารถดูได้จากภาพที่ 3.4

```
int EphemeralSecretAgreement_Compressed_B(const unsigned char* PrivateKeyB, const unsigned char* PKA, unsigned char* SharedSecretB)
{ // Bob's ephemeral shared secret computation
  // It produces a shared secret key SharedSecretB using his secret key PrivateKeyB and Alice's decompressed data point_R and param_A
  // Inputs: Bob's PrivateKeyB is an integer in the range [1, oB-1], where oB = 3^OBOB_EXP.
  //         Alice's decompressed data consists of point_R in (X:Z) coordinates and the curve parameter param_A in GF(p^2).
  // Output: a shared secret SharedSecretB that consists of one element in GF(p^2).
    unsigned int i, ii = 0, row, m, index = 0, pts_index[MAX_INT_POINTS_BOB], npts = 0;
    f2elm_t A24plus = {0}, A24minus = {0};
    point_proj_t R, pts[MAX_INT_POINTS_BOB];
    f2elm_t jinv, A, coeff[3];
    f2elm_t param_A = {0};

    if (PrivateKeyB == NULL || SharedSecretB == NULL) {
        return 0;
    }

    PublicKeyADecompression_B(PrivateKeyB, PKA, R, param_A);

    fp2copy((felm_t*)param_A, A);

    fpadd((digit_t*)&Montgomery_one, (digit_t*)&Montgomery_one, A24minus[0]);
    fp2add(A, A24minus, A24plus);
    fp2sub(A, A24minus, A24minus);

    // Traverse tree
    index = 0;
    for (row = 1; row < MAX_Bob; row++) {
        while (index < MAX_Bob-row) {
            fp2copy(R->X, pts[npts]->X);
            fp2copy(R->Z, pts[npts]->Z);
            pts_index[npts++] = index;
            m = strat_Bob[ii++];
            xTPLe(R, R, A24minus, A24plus, (int)m);
            index += m;
        }
```

ภาพที่ 3.4 Code ของส่วน Secret agreement ในโปรโตคอลสำหรับ Bob

เราจะพบอีกฟังก์ชันหนึ่งที่ชื่อ PublicKeyADecompression_B ที่สามารถดูได้จากภาพที่ 3.5 และสังเกตได้ว่าตัว code หลังจากบรรทัดนั้นไปจะเป็นเริ่มเป็นการคำนวณหา isogeny สำหรับ Bob แล้ว ดังนั้นหากจะมีการตรวจสอบ pk_A ก็จะต้องเกิดขึ้นตั้งแต่ก่อนจะเข้าฟังก์ชันนั้นหรือไม่ก็ภายในฟังก์ชันนั้น



```
void PublicKeyADecompression_B(const unsigned char* SecretKeyB, const unsigned char* CompressedPKA,
                               point_proj_t R, f2elm_t A)
{ // Alice's public key value decompression computed by Bob
  // Inputs: Bob's private key SecretKeyB, and
  //         Alice's compressed public key data CompressedPKA, which consists of three elements in Z_Bob_order and one element in GF(p^2),
  // Output: a point point_R in coordinates (X:Z) and the curve parameter param_A in GF(p^2). Outputs are stored in Montgomery representation.
    point_t R1, R2;
    //point_proj_t* R = (point_proj_t*)point_R;
    point_full_proj_t P, Q;
    f2elm_t A24, one = {0};
    digit_t t1[NWORDS_ORDER] = {0}, t2[NWORDS_ORDER] = {0}, t3[NWORDS_ORDER] = {0}, t4[NWORDS_ORDER] = {0},
            vone[NWORDS_ORDER] = {0}, temp[NWORDS_ORDER] = {0}, SKin[NWORDS_ORDER] = {0};
    unsigned char bit, rs[2];

    vone[0] = 1;
    to_Montgomery_mod_order(vone, vone, (digit_t*)Bob_order, (digit_t*)&Montgomery_RB2, (digit_t*)&Montgomery_RB1);  // Converting to Montgomery represents

    fpcopy((digit_t*)&Montgomery_one, one[0]);

    fp2_decode(&CompressedPKA[3*ORDER_B_ENCODED_BYTES], A);

    bit = CompressedPKA[3*ORDER_B_ENCODED_BYTES + FP2_ENCODED_BYTES] >> 7;
    memcpy(rs, &CompressedPKA[3*ORDER_B_ENCODED_BYTES + FP2_ENCODED_BYTES], 2);
    rs[0] &= 0x7F;

    bool valid = PublicKey_Validation('a', A, CompressedPKA, bit, rs[0], rs[1]);

    BuildOrdinaryE3nBasis_decompression(A, P, Q, rs[0], rs[1]);

    fp2copy(P->X,R1->x);
    fp2copy(P->Y,R1->y);
    fp2copy(Q->X,R2->x);
    fp2copy(Q->Y,R2->y);
```

ภาพที่ 3.5 Code ของส่วน Key decompression ในโปรโตคอลสำหรับ Bob

โดยจากการศึกษาในท้ายที่สุด เราพบว่าถ้าจะต้องมีการตรวจสอบกุญแจสาธารณะก็ควรจะต้องตรวจสอบที่ฟังก์ชัน PublicKeyADecompression_B นี้เอง เพราะที่จริงแล้วตัวโปรโตคอลไม่ได้ส่งข้อมูลอย่างที่ได้กล่าวไว้ในข้างต้นเสียทีเดียวในบทที่ 2 แต่จะเพิ่มส่วนของการ encode ให้อยู่ในรูปของ octet string ด้วย และเรามีฟังก์ชันนี้เองที่มีส่วนในการ decode นั้นก่อนที่จะนำเข้าไปในส่วนของการสร้าง basis สำหรับ $E[3^{e_B}]$ ซึ่งมีการใช้ข้อมูลคือค่า curve parameter A, ค่าที่ใช้ในการบอกจุด P, Q ได้แก่ $bit, t_1, t_2, t_3, r_1, r_2$ ตามที่เคยกล่าวไว้ ดังนั้นถ้าจะมีการตรวจกุญแจสาธารณะที่ได้รับมาจาก $Alice\ pk_A$ ก็ต้องเริ่มตรวจตั้งแต่ส่วนนี้ก่อนที่จะไปเข้าฟังก์ชันที่ชื่อ BuildOrdinaryE3nBasis_decompression ในรูปแบบของ

valid = Publickey_Validation('a', A, CompressedPKA, bit, rs[0], rs[1]);

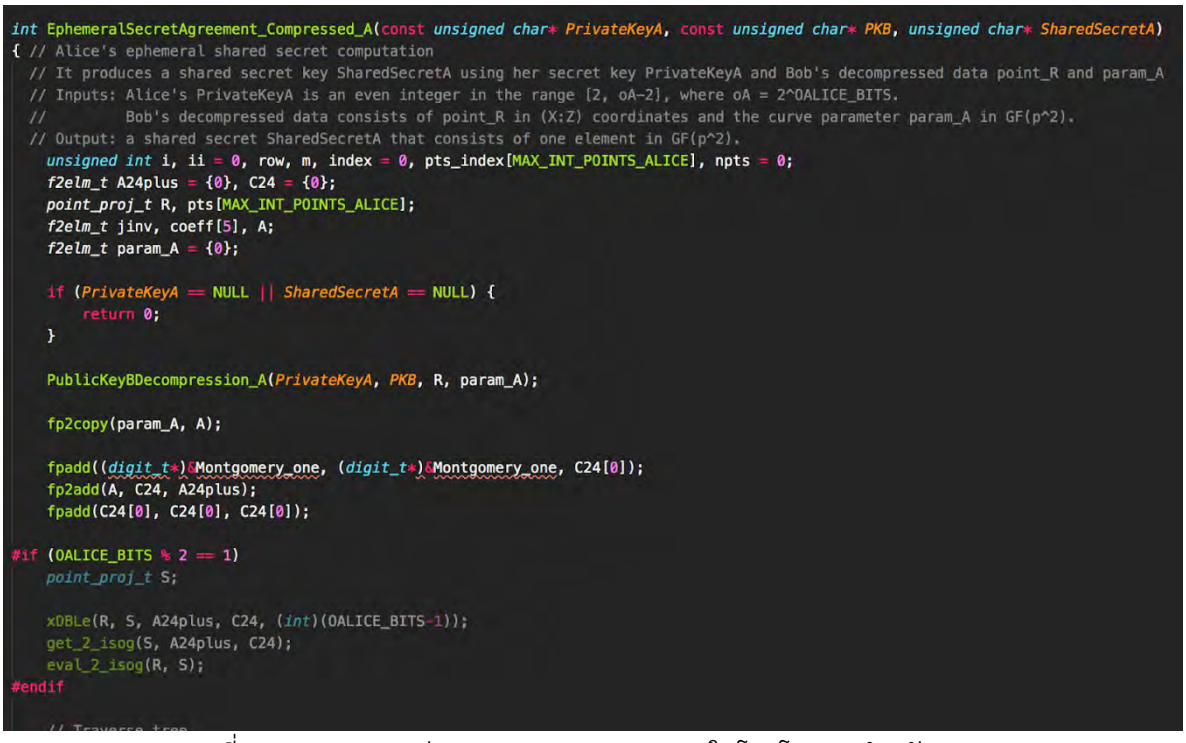
เป็นอันจบส่วนการพิจารณาสำหรับ Bob ที่ต้องการตรวจ pk_A

อีกส่วนหนึ่งที่จะพิจารณาคือในฟังก์ชัน $Decaps()$ ที่ $Alice$ ได้รับ $c_0\ (pk_B)$ โดยในการพิจารณาสุดท้ายแล้วก็จะได้ข้อสรุปที่คล้ายกับของ Bob เราเริ่มจากส่วนฟังก์ชันหลักที่ดูได้จากภาพที่ 3.6

```
int crypto_kem_dec(unsigned char *ss, unsigned char *ct, const unsigned char *sk)
{ // SIKE's decapsulation
  // Input:   secret key sk                       (CRYPTO_SECRETKEYBYTES = MSG_BYTES + SECRETKEY_A_BYTES + CRYPTO_PUBLICKEYBYTES bytes)
  //          compressed ciphertext message ct    (CRYPTO_CIPHERTEXTBYTES = COMPRESSED_CHUNK_CT + MSG_BYTES bytes)
  // Outputs: shared secret ss                    (CRYPTO_BYTES bytes)
    unsigned char ephemeralsk_[SECRETKEY_B_BYTES] = {0};
    unsigned char jinvariant_[FP2_ENCODED_BYTES] = {0}, h_[MSG_BYTES];
    unsigned char c0_comp_[COMPRESSED_CHUNK_CT] = {0};
    unsigned char temp[CRYPTO_CIPHERTEXTBYTES + MSG_BYTES] = {0};

    // Decrypt
    EphemeralSecretAgreement_Compressed_A(sk + MSG_BYTES, ct, jinvariant_);
    shake256(h_, MSG_BYTES, jinvariant_, FP2_ENCODED_BYTES);

    for (int i = 0; i < MSG_BYTES; i++) {
        temp[i] = ct[i + COMPRESSED_CHUNK_CT] ^ h_[i];
    }

    // Generate ephemeralsk_ <- G(m||pk) mod oB
    memcpy(&temp[MSG_BYTES], &sk[MSG_BYTES + SECRETKEY_A_BYTES], CRYPTO_PUBLICKEYBYTES);
    shake256(ephemeralsk_, SECRETKEY_B_BYTES, temp, MSG_BYTES + CRYPTO_PUBLICKEYBYTES);
    FormatPrivKey_B(ephemeralsk_);

    // Generate shared secret ss <- H(m||ct) or output ss <- H(s||ct)
    EphemeralKeyGeneration_Compressed_B(ephemeralsk_, c0_comp_);

    if (memcmp(c0_comp_, ct, COMPRESSED_CHUNK_CT) != 0) {
        printf("\n COMPARISON FAILED!!!\n\n");
        memcpy(temp, sk, MSG_BYTES);
    }

    memcpy(&temp[MSG_BYTES], ct, CRYPTO_CIPHERTEXTBYTES);
    shake256(ss, CRYPTO_BYTES, temp, CRYPTO_CIPHERTEXTBYTES + MSG_BYTES);

    return 0;
}
```

ภาพที่ 3.6 Code ของส่วน Decapsulation ในโปรโตคอล

ฟังก์ชันที่เราจะสนใจก็คือการที่ $Alice$ ได้รับ c_0 หรือก็คือ pk_B ของ Bob ในส่วนของฟังก์ชัน EphemeralSecretAgreement_Compressed_A ที่จะใช้ในการคำนวณ j-invariant ดังภาพที่ 3.7



```
int EphemeralSecretAgreement_Compressed_A(const unsigned char* PrivateKeyA, const unsigned char* PKB, unsigned char* SharedSecretA)
{ // Alice's ephemeral shared secret computation
  // It produces a shared secret key SharedSecretA using her secret key PrivateKeyA and Bob's decompressed data point_R and param_A
  // Inputs: Alice's PrivateKeyA is an even integer in the range [2, oA-2], where oA = 2^OALICE_BITS.
  //         Bob's decompressed data consists of point_R in (X:Z) coordinates and the curve parameter param_A in GF(p^2).
  // Output: a shared secret SharedSecretA that consists of one element in GF(p^2).
    unsigned int i, ii = 0, row, m, index = 0, pts_index[MAX_INT_POINTS_ALICE], npts = 0;
    f2elm_t A24plus = {0}, C24 = {0};
    point_proj_t R, pts[MAX_INT_POINTS_ALICE];
    f2elm_t jinv, coeff[5], A;
    f2elm_t param_A = {0};

    if (PrivateKeyA == NULL || SharedSecretA == NULL) {
        return 0;
    }

    PublicKeyBDecompression_A(PrivateKeyA, PKB, R, param_A);

    fp2copy(param_A, A);

    fpadd((digit_t*)&Montgomery_one, (digit_t*)&Montgomery_one, C24[0]);
    fp2add(A, C24, A24plus);
    fpadd(C24[0], C24[0], C24[0]);

#if (OALICE_BITS % 2 == 1)
    point_proj_t S;

    xDBLe(R, S, A24plus, C24, (int)(OALICE_BITS-1));
    get_2_isog(S, A24plus, C24);
    eval_2_isog(R, S);
#endif

    // Traverse tree
```

ภาพที่ 3.7 Code ของส่วน Secret agreement ในโปรโตคอลสำหรับ $Alice$

เราก็จะพบรูปแบบของฟังก์ชันที่คล้ายกันกับกรณีที่แล้วคือมี PublicKeyBDecompression_A ในภาพที่ 3.8 ที่รับ sk_A และ pk_B โดยตัว code หลังจากฟังก์ชันนั้นก็จะเป็นการคำนวณหา isogeny สำหรับ $Alice$ แล้ว ดังนั้นถ้าจะต้องมีการตรวจสอบกุญแจสาธารณะก็จะต้องเกิดที่ภายในฟังก์ชันนั้น

```
void PublicKeyBDecompression_A(const unsigned char* SecretKeyA, const unsigned char* CompressedPKB,
                               point_proj_t R, f2elm_t A)
{ // Bob's public key value decompression computed by Alice
  // Inputs: Alice's private key SecretKeyA, and
  //         Bob's compressed public key data CompressedPKB, which consists of three elements in Z_orderA and one element in GF(p^2).
  // Output: a point point_R in coordinates (X:Z) and the curve parameter param_A in GF(p^2). Outputs are stored in Montgomery representation.
    point_t S1 = {0}, S2 = {0};
    point_full_proj_t P = {0};
    //point_proj_t* R = (point_proj_t*)point_R;
    f2elm_t A24 = {0}, one = {0};
    digit_t tmp1[2*NWORDS_ORDER] = {0}, tmp2[2*NWORDS_ORDER] = {0}, vone[2*NWORDS_ORDER] = {0};
    digit_t SKin[NWORDS_ORDER] = {0}, comp_temp[NWORDS_ORDER] = {0};
    uint64_t mask = (digit_t)(-1);
    unsigned char bit, isASqr_r[2];

    mask >>= (MAXBITS_ORDER - OALICE_BITS);
    vone[0] = 1;
    fpcopy((digit_t*)&Montgomery_one, one[0]);

    fp2_decode(&CompressedPKB[3*ORDER_A_ENCODED_BYTES], A);

    bit = CompressedPKB[3*ORDER_A_ENCODED_BYTES + FP2_ENCODED_BYTES] >> 7;
    isASqr_r[0] = CompressedPKB[3*ORDER_A_ENCODED_BYTES + FP2_ENCODED_BYTES] & 0x1;
    isASqr_r[1] = CompressedPKB[3*ORDER_A_ENCODED_BYTES + FP2_ENCODED_BYTES + 1];

    bool valid = PublicKey_Validation('b', A, CompressedPKB, bit, isASqr_r[0], isASqr_r[1]);

    get_2_torsion_entangled_basis_decompression(A, S1, S2, isASqr_r[0], isASqr_r[1]);

    fp2add(A, one, A24);
    fp2add(A24, one, A24);
    fp2div2(A24, A24);
    fp2div2(A24, A24);
```

ภาพที่ 3.8 Code ของส่วน Key decompression ในโปรโตคอลสำหรับ $Alice$

ในฟังก์ชันนี้เราก็พบขั้นตอนการทำงานที่คล้ายกันกับของกรณีที่แล้ว คือมีการ decode ค่ากลับมาก่อนให้อยู่ในรูปของกุญแจสาธารณะทั่วไป ก่อนที่จะนำไปสร้าง basis R_1, R_2 ด้วยเหตุนี้เองเราจึงต้องมีการตรวจสอบกุญแจสสาธารณะเกิดขึ้นที่ขั้นตอนนี้ก่อนที่จะไปเข้าฟังก์ชันที่ชื่อว่า get_2_torsion_entangled_basis_decompression ในรูปแบบของ

valid = PublicKey_Validation('b', A, CompressedPKB, bit, isASqr_r[0], isASqr_r[1]);

เป็นอันจบส่วนการพิจารณาสำหรับ $Alice$ ที่ต้องตรวจ pk_B และเป็นอันจบการพิจารณาในส่วนของการวิเคราะห์การทำงานของตัวโปรโตคอล

3.2 การออกแบบวิธีการทดสอบ

ตัวซอฟต์แวร์ดั้งเดิมนั้นมีส่วนที่ช่วยในการตรวจสอบทั้งระบบการทำ key encapsulation มาให้ในรูปแบบของ test_sike ที่วัดว่า $KeyGen(), Encaps(), Decaps()$ นั้นใช้จำนวนรอบ CPU ไปเท่าไหร่ในแต่ละฟังก์ชัน รวมถึงการตรวจสอบว่าโปรโตคอลทำงานได้อย่างถูกต้องด้วยเช่นเดียวกัน แต่เนื่องจากงานของเรานั้นแทบจะไม่ได้มีส่วนเกี่ยวข้องกับส่วน key encapsulation, public key encryption เลย แต่อยู่ในส่วนของกระบวนการบน SIDH อันได้แก่การคำนวณค่า j-invariant หรือที่เรียกว่า shared secret computation ดังนั้นจึงเลือกที่จะประเมินผลการทำงานในส่วนนั้นเสียมากกว่าที่จะประเมินผลโดยรวมทั้ง

ระบบ เพื่อการนี้เองเลยได้มีการเขียนวิธีการประเมินผลการทำงานของส่วน SIDH โดยเฉพาะดังนี้ โดยนำไปสร้างเป็นไฟล์ใหม่ที่ชื่อว่า test_sidh ดังภาพที่ 3.9 และภาพที่ 3.10

cryptotest_kex()

```c
int cryptotest_kex() {
    unsigned int i;
    unsigned char PrivateKeyA[SIDH_SECRETKEYBYTES], PrivateKeyB[SIDH_SECRETKEYBYTES];
    unsigned char PublicKeyA[SIDH_PUBLICKEYBYTES], PublicKeyB[SIDH_PUBLICKEYBYTES];
    unsigned char SharedSecretA[SIDH_BYTES], SharedSecretB[SIDH_BYTES];
    bool passed = true;

    printf("\n\nTESTING EPHEMERAL ISOGENY-BASED KEY EXCHANGE SYSTEM %s\n", SCHEME_NAME);
    printf("--------------------------------------------------------------------------------------------------------\n\n");

    for (i = 0; i < TEST_LOOPS; i++)
    {
        random_mod_order_A(PrivateKeyA);
        random_mod_order_B(PrivateKeyB);

        EphemeralKeyGeneration_A(PrivateKeyA, PublicKeyA);                  // Get some value as Alice's secret key and compute Alice's public k
        EphemeralKeyGeneration_B(PrivateKeyB, PublicKeyB);                  // Get some value as Bob's secret key and compute Bob's public key
        EphemeralSecretAgreement_A(PrivateKeyA, PublicKeyB, SharedSecretA); // Alice computes her shared secret using Bob's public key
        EphemeralSecretAgreement_B(PrivateKeyB, PublicKeyA, SharedSecretB); // Bob computes his shared secret using Alice's public key

        if (memcmp(SharedSecretA, SharedSecretB, SIDH_BYTES) != 0) {
            passed = false;
            break;
        }
    }

    if (passed == true) {
        printf("  Key exchange tests ..................................... PASSED\n");
        return PASSED;
    }
    else {
        printf("  Key exchange tests ... FAILED\n");
        return FAILED;
    }
}
```

ภาพที่ 3.9 Code ของการตรวจสอบการทำงานของโปรโตคอลในส่วน SIDH

เพื่อใช้ในการตรวจสอบความถูกต้องของระบบ SIDH โดยตรวจสอบว่า $\boldsymbol{shared\ secret}$ ที่สร้างมาจากของ $\boldsymbol{Alice, Bob}$ มีความตรงกัน

cryptorun_kex()

```
int cryptorun_kex() {
    unsigned int n;
    unsigned char PrivateKeyA[SIDH_SECRETKEYBYTES], PrivateKeyB[SIDH_SECRETKEYBYTES];
    unsigned char PublicKeyA[SIDH_PUBLICKEYBYTES], PublicKeyB[SIDH_PUBLICKEYBYTES];
    unsigned char SharedSecretA[SIDH_BYTES], SharedSecretB[SIDH_BYTES];
    unsigned long long cycles, cycles1, cycles2;
    f2elm_t param_A = {0}, param_R;
    point_proj_t R;

    printf("---- BENCHMARKING %s average cycles over %d runs\n", SCHEME_NAME, BENCH_LOOPS);
    random_mod_order_A(PrivateKeyA);
    random_mod_order_B(PrivateKeyB);

    // Benchmarking Alice's key generation
    cycles = 0;
    for (n = 0; n < BENCH_LOOPS; n++)
    {
        cycles1 = cpucycles();
        EphemeralKeyGeneration_A(PrivateKeyA, PublicKeyA);
        cycles2 = cpucycles();
        cycles = cycles+(cycles2-cycles1);
    }
    printf("  Alice's key generation runs in ..............................  %10lld ", cycles/BENCH_LOOPS); print_unit;
    printf("\n");

    // Benchmarking Bob's key generation
    cycles = 0;
    for (n = 0; n < BENCH_LOOPS; n++)
    {
        cycles1 = cpucycles();
        EphemeralKeyGeneration_B(PrivateKeyB, PublicKeyB);
        cycles2 = cpucycles();
        cycles = cycles+(cycles2-cycles1);
    }
    printf("  Bob's key generation runs in ................................  %10lld ", cycles/BENCH_LOOPS); print_unit;
    printf("\n");

    // Benchmarking Alice's shared key computation
    cycles = 0;
    for (n = 0; n < BENCH_LOOPS; n++)
    {
        cycles1 = cpucycles();
        EphemeralSecretAgreement_A(PrivateKeyA, PublicKeyB, SharedSecretA);
        cycles2 = cpucycles();
        cycles = cycles+(cycles2-cycles1);
    }
    printf("  Alice's shared key computation runs in .......................  %10lld ", cycles/BENCH_LOOPS); print_unit;
    printf("\n");

    // Benchmarking Bob's shared key computation
    cycles = 0;
    for (n = 0; n < BENCH_LOOPS; n++)
    {
        cycles1 = cpucycles();
        EphemeralSecretAgreement_B(PrivateKeyB, PublicKeyA, SharedSecretB);
        cycles2 = cpucycles();
        cycles = cycles+(cycles2-cycles1);
    }
    printf("  Bob's shared key computation runs in .........................  %10lld ", cycles/BENCH_LOOPS); print_unit;
    printf("\n");

    return PASSED;
}
```

ภาพที่ 3.10 Code ของการตรวจสอบประสิทธิภาพของโปรโตคอลในส่วน SIDH

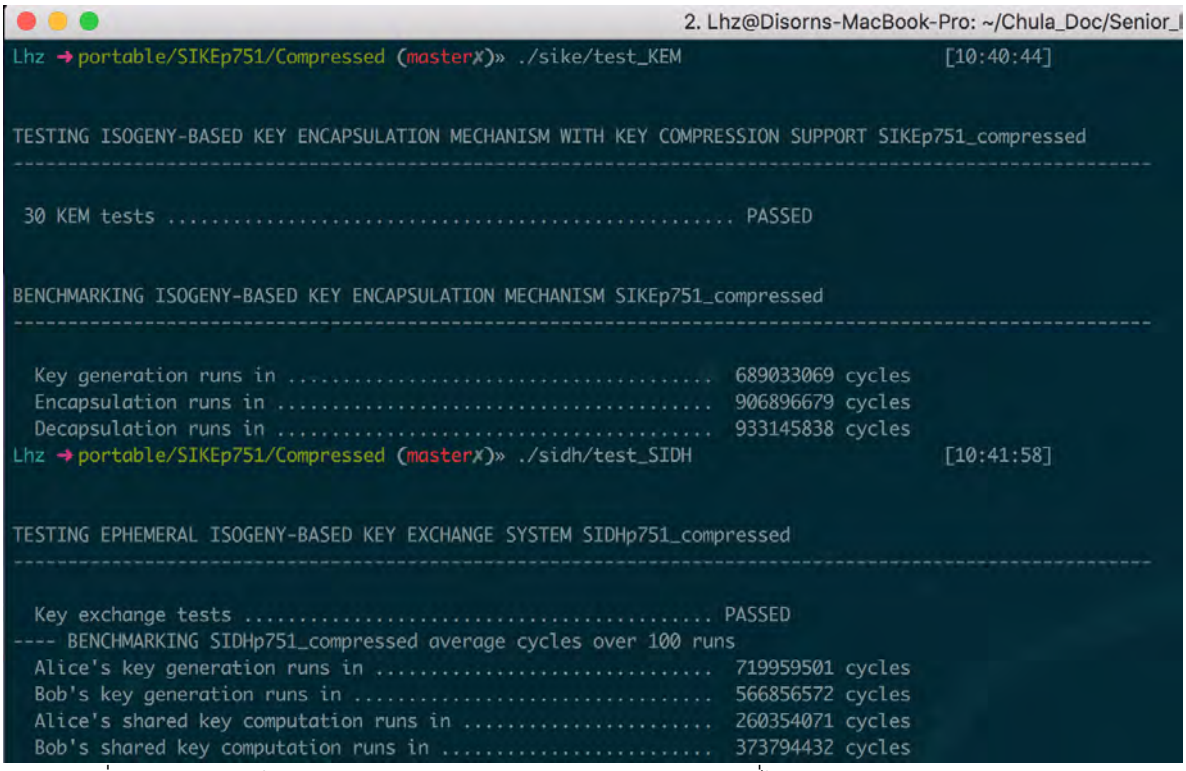
เพื่อใช้ในการวัดรอบ CPU ที่ใช้ในการทำ key generation และ การคำนวณ $shared\ secret$

ส่วนอื่นนอกจากนี้คือการเตรียมไฟล์ให้สามารถ compile ไปได้กับทั้งโปรแกรม ทั้งในส่วนของไฟล์ที่เราเพิ่มมาคือ validation และ test_sidh เราทำได้โดยการแก้ไขในส่วนของ Makefile ที่ใช้สร้างทั้งโปรโตคอลนี้ขึ้นมา ดังภาพที่ 3.11

```
endif

AR=ar rcs
RANLIB=ranlib

CFLAGS=$(OPT) $(ADDITIONAL_SETTINGS) -D $(ARCHITECTURE) -D __LINUX__ -D $(USE_OPT_LEVEL) $(MULX) $(ADX)
LDFLAGS=-lm
ifeq "$(USE_OPT_LEVEL)" "_GENERIC_"
    EXTRA_OBJECTS_751=objs751/fp_generic.o
else ifeq "$(USE_OPT_LEVEL)" "_FAST_"
    EXTRA_OBJECTS_751=objs751/fp_x64.o objs751/fp_x64_asm.o
endif
OBJECTS_751=objs751/P751.o $(EXTRA_OBJECTS_751) objs/random.o objs/fips202.o objs/validation.o

all: lib751 tests KATS

objs751/%.o: P751/%.c
    @mkdir -p $(@D)
    $(CC) -c $(CFLAGS) $< -o $@

ifeq "$(USE_OPT_LEVEL)" "_GENERIC_"
    objs751/fp_generic.o: P751/generic/fp_generic.c
        $(CC) -c $(CFLAGS) P751/generic/fp_generic.c -o objs751/fp_generic.o
else ifeq "$(USE_OPT_LEVEL)" "_FAST_"
    objs751/fp_x64.o: P751/AMD64/fp_x64.c
        $(CC) -c $(CFLAGS) P751/AMD64/fp_x64.c -o objs751/fp_x64.o

    objs751/fp_x64_asm.o: P751/AMD64/fp_x64_asm.S
        $(CC) -c $(CFLAGS) P751/AMD64/fp_x64_asm.S -o objs751/fp_x64_asm.o
endif

objs/random.o: random/random.c
    @mkdir -p $(@D)
    $(CC) -c $(CFLAGS) random/random.c -o objs/random.o

objs/validation.o: validation.c
    $(CC) -c $(CFLAGS) validation.c -o objs/validation.o

objs/fips202.o: sha3/fips202.c
    $(CC) -c $(CFLAGS) sha3/fips202.c -o objs/fips202.o

lib751: $(OBJECTS_751)
    rm -rf lib751 sike sidh
    mkdir lib751 sike sidh
    $(AR) lib751/libsike.a $^
    $(RANLIB) lib751/libsike.a

tests: lib751
    $(CC) $(CFLAGS) -L./lib751 tests/test_SIKEp751.c tests/test_extras.c -lsike $(LDFLAGS) -o sike/test_KEM $(ARM_SETTING)
    $(CC) $(CFLAGS) -L./lib751 tests/test_SIDHp751.c tests/test_extras.c -lsike $(LDFLAGS) -o sidh/test_SIDH $(ARM_SETTING)
# AES
```

ภาพที่ 3.11 Code ส่วน Makefile ที่ใช้ในการสร้างโปรโตคอล

และสุดท้ายคือผลลัพธ์จากการรันโปรโตคอลนี้แบบดั้งเดิมที่ไม่ได้มีการตรวจสอบกุญแจสาธารณะนั้นสามารถวัดรอบ CPU ได้ตามภาพที่ 3.12



ภาพที่ 3.12 ผลลัพธ์จากการประเมินผลรอบการทำงานของขณะที่ไม่มีการตรวจสอบกุญแจสาธารณะ

ทั้งหมดนี้เป็นอันจบส่วนการพิจารณาการเตรียมประเมินผลการทำงานของโปรโตคอลหากว่าต้องมีการเพิ่มกระบวนการตรวจสอบกุญแจสาธารณะเข้าไป

3.3 เครื่องมือที่ใช้ในการทดสอบโปรโตคอล

3.3.1 อุปกรณ์ที่ใช้ในการทำงาน

- MacBook Pro (Retina, 13-inch, Early 2015)

  ระบบปฏิบัติการ macOS High Sierra 10.13.6

  Processor 2.7 GHz Intel Core i5

  Memory 8 GB 1867 MHz DDR3

  Graphics Intel Iris Graphic 6100 1536 MB

- Personal Computer

  ระบบปฏิบัติการ Windows 10 Pro

  Processor 2.9 GHz Intel Core i5

  Memory 16GB 2400 MHz DDR4

  Graphics NVIDIA GeForce GTX 1070 Ti 8GB

3.3.2 ซอฟต์แวร์ที่ใช้ในการทำงาน

- Visual studio code version 1.43.0

  Extension: C/C++, Live Share, Vim

- SageMath version 8.7
- Cmake version 3.12.4
- Clang compiler version 10.0.0
- GMP version 6.1.2

3.4 การตรวจกุญแจสาธารณะของ Bob ที่ตรวจโดย $Alice$

ในส่วนนี้จะพูดถึงวิธีการพิจารณากุญแจสาธารณะของ Bob pk_B ที่ส่งให้ $Alice$ ในขั้นตอนของ decapsulation โดยเราจะทบทวนเนื้อหากันก่อนว่า pk_B นั้นประกอบด้วยอะไรบ้าง และมีส่วนใดที่ทำให้ $Alice$ สามารถตรวจสอบได้ว่า Bob นั้นทำตามข้อกำหนดของโปรโตคอลอย่างถูกต้อง
เริ่มจากลักษณะของ pk_B จะเขียนได้เป็น

- $pk_B = (bit, t_1, t_2, t_3, A, s, r)$
- bit ใช้บอกลักษณะของ t_1, t_2, t_3 หรือก็คือการบอกว่า α_P หรือ β_P ที่เป็นสมาชิกใน Z_n^* โดย bit มีค่าเพียง 0 หรือ 1 (เป็นสมาชิกของ Z_2)
- t_1, t_2, t_3 คือรูปแบบของค่า $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ในลักษณะของ
- $t_1 = \alpha_P^{-1}\beta_P$ หรือ $\beta_P^{-1}\alpha_P$
- $t_2 = \alpha_P^{-1}\alpha_Q$ หรือ $\beta_P^{-1}\alpha_Q$
- $t_3 = \alpha_P^{-1}\beta_Q$ หรือ $\beta_P^{-1}\beta_Q$
- โดยที่ t_1, t_2, t_3 เป็นสมาชิกใน Z_n เมื่อ $n = 2^{e_A}$

- A เป็น curve parameter สำหรับ elliptic curve $E: y^2 = x^3 + Ax^2 + x$ ที่เป็น supersingular curve และมีค่า j-invariant อยู่ใน F_{p^2}
- s เป็น entangle bit ที่ช่วยในการคำนวณ basis R_1, R_2 สำหรับ $E[2^{e_A}]$
- r เป็น counter index ในตาราง QNR, QR ที่เก็บค่า $v = \frac{1}{(1+ur^2)}$ โดยที่เราจะมีตารางของค่า $u = u_0^2$ และค่า u_0 ที่เป็นสมาชิกใน $F_{p^2} \setminus F_p$ และใช้ช่วยในการคำนวณ basis R_1, R_2 สำหรับ $E[2^{e_A}]$

เราจะเริ่มการตรวจสอบกุญแจสาธารณะของ Bob ตามขั้นตอนดังนี้

1. ตรวจสอบ curve parameter A
2. ตรวจสอบ s, r ที่จะนำไปสร้าง basis R_1, R_2
3. ตรวจสอบ t_1, t_2, t_3 ว่าเกี่ยวข้องกับจุด $P, Q = \phi_B(P_A), \phi_B(Q_A)$ ซึ่งจะต้องเป็นสมาชิกของ $E[2^{e_A}]$ โดยใช้สมบัติของ R_1, R_2 ที่ยืนยันแล้วว่าสร้างได้ถูกต้องมาช่วยในการตรวจสอบ

3.4.1 ตรวจสอบ A

ตามที่ได้เคยบอกไปว่าเราจะสนใจเพียง supersingular elliptic curve ที่มีค่า j-invariant อยู่ใน F_{p^2} เท่านั้น โดยเราจะสามารถตรวจสอบก่อนได้เลยว่าสำหรับ Curve $A = a + bi$ ที่ $a, b \in F_p$ มีค่า j-invariant อยู่ใน F_{p^2} หรือไม่ดังนี้

- เขียน $A = a + bi$
- คำนวณ $256(A^2 - 3)^3$ แล้วเขียนในรูปของ $c + di$
- คำนวณ $A^2 - 4$ แล้วเขียนในรูปของ $e + fi$ โดยที่ $a, b, c, d, e, f \in F_p$

โดยที่ curve นั้นอยู่ในรูปของ montgomery form ที่มีเงื่อนไขด้วยว่า $A^2 - 4 \neq 0$ ดังนั้น e และ f ต้องไม่เป็น 0 พร้อมกัน และจากบทพิสูจน์ของ Frobenius automorphism เราจะบอกได้ว่า j-invariant ของ E_A จะเป็นสมาชิกใน F_p ก็ต่อเมื่อ $cf = de$ เท่านั้น

เราจึงสรุปการตรวจสอบได้โดยตรวจสอบว่าถ้า $cf \neq de$ แล้ว j-invariant ของ E_A จะอยู่ใน F_{p^2} รวมไปถึงจะตรวจสอบได้ด้วยว่า $A \in F_p$ นั้นถูกส่งมาหรือไม่ (การตรวจสอบ subfield curve) เพราะถ้าเป็นกรณีนั้นเราจะได้ว่า $b = 0$ ที่ส่งผลให้ $d = f = 0$ ด้วยเช่นเดียวกัน

ส่วนถัดมาที่ต้องตรวจสอบคือการตรวจสอบว่า E_A นั้นเป็น supersingular elliptic curve โดยตัวนิยามของ supersingular elliptic curve นั้นมีได้หลายแบบ แต่เราจะเลือกนิยามที่สามารถใช้ในการตรวจสอบง่ายที่สุดคือ การตรวจสอบว่า E_A ไม่ได้เป็น ordinary curve

(การตรวจสอบด้วยวิธีนี้ไม่ได้สำเร็จ 100% แต่ถ้าตรวจสอบสำเร็จแล้วโอกาสที่จะพลาดว่า Curve นั้นเป็น ordinary curve มีเพียงราว $1/p \approx 1/2^{751}$ เท่านั้น โดยเราถือว่าโอกาศเกิดขึ้นมีน้อยมาก) แล้วเราจะเริ่มจากนิยามของ supersingular elliptic curve ดังนี้

ถ้า E_A เป็น supersingular elliptic curve อยู่บน field ที่มี characteristic $p > 3$ และ E นิยามอยู่บน F_q ที่ $q = p^2$ กรณีใดกรณีหนึ่งต่อไปนี้ต้องเป็นจริง

- $E(F_{p^2}) \cong (Z/(p-1)Z)^2$
- $E(F_{p^2}) \cong (Z/(p+1)Z)^2$

ทำให้เราสามารถออกแบบอัลกอริทึมในการตรวจสอบว่า curve E_A นั้นไม่เป็น ordinary curve ได้คือ

- สุ่มจุด P ที่เป็นสมาชิกของ $E(F_{p^2})$
- คำนวณหา $[p-1]P$ ตรวจสอบว่าได้จุดที่ infinity (O) หรือไม่
- คำนวณหา $[p+1]P$ ตรวจสอบว่าได้จุดที่ infinity (O) หรือไม่

ซึ่งถ้าทั้งสองการตรวจสอบนี้บอกว่า $[p-1]P \neq O$ และ $[p+1]P \neq O$ เราจะบอกว่า curve E_A นั้นเป็น ordinary curve แต่ถ้าเคสใดเคสหนึ่งเป็นจริง $[p-1]P = O$ หรือ $[p+1]P = O$ เราจะบอกว่า curve E_A นั้นเป็น supersingular curve

โดยสรุปแล้ว ทั้งสองส่วนนี้คือการตรวจสอบทั้งหมดของ curve parameter A ที่สามารถทำได้ในขณะนี้ โดยจะขอจบการตรวจสอบในส่วนของ curve ที่ส่วนนี้เอง

3.4.2 ตรวจสอบ s, r

สิ่งแรกที่เราตรวจได้เลยคือค่า s ที่ใช้บอกว่าต้องใช้ค่าจากตาราง QNR หรือ QR โดยเราจะใช้ QR ถ้า $s = 0$ และจะใช้ QNR ถ้า $s = 1$ ซึ่งการเลือกใช้ตารางนี้ขึ้นอยู่กับว่า curve parameter A นั้นเป็น QR หรือ QNR ที่เราแบ่งเป็นกรณีไว้ดังนี้

- ถ้า A เป็น QR ต้องใช้ค่าในตาราง QNR หรือก็คือ $s = 1$ ถ้า A เป็น QR
- ถ้า A เป็น QNR ต้องใช้ค่าในตาราง QR หรือก็คือ $s = 0$ ถ้า A เป็น QNR

ดังนั้นการจะตรวจสอบค่า s จะต้องถัดมาจากการตรวจสอบว่า A นั้นถูกต้องแล้วเท่านั้นโดยเราทำได้ดังนี้

- เขียน $A = a + bi$ และเรามี $a, b \in F_p$
- คำนวณค่า $z = a^2 + b^2$
- คำนวณค่า $S = z^{(p+1)/4}$
- ถ้า $S^2 = z$ แปลว่า A เป็น QR หรือก็คือ s ต้องเท่ากับ 1

- ถ้า $S^2 \neq z$ แปลว่า A เป็น QNR หรือก็คือ s ต้องเท่ากับ 0

ถ้าค่า s ไม่ตรงกับสองกรณีนี้ แปลว่ากุญแจสาธารณะที่ได้รับมานั้นไม่ถูกต้อง

ส่วนถัดมาคือการตรวจค่า r ที่จะคล้ายกันคือเป็นการตรวจสอบว่าค่าที่เราเลือกมาในตาราง $T_i[r]$ นั้นต้องมาจาก QNR หรือ QR ตามที่เรากำหนดโปรโตคอลไว้หรือไม่ ซึ่งทำได้ดังนี้หลังจากที่เรายืนยันแล้วว่าค่า S ในการเลือกตารางมาใช้นั้นถูกต้อง

- เลือกตาราง T_1 ของ QNR หรือ T_2 ของ QR ขึ้นกับค่า s ที่ได้รับ
- ดูตารางค่า $T_i[r]$ เก็บไว้เป็นค่า v
- คำนวณ $x = -Av$
- คำนวณ $t = x(x^2 + Ax + 1)$
- ตรวจสอบ quadraticity ของ t โดยใช้วิธีกับที่เราตรวจ A คือ
- เขียน $t = a + bi$
- คำนวณค่า $z = a^2 + b^2$
- คำนวณค่า $S = z^{(p+1)/4}$

ทีนี้ในส่วนของเงื่อนไขของ S กับ z จะต่างกันคือ

- ถ้า $S^2 = z$ แปลว่า r ที่ได้รับมานั้นถูกต้อง
- ถ้า $S^2 \neq z$ แปลว่า r ที่ได้รับมานั้นไม่ถูกต้อง

เพราะจากตัวโปรโตคอลในการสร้าง R_1, R_2 เราจะต้องได้ t ที่เป็น QR เสมอในทุกกรณี

ถ้าการตรวจนี้ผ่านเราก็จะปล่อยให้สามารถนำค่า t, S, z ที่คำนวณไว้แล้วไปใช้ต่อในการสร้าง basis R_1, R_2 สำหรับ $E[2^{e_A}]$ ได้ แต่ถ้าไม่ผ่านแปลว่า s, r ที่ได้รับมานั้นไม่ถูกต้องตามโปรโตคอล เป็นอันจบการตรวจสอบในส่วนของค่า s, r ที่ใช้สร้าง basis R_1, R_2 นี้เอง

3.4.3 ตรวจสอบ $\alpha_P, \beta_P, \alpha_Q, \beta_Q$

ส่วนนี้เองที่เป็นส่วนสุดท้ายในการตรวจสอบ โดยในตอนแรกนั้นจะขอเสนอแนวทางการตรวจสอบในรูปแบบของ $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ธรรมดาก่อนที่จะถูกยุบรวมเหลือ 3 ค่า t_1, t_2, t_3 ที่ขึ้นอยู่กับค่า bit ที่ส่งมา โดยเราจะพูดถึงส่วนนั้นในหัวข้อถัดไป (3.6)

ทบทวนก่อนว่า $Alice$ มีจุดสาธารณะคือ P_A, Q_A ที่เป็นสมาชิกของ $E(F_{p^2})$ และเป็น basis สำหรับ $E[2^{e_A}]$ หลังจากนั้น Bob จะทำการคำนวณ isogeny บนสองจุดนี้ ทำให้กลายเป็นจุดใหม่คือ $\phi_B(P_A), \phi_B(Q_A) = P, Q$

จากนั้นด้วยวิธีการหา basis point R_1, R_2 ที่เป็นสมาชิกใน $E[2^{e_A}]$ และมีสมบัติ independence ซึ่งกันและกัน เราจะเขียนได้ว่า $P = \alpha_P R_1 + \beta_P R_2$ และ $Q = \alpha_Q R_1 + \beta_Q R_2$

เราจะเริ่มจากการตรวจว่าจุด P, Q ที่ได้รับในรูปแบบเดิมก่อนที่จะถูกบีบอัดว่าถูกต้องหรือไม่ โดยข้อกำหนดเดิมของตัวโปรโตคอลนั้นจุด P, Q จะต้องมี order เท่ากันคือ l^e บน elliptic curve E_A ในรูปแบบ montgomery form และเป็น isogenous กับ curve เริ่มต้น E_0 ไม่อย่างนั้นจะมีการโจมตีที่ชื่อว่า small subgroup attack จากการที่จุด P, Q มี order ที่ไม่มาก

อีกการตรวจสอบหนึ่งคือสำหรับจุด P, Q ที่ได้รับ เราต้องตรวจสอบว่า $Q \neq [\lambda_1]P \wedge P \neq [\lambda_2]Q$ สำหรับทุกค่า $\lambda_i \in Z$ เพราะถ้าจุดใดจุดหนึ่งนั้นเป็นผลลัพธ์มาจากการคูณด้วยจำนวนเต็มกับอีกจุดหนึ่งจะทำให้ $shared\ secret$ ที่คำนวณได้นั้นไม่ขึ้นกับกุญแจลับของอีกฝั่งที่ทำตามเงื่อนไขของโปรโตคอล ในส่วนนี้เราถือว่าสามารถตรวจสอบได้ง่าย เพราะว่าสำหรับ P_1, P_2 ที่ $P_1 = [a]P$ และ $P_2 = [b]P$ เราจะได้ว่าค่าของ Weil pairing จะเป็นรูปแบบ trivial คือ $e(P_1, P_2) = e([a]P, [b]P) = e(P, P)^{ab} = 1^{ab} = 1$

ทั้งสองการตรวจสอบนี้สามารถทำได้โดยการใช้สมบัติของ Weil pairing มาช่วยในการคำนวณและตรวจสอบได้ โดยการตรวจ order ของจุด P, Q ที่ได้รับว่าต้องมีค่า $l_A^{e_A}$ ก็คือการคำนวณค่า Weil pairing $e_{l_A^{e_A}}(P, Q)$ ที่ต้องมีค่าเดียวกับของจุดเริ่มต้นสาธารณะ $e_{l_A^{e_A}}(P_A, Q_A)$ ที่จากการกำหนดจุดเริ่มต้นนั้นมีค่า $e(P_A, Q_A) = l_A^{e_A - 1}$ ทำให้จุด P, Q ที่มาจากการใช้ isogeny ก็ต้องมี Weil pairing ที่เท่ากันคือ $e(P, Q) = l_A^{e_A - 1}$ และจากสมบัติของ Weil pairing ที่มีนิยามว่า

ถ้าจุด P, Q อยู่ใน $E[mn]$ ค่า n-th power ของ Weil pairing $e_{mn}(P, Q)$ จะหาได้จาก $e_{mn}(P, Q)^n = e_m([n]P, [n]Q)$
จากเงื่อนไขนี้เองเรากำหนดให้ $mn = l^{e_A} = 2^{372}$ เช่น $m = 2^2$ และ $n = 2^{370}$ จะได้ว่า

$$e_{2^{372}}(P, Q)^{2^{370}} = e_{2^2 2^{370}}(P, Q)^{2^{370}} = e_{2^2}([2^{370}]P, [2^{370}]Q)$$

ทำให้เราสรุปได้ว่าถ้า P, Q นั้นมี order 2^{372} ค่าของจุด $P' = [2^{370}]P$ และ $Q' = [2^{370}]Q$ ก็ต้องมี order $2^2 = 4$ ทำให้ค่าของ Weil pairing $e_4(P', Q') \neq 1$

และเราจะมีอีกสมบัติหนึ่งของ Weil pairing ก็คือ $e(P, P) = 1$ เสมอ ตัวเงื่อนไขนี้เองที่ทำให้เราสามารถตรวจกุญแจสาธารณะทั้งสองสมบัติที่กล่าวไว้ในตอนแรกคือ order มีค่า l^e และจุด P', Q' นั้นไม่ได้เกี่ยวข้องกัน

การตรวจสอบทั้งหมดที่กล่าวมานี้ เทียบเท่ากับการตรวจว่าจุด P' ต้องไม่เท่ากับจุด Q' เราถึงจะบอกได้ว่าจุด P, Q ที่ Bob ส่งมานั้นถูกต้อง เป็นอันจบส่วนการพิจารณาการตรวจสอบกุญแจสาธารณะในรูปแบบเดิมก่อนถูกบีบอัด

ในส่วนถัดมาเราก็จะอาศัยวิธีตรวจสอบแบบเดิมนี้เองในการตรวจสอบกุญแจสาธารณะ เพียงแต่เราแทนค่าตามรูปแบบของกุญแจสาธารณะที่ถูกบีบอัดคือ $P = \alpha_P R_1 + \beta_P R_2$ และ $Q = \alpha_Q R_1 + \beta_Q R_2$ จะได้ว่า

$$P' = [2^{370}]P = [\alpha_P 2^{370}]R_1 + [\beta_P 2^{370}]R_2$$
$$Q' = [2^{370}]Q = [\alpha_Q 2^{370}]R_1 + [\beta_Q 2^{370}]R_2$$

แล้วเรานำมาคำนวณ Weil pairing ในรูปแบบเดียวกัน โดยในคราวนี้เราจะอาศัยสมบัติของ Weil pairing ในเรื่องของ bilinearity, non-degenerate มาช่วยในการจัดรูปผลลัพธ์

$$\begin{aligned} e_4(P', Q') &= e_4\big([\alpha_P 2^{370}]R_1 + [\beta_P 2^{370}]R_2, [\alpha_Q 2^{370}]R_1 + [\beta_Q 2^{370}]R_2\big) \\ &= e_4\big([\alpha_P 2^{370}]R_1, [\alpha_Q 2^{370}]R_1 + [\beta_Q 2^{370}]R_2\big) \\ &\quad e_4\big([\beta_P 2^{370}]R_2, [\alpha_Q 2^{370}]R_1 + [\beta_Q 2^{370}]R_2\big) \\ &= e_4\big([\alpha_P 2^{370}]R_1, [\alpha_Q 2^{370}]R_1\big)\, e_4\big([\alpha_P 2^{370}]R_1, [\beta_Q 2^{370}]R_2\big) \\ &\quad e_4\big([\beta_P 2^{370}]R_2, [\alpha_Q 2^{370}]R_1\big)\, e_4\big([\beta_P 2^{370}]R_2, [\beta_Q 2^{370}]R_2\big) \end{aligned}$$

ทำให้เรามีทั้งหมด 4 เทอมที่คูณกันอยู่ได้แก่

1. $e_4\big([\alpha_P 2^{370}]R_1, [\alpha_Q 2^{370}]R_1\big)$
2. $e_4\big([\alpha_P 2^{370}]R_1, [\beta_Q 2^{370}]R_2\big)$
3. $e_4\big([\beta_P 2^{370}]R_2, [\alpha_Q 2^{370}]R_1\big)$
4. $e_4\big([\beta_P 2^{370}]R_2, [\beta_Q 2^{370}]R_2\big)$

ซึ่งเราไม่ต้องการให้ผลคูณของทั้ง 4 พจน์นี้คูณกันแล้วเป็น 1 เพื่อให้ $e_4(P', Q') \neq 1$ ทำให้เราสรุปได้ว่า

- $\alpha_P \neq \alpha_Q$และ $\beta_P \neq \beta_Q$
- $\alpha_P 2^{370}$ order ของ R_1
- $\beta_Q 2^{370}$ order ของ R_2
- $\beta_P 2^{370}$ order ของ R_2
- $\alpha_Q 2^{370}$ order ของ R_1

ทำให้การตรวจสอบเหลือเพียงแค่

- $\alpha_P \neq \alpha_Q$
- $\beta_P \neq \beta_Q$
- $\alpha_P 2^{370}, \beta_P 2^{370}, \alpha_Q 2^{370}, \beta_Q 2^{370} \neq 2^{372}$ ภายใต้ $mod\ 2^{372}$

เป็นอันจบส่วนของการตรวจสอบค่า α, β ของจุด P, Q สำหรับกุญแจสาธารณะของ Bob ที่ส่วนนี้

3.5 การตรวจกุญแจสาธารณะของ $Alice$ ที่ตรวจโดย Bob

ในส่วนนี้จะพูดถึงวิธีการพิจารณากุญแจสาธารณะของ $Alice$ pk_A ที่ส่งให้ Bob ในขั้นตอนของ encapsulation โดยเราจะทบทวนเนื้อหากันก่อนว่า pk_A นั้นประกอบด้วยอะไรบ้าง และมีส่วนใดที่ทำให้ Bob สามารถตรวจสอบได้ว่า $Alice$ นั้นทำตามข้อกำหนดของโปรโตคอลอย่างถูกต้อง
เริ่มจากลักษณะของ pk_A จะเขียนได้เป็น

- $pk_A = (bit, t_1, t_2, t_3, A, r_1, r_2)$
- bit ใช้บอกลักษณะของ t_1, t_2, t_3 หรือก็คือการบอกว่า α_P หรือ β_P ที่เป็นสมาชิกใน Z_n^* โดย bit มีค่าเพียง 0 หรือ 1 (เป็นสมาชิกของ Z_2)
- t_1, t_2, t_3 คือรูปแบบของค่า $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ในลักษณะของ
- $t_1 = \alpha_P^{-1}\beta_P$ หรือ $\beta_P^{-1}\alpha_P$
- $t_2 = \alpha_P^{-1}\alpha_Q$ หรือ $\beta_P^{-1}\alpha_Q$
- $t_3 = \alpha_P^{-1}\beta_Q$ หรือ $\beta_P^{-1}\beta_Q$
- โดยที่ t_1, t_2, t_3 เป็นสมาชิกใน Z_n เมื่อ $n = 2^{e_A}$
- A เป็น curve parameter สำหรับ elliptic curve $E: y^2 = x^3 + Ax^2 + x$ ที่เป็น supersingular curve และมีค่า j-invariant อยู่ใน F_{p^2}
- r_1 เป็น elligator counter ที่ช่วยในการคำนวณ basis R_1 สำหรับ $E[3^{e_B}]$
- r_2 เป็น elligator counter ที่ช่วยในการคำนวณ basis R_2 สำหรับ $E[3^{e_B}]$

เราจะเริ่มการตรวจสอบกุญแจสาธารณะของ $Alice$ ตามขั้นตอนดังนี้

1. ตรวจสอบ curve parameter A
2. ตรวจสอบ r_1, r_2 ที่จะนำไปสร้าง basis R_1, R_2
3. ตรวจสอบ t_1, t_2, t_3 ว่าเกี่ยวข้องกับจุด $P, Q = \phi_A(P_B), \phi_A(Q_B)$ ซึ่งจะต้องเป็นสมาชิกของ $E[3^{e_B}]$ โดยใช้สมบัติของ R_1, R_2 ที่ยืนยันแล้วว่าสร้างได้ถูกต้องมาช่วยในการตรวจสอบ

3.5.1 ตรวจสอบ A

ในส่วนนี้นั้นจะใช้วิธีเดียวกับที่ $Alice$ ตรวจสอบ curve parameter A ของ pk_B ดังนั้นจะขอละการอธิบายไว้ในส่วนนี้ แต่ผลลัพธ์ที่ได้จะเหมือนกันคือเราจะยืนยันได้ว่า A นั้นเป็น parameter ที่เป็น supersingular elliptic curve และค่า j-invariant ของ E_A นั้นอยู่ใน F_{p^2}

3.5.2 ตรวจสอบ r_1, r_2

เราจะเริ่มจากว่าการจะสร้าง basis R_1, R_2 ที่ถูกต้องใน $E[3^{e_B}]$ นั้นจะมาจากการดูค่าในตารางสาธารณะ T (ใช้ตารางเดียวกันสำหรับทั้ง R_1, R_2) ของค่า $v = \frac{1}{1+Ur^2}$ เมื่อให้ $U = 4 + i$ โดยทั้งสองค่าจะตรวจสอบแบบเดียวกัน แต่เราจะเริ่มพิจารณาตั้งแต่วิธีการสร้าง basis R ทั่วไปก่อนสำหรับ $E[3^{e_B}]$ ที่สามารถทำดังนี้

- รับค่า curve parameter A และค่า r
- ดูค่าในตาราง เก็บไว้เป็น $v = T[r]$
- คำนวณ $x = -Av$
- คำนวณ $y = x(x^2 + Ax + 1)$ เขียนในรูป $a + bi$
- คำนวณ $N = a^2 + b^2$
- คำนวณ $z = N^{(p+1)/4}$
- ตรวจสอบว่า $z^2 = N$ หรือไม่
- ถ้าเท่ากัน แปลว่าสร้าง R สำเร็จ
- แต่ถ้าไม่เท่ากันให้ปรับค่า x เป็น $x = -x - A$ แล้วคำนวณต่อตามขั้นตอนการสร้าง basis
- สุดท้ายจะได้ basis R คือ $(x, z) = [2^{372}](x, z)$

จะเห็นได้ว่าเราตรวจสอบความเป็น QR ของค่า y ที่เกี่ยวข้องกับค่า A, x โดยที่ค่า x นั้นขึ้นอยู่กับค่า r อีกที ซึ่งถ้าหากการตรวจสอบไม่สำเร็จเราก็จะปรับค่า x นั้นแล้วทำต่อเลย ทั้งที่มีความเป็นไปได้ว่าถ้าค่า r ที่ได้รับนั้นไม่ถูกต้องต่อให้เราปรับค่า x แล้วการตรวจสอบนั้นจะไม่สำเร็จอีกรอบก็ได้ ส่งผลทำให้เราได้ตัว basis R ที่ไม่ถูกต้อง ดังนั้นถ้าจะทำการตรวจสอบเราจะต้องเริ่มจาก

- ตรวจสอบได้ว่า $z^2 \neq N$
- ปรับค่า x เป็น $x = -x - A$
- คำนวณ $y_2 = x(x^2 + Ax + 1)$ เขียนในรูป $c + di$
- คำนวณ $m = c^2 + d^2$
- คำนวณ $z_2 = m^{(p+1)/4}$

แล้วทำการตรวจว่าสอบ $z_2^2 = m$ หรือไม่ ซึ่งถ้าการตรวจสอบรอบนี้นั้นไม่สำเร็จแปลว่าค่า r ที่ได้รับนั้นไม่ถูกต้อง โดยเราทำแบบนี้กับทั้ง r_1 และ r_2 เป็นอันจบส่วนการตรวจสอบค่า r_1, r_2 ที่ใช้สร้าง basis R_1, R_2

3.5.3 ตรวจสอบ $\alpha_P, \beta_P, \alpha_Q, \beta_Q$

ส่วนนี้เป็นส่วนสุดท้ายที่ต้องตรวจสอบเช่นเดียวกันสำหรับ Bob โดยในตอนแรกนั้นจะขอเสนอแนวทางการตรวจสอบในรูปแบบของ $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ธรรมดาก่อนที่จะถูกยุบรวมเหลือ 3 ค่า t_1, t_2, t_3 ที่ขึ้นอยู่กับค่า bit ที่ส่งมา

ทบทวนก่อนว่า Bob มีจุดสาธารณะคือ P_B, Q_B ที่เป็นสมาชิกของ $E\left(F_{p^2}\right), E(F_P)$ และเป็น basis สำหรับ $E[3^{e_B}]$ หลังจากนั้น $Alice$ จะต้องทำการคำนวณ isogeny บนสองจุดนี้ให้กลายเป็นจุดใหม่คือ $\phi_A(P_B), \phi_A(Q_B) = P, Q$

จากนั้นด้วยวิธีการหา basis point R_1, R_2 ที่เป็นสมาชิกใน $E[3^{e_B}]$ และมีสมบัติ independence ซึ่งกันและกัน เราจะเขียนได้ว่า $P = \alpha_P R_1 + \beta_P R_2$ และ $Q = \alpha_Q R_1 + \beta_Q R_2$

การตรวจสอบของ Bob จะทำเหมือนกับของ $Alice$ โดยจะเริ่มจากรูปแบบธรรมดาก่อนที่ไม่มีการบีบอัดกุญแจสาธารณะซึ่งสมบัติที่ต้องการตรวจสอบนั้นจะเหมือนกัน โดยส่วนที่ต่างกันอยู่การคำนวณ Weil pairing เริ่มต้นของจุด P_B, Q_B เดิมที่จะมีค่า $e(P_B, Q_B) = l_B^{e_B}$ แทน และส่วนที่เหลือจะเป็นการเปลี่ยน parameter เป็น l_B และ e_B รวมถึงการเลือกค่า m, n ที่ต่างกันโดยในที่นี้จะใช้ $mn = l_B^{e_B} = 3^{239}$, $m = 3$ และ $n = 3^{238}$ แล้วเขียนในรูปของ

$$e_{3^{239}}(P, Q)^{3^{238}} = e_{3^1 3^{238}}(P, Q)^{3^{238}} = e_3([3^{238}]P, [3^{238}]Q)$$

และได้ข้อสรุปว่าถ้าให้ $P' = [3^{238}]P$ และ $Q' = [3^{238}]Q$ จะได้ว่าค่าของ Weil pairing $e_3(P', Q') \neq 1$ สำหรับรูปแบบของกุญแจสาธารณะแบบเดิมก่อนที่จะถูกทำการบีบอัด

สุดท้ายเราก็ใช้การแทนค่า $P = \alpha_P R_1 + \beta_P R_2$ และ $Q = \alpha_Q R_1 + \beta_Q R_2$ เพื่อทำให้เป็นการตรวจสอบกุญแจสาธารณะในรูปแบบที่ถูกบีบอัด ทำให้ได้ว่า

$$P' = [3^{238}]P = [\alpha_P 3^{238}]R_1 + [\beta_P 3^{238}]R_2$$
$$Q' = [3^{238}]Q = \left[\alpha_Q 3^{238}\right]R_1 + \left[\beta_Q 3^{238}\right]R_2$$

แล้วจัดรูปผลลัพธ์ของการหา Weil pairing

$$\begin{aligned} e_3(P', Q') &= e_3\left([\alpha_P 3^{238}]R_1 + [\beta_P 3^{238}]R_2, \left[\alpha_Q 3^{238}\right]R_1 + \left[\beta_Q 3^{238}\right]R_2\right) \\ &= e_3\left([\alpha_P 3^{238}]R_1, \left[\alpha_Q 3^{238}\right]R_1 + \left[\beta_Q 3^{238}\right]R_2\right) \\ &\quad e_3\left([\beta_P 3^{238}]R_2, \left[\alpha_Q 3^{238}\right]R_1 + \left[\beta_Q 3^{238}\right]R_2\right) \\ &= e_3\left([\alpha_P 3^{238}]R_1, \left[\alpha_Q 3^{238}\right]R_1\right) e_3\left([\alpha_P 3^{238}]R_1, \left[\beta_Q 3^{238}\right]R_2\right) \\ &\quad e_3\left([\beta_P 3^{238}]R_2, \left[\alpha_Q 3^{238}\right]R_1\right) e_3\left([\beta_P 3^{238}]R_2, \left[\beta_Q 3^{238}\right]R_2\right) \end{aligned}$$

ที่เราได้ข้อสรุปคล้ายที่คล้ายกับการตรวจสอบของ $Alice$ คือ

- $\alpha_P \neq \alpha_Q$และ $\beta_P \neq \beta_Q$
- $\alpha_P 3^{238}$ order ของ R_1

- $\beta_Q 3^{238}$ order ของ R_2
- $\beta_P 3^{238}$ order ของ R_2
- $\alpha_Q 3^{238}$ order ของ R_1

ทำให้การตรวจสอบเหลือเพียงแค่

- $\alpha_P \neq \alpha_Q$
- $\beta_P \neq \beta_Q$
- $\alpha_P 3^{238}, \beta_P 3^{238}, \alpha_Q 3^{238}, \beta_Q 3^{238} \neq 3^{239}$ ภายใต้ $mod\ 3^{239}$

เป็นอันจบส่วนของการตรวจสอบค่า α, β ของจุด P, Q สำหรับกุญแจสาธารณะของ $Alice$ ที่ส่วนนี้

3.6 การตรวจสอบค่า (bit, t_1, t_2, t_3)

ทบทวนก่อนว่าว่ากุญแจสาธารณะ pk ที่มาในรูปของ (bit, t_1, t_2, t_3) จากผลลัพธ์ของการบีบอัดกุญแจสาธารณะนั้นมีได้สองแบบคือ

$$\left(0, \alpha_P^{-1}\beta_P, \alpha_P^{-1}\alpha_Q, \alpha_P^{-1}\beta_Q\right) \text{ หรือ } \left(1, \beta_P^{-1}\alpha_P, \beta_P^{-1}\alpha_Q, \beta_P^{-1}\beta_Q\right)$$

ที่เราเลือกมาจากการตรวจสอบว่าถ้า $\alpha_P \in Z_n^*$ ให้ใช้แบบที่ 1 แต่ถ้า $\beta_P \in Z_n^*$ ให้ใช้แบบที่ 2 โดยการแนบค่า bit มาด้วยนี้เองในการส่งกุญแจสาธารณะ

การที่ $a \in Z_n^*$ ในที่นี้นั้นหมายความว่า ค่า a นั้นมี inverse ภายใต้ $mod\ n$ เขียนแทนด้วย $a^{-1}, a^{-1} \in Z_n,\ aa^{-1} \equiv 1\ mod\ n$

เราจะเริ่มจากการตรวจสอบว่า bit นั้นบอกได้ถูกต้องว่า α_P หรือ β_P นั้นมี inverse ใน $mod\ n$ ที่เราตรวจสอบได้จากการที่ค่า α_P, α_P^{-1} หรือ β_P, β_P^{-1} จะต้องเป็น coprime กับ n เท่านั้นถึงจะสามารถมี inverse ได้ใน $mod\ n$ (ไม่มีตัวประกอบเฉพาะร่วมกัน) ซึ่งวิธีการตรวจสอบ coprime ของจำนวนเต็ม (a, b) โดยทั่วไปนั้นมีได้หลายวิธีเช่น

- $gcd(a, b) = 1$
- $lcm(a, b) = ab$
- สามารถหาค่า $x, y \in Z$ ที่ทำให้ $ax + by = 0$

ซึ่งวิธีที่กล่าวมานี้สามารถทำได้ เช่นเราสามารถใช้ euclidean algorithm ในการคำนวณหาค่า $gcd(a, b) = 1$ หรือการคำนวณค่า $lcm(a, b) = ab$ โดยตรง หรืออีกวิธีคือการแก้สมการหาค่า x, y นั้นก็เป็นไปได้เช่นเดียวกัน แต่วิธีที่กล่าวมาย่อมทำให้เกิดการคำนวณจำนวนมากอย่างแน่นอน เพราะจำนวน

ที่เราพูดถึงในโปรโตคอลนั้นมีขนาดใหญ่มาก และอย่าลืมว่าเราไม่ได้ส่งค่า a, b นั้นมาโดยตรง แต่มาในรูปของ t_1, t_2, t_3 ซึ่งทำให้วิธีที่กล่าวมาข้างต้นดูยุ่งยากขึ้นไปอีก

กลับกันเราอาศัยข้อสังเกตว่าค่า n ในโปรโตคอลนี้จะอยู่ในรูปของ l^e หรือก็คือจำนวนเฉพาะที่เป็นตัวประกอบของ n นั้นมีเพียงแค่ l เท่านั้น ทำให้การตรวจสอบสมบัติของ coprime นี้เหลือเพียงแค่ว่า

- a ไม่ได้มี l เป็นตัวประกอบที่เป็นจำนวนเฉพาะ หรือก็คือ $l \nmid a$ นั่นเอง

3.6.1 การตรวจ bit กรณีของ $bit = 0$

เราจะตรวจสอบว่า $l \nmid \alpha_P$ ที่เทียบเท่ากับการตรวจสอบว่า $l \nmid \alpha_P^{-1}$ แต่เนื่องจากค่าที่เรามีนั้นอยู่ในรูปแบบของ $t_1 = \alpha_P^{-1}\beta_P, t_2 = \alpha_P^{-1}\alpha_Q, t_3 = \alpha_P^{-1}\beta_Q$ เราจึงต้องตรวจดังนี้
กรณีอย่างง่ายสุดคือ $\exists i \in \{1,2,3\}, l \nmid t_i$ ที่เราจะสามารถสรุปได้เลยว่า $l \nmid \alpha_P^{-1}$ แน่นอน
แต่อีกกรณีหนึ่งคือ $\forall i \in \{1,2,3\}, l \mid t_i$ เราต้องตรวจสอบต่อโดย

เราต้องการให้ $(l \mid \beta_P) \wedge (l \mid \alpha_Q) \wedge (l \mid \beta_Q)$ (จากทฤษฎีว่าถ้า $l \mid ab$ แล้ว $l \mid a \vee l \mid b$ แล้วเราไม่ต้องการให้อยู่ในกรณีของ $l \mid \alpha_P^{-1}$) ที่เทียบเท่ากับการตรวจสอบว่า

$$l \mid \beta_P + \alpha_Q + \beta_Q$$

ซึ่งถ้าเราคูณ α_P^{-1} ไปตลอดทั้งค่านั้นจะกลายเป็น

$$l \mid \alpha_P^{-1}(\beta_P + \alpha_Q + \beta_Q)$$
$$l \mid (t_1 + t_2 + t_3)$$

นั่นก็คือ ถ้า $\forall i \in \{1,2,3\}, l \mid t_i$ และ $l \mid (t_1 + t_2 + t_3)$ แล้ว $l \nmid \alpha_P^{-1}$

ในกรณีของ $Alice$ ที่ตรวจ $Bob\ pk_B$ เราก็จะใช้ $l = 2$ ส่วนในกรณีของ Bob ที่ตรวจ $Alice\ pk_A$ เราก็จะใช้ $l = 3$ เป็นอันจบการตรวจสอบกรณีของ $bit = 0$ นี่เอง

3.6.2 การตรวจ bit กรณีของ $bit = 1$

เราจะตรวจสอบว่า $l \nmid \beta_P$ ที่เทียบเท่ากับการตรวจสอบว่า $l \nmid \beta_P^{-1}$ แต่เนื่องจากค่าที่เรามีนั้นอยู่ในรูปแบบของ $t_1 = \beta_P^{-1}\alpha_P, t_2 = \beta_P^{-1}\alpha_Q, t_3 = \beta_P^{-1}\beta_Q$ เราจึงต้องตรวจดังนี้
กรณีอย่างง่ายสุดคือ $\exists i \in \{1,2,3\}, l \nmid t_i$ ที่เราจะสามารถสรุปได้เลยว่า $l \nmid \beta_P^{-1}$ แน่นอน
แต่อีกกรณีหนึ่งคือ $\forall i \in \{1,2,3\}, l \mid t_i$ เราต้องตรวจสอบต่อโดย

เราต้องการให้ $(l \mid \alpha_P) \wedge (l \mid \alpha_Q) \wedge (l \mid \beta_Q)$ ที่เทียบเท่ากับการตรวจสอบว่า

$$l \mid \alpha_P + \alpha_Q + \beta_Q$$

ซึ่งถ้าเราคูณ β_P^{-1} ไปตลอดทั้งค่านั้นจะกลายเป็น

$$l \mid \beta_P^{-1}(\beta_P + \alpha_Q + \beta_Q)$$
$$l \mid (t_1 + t_2 + t_3)$$

นั่นก็คือ ถ้า $\forall i \in \{1,2,3\}, l \mid t_i$ และ $l \mid (t_1 + t_2 + t_3)$ แล้ว $l \nmid \beta_P^{-1}$

ในกรณีของ $Alice$ ที่ตรวจ $Bob\, pk_B$ เราก็จะใช้ $l = 2$ ส่วนในกรณีของ Bob ที่ตรวจ $Alice\, pk_A$ เราก็จะใช้ $l = 3$ เป็นอันจบการตรวจสอบกรณีของ $bit = 1$ นี่เอง

หลังจากเราตรวจสอบแล้วว่าค่าของ bit ที่ได้รับนั้นส่งมาอย่างถูกต้อง ส่วนสุดท้ายคือการตรวจว่าค่า t_1, t_2, t_3 นั้นคงสมบัติเช่นเดียวกับข้อสรุปที่ได้จากในหัวข้อ 3.4.3 และ 3.5.3 ก็คือ

- $\alpha_P \neq \alpha_Q$ และ $\beta_P \neq \beta_Q$
- $\alpha_P 2^{370}, \beta_P 2^{370}, \alpha_Q 2^{370}, \beta_Q 2^{370} \neq 2^{372}$ ภายใต้ $mod\ 2^{372}$ สำหรับ $Alice$ ทีต้องการตรวจ pk_B
- $\alpha_P 3^{238}, \beta_P 3^{238}, \alpha_Q 3^{238}, \beta_Q 3^{238} \neq 3^{239}$ ภายใต้ $mod\ 3^{239}$ สำหรับ Bob ทีต้องการตรวจ pk_A

3.6.3 การตรวจ (t_1, t_2, t_3) กรณีของ $bit\ =\ 0$

จากกุญแจสาธารณะที่เราได้รับคือ $\left(0, \alpha_P^{-1}\beta_P, \alpha_P^{-1}\alpha_Q, \alpha_P^{-1}\beta_Q\right)$ เราจะเริ่มจากเงื่อนไขอย่างแรกคือ $\alpha_P \neq \alpha_Q$ ที่ถ้าหากว่า $\alpha_P = \alpha_Q$ แปลว่าค่า $t_2 \equiv \alpha_P^{-1}\alpha_Q \equiv \alpha_P^{-1}\alpha_P \equiv 1\ mod\ n$

เงื่อนไขถัดมาที่ตรวจได้คือ $\beta_P \neq \beta_Q$ โดยเรามีค่า $\alpha_P^{-1}\beta_P, \alpha_P^{-1}\beta_Q$ แทนด้วย t_1, t_3 นั้นจะมีค่าเท่ากันถ้าเงื่อนไขนั้นเป็นเท็จ ทำให้การตรวจนี้เทียบเท่ากับการตรวจสอบว่า $t_1 \neq t_3$

ส่วนถัดมาคือการตรวจเงื่อนไขของค่า $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ว่าเมื่อคูณกับค่าที่เราเลือกไว้ในตอนแรกคือ $2^{370}, 3^{238}$ แล้วไม่ได้มีค่าตรงกับ order ของจุด R_1, R_2 ทีมีค่า $2^{372}, 3^{239}$ ของกุญแจสาธารณะในแต่ละฝ่าย โดยเราจะเริ่มจากการที่ยืนยันได้แล้วจากการที่ α_P, α_P^{-1} ที่ได้รับมานั้นมี inverse ใน $mod\ n, n \in \{2^{372}, 3^{239}\}$ จากการตรวจในส่วนแรก

จาก $\alpha_P 2^{370} \not\equiv 2^{372} \not\equiv 0\ mod\ 2^{372}$ เราเขียนได้ว่า $\alpha_P \not\equiv 0\ mod\ 2^2, \alpha_P^{-1} \not\equiv 0\ mod\ 4$ ในกรณีของ pk_B หรืออีกกรณีหนึ่งคือ
$\alpha_P 3^{238} \not\equiv 3^{239} \not\equiv 0\ mod\ 3^{239}$ เราก็สามารถเขียนได้ว่า $\alpha_P \not\equiv 0\ mod\ 3^1, \alpha_P^{-1} \not\equiv 0\ mod\ 3$ ในกรณีของ pk_A เช่นเดียวกัน ทำให้เราสามารถสรุปได้ว่า

- $t_1 = \alpha_P^{-1}\beta_P \equiv 0\ mod\ m$ ก็ต่อเมื่อ $\beta_P \equiv 0\ mod\ m$
- $t_2 = \alpha_P^{-1}\alpha_Q \equiv 0\ mod\ m$ ก็ต่อเมื่อ $\alpha_Q \equiv 0\ mod\ m$
- $t_3 = \alpha_P^{-1}\beta_Q \equiv 0\ mod\ m$ ก็ต่อเมื่อ $\beta_Q \equiv 0\ mod\ m$

เมื่อให้ $m\ \in \{3,4\}$ สำหรับการตรวจ pk_A, pk_B ตามลำดับ

ทำให้สุดท้ายการตรวจสอบว่าตัว t_1, t_2, t_3 ว่าส่งมาอย่างถูกต้องหรือไม่นั้นคือการตรวจว่า t_i ต้องไม่ congruence กับ 0 ใน $mod\ m$

$$(t_1 \not\equiv 0\ mod\ m) \wedge (t_2 \not\equiv 0\ mod\ m) \wedge (t_3 \not\equiv 0\ mod\ m)$$

เราถึงจะยอมรับว่าค่า (bit, t_1, t_2, t_3) นั้นถูกส่งมาอย่างถูกต้อง

3.6.4 การตรวจ $(\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3)$ กรณีของ $\mathrm{bit} = 1$

จากกุญแจสาธารณะที่เราได้รับคือ $\left(1, \beta_P^{-1}\alpha_P, \beta_P^{-1}\alpha_Q, \beta_P^{-1}\beta_Q\right)$ เราจะเริ่มจากเงื่อนไขอย่างแรกที่ตรวจคือ $\alpha_P \neq \alpha_Q$ ที่ถ้าหากว่า $\alpha_P = \alpha_Q$ แปลว่าค่า $\beta_P^{-1}\alpha_P, \beta_P^{-1}\alpha_Q$ ซึ่งแทนด้วย t_1, t_2 นั้นจะมีค่าเท่ากันโดยการตรวจนี้เทียบเท่ากับการตรวจสอบว่า $t_1 \neq t_2$

เงื่อนไขถัดมาที่ตรวจได้ในรูปแบบนี้คือ $\beta_P \neq \beta_Q$ ซึ่งถ้าหากว่า $\beta_P = \beta_Q$ เราจะสรุปได้ว่า $t_3 \equiv \beta_P^{-1}\beta_Q \equiv \beta_P^{-1}\beta_P \equiv 1 \; mod \; n$

ส่วนถัดมาคือการตรวจเงื่อนไขของค่า $\alpha_P, \beta_P, \alpha_Q, \beta_Q$ ว่าเมื่อคูณกับค่าที่เราเลือกไว้ในตอนแรกคือ $2^{370}, 3^{238}$ แล้วไม่ได้มีค่าตรงกับ order ของจุด R_1, R_2 ทีมีค่า $2^{372}, 3^{239}$ ของกุญแจสาธารณะในแต่ละฝ่าย โดยเราจะเริ่มจากการที่ยืนยันได้แล้วจากการที่ β_P, β_P^{-1} ที่ได้รับมานั้นมี inverse ใน $mod \; n, n \in \{2^{372}, 3^{239}\}$ จากการตรวจในส่วนแรก

จาก $\beta_P 2^{370} \not\equiv 2^{372} \not\equiv 0 \; mod \; 2^{372}$ เราเขียนได้ว่า $\beta_P \not\equiv 0 \; mod \; 2^2, \beta_P^{-1} \not\equiv 0 \; mod \; 4$ ในกรณีของ pk_B หรืออีกกรณีหนึ่งคือ
$\beta_P 3^{238} \not\equiv 3^{239} \not\equiv 0 \; mod \; 3^{239}$ เราก็สามารถเขียนได้ว่า $\beta_P \not\equiv 0 \; mod \; 3^1, \beta_P^{-1} \not\equiv 0 \; mod \; 3$ ในกรณีของ pk_A เช่นเดียวกัน ทำให้เราสามารถสรุปได้ว่า

- $t_1 = \beta_P^{-1}\alpha_P \equiv 0 \; mod \; m$ ก็ต่อเมื่อ $\beta_P \equiv 0 \; mod \; m$
- $t_2 = \beta_P^{-1}\alpha_Q \equiv 0 \; mod \; m$ ก็ต่อเมื่อ $\alpha_Q \equiv 0 \; mod \; m$
- $t_3 = \beta_P^{-1}\beta_Q \equiv 0 \; mod \; m$ ก็ต่อเมื่อ $\beta_Q \equiv 0 \; mod \; m$

เมื่อให้ $m \in \{3,4\}$ สำหรับการตรวจ pk_A, pk_B ตามลำดับ

ทำให้สุดท้ายการตรวจสอบว่าตัว t_1, t_2, t_3 ว่าส่งมาอย่างถูกต้องหรือไม่นั้นคือการตรวจว่า t_i ต้องไม่ congruence กับ 0 ใน $mod \; m$

$$(t_1 \not\equiv 0 \; mod \; m) \wedge (t_2 \not\equiv 0 \; mod \; m) \wedge (t_3 \not\equiv 0 \; mod \; m)$$

เราถึงจะยอมรับว่าค่า (bit, t_1, t_2, t_3) นั้นถูกส่งมาอย่างถูกต้อง

ทั้งหมดที่กล่าวมานี้เองเป็นการตรวจสอบว่ากุญแจสาธารณะ pk_A, pk_B ที่ได้รับมานั้นถูกต้องตามรูปแบบที่โปรโตคอลกำหนด โดยเราจะปล่อยให้คำนวณต่อไปถ้าผ่านการตรวจสอบทั้งหมด และจะบังคับให้หยุดการทำงานถ้ามีเงื่อนไขใดที่ไม่ผ่าน แต่อย่างไรก็ตาม การตรวจสอบในรูปแบบนี้นั้นบอกได้เพียงแค่ว่ากุญแจสาธารณะที่ส่งมานั้น “ดูแล้วน่าจะถูกต้อง หรือ ไม่ส่งผลกระทบต่อโปรโตคอลหากต้องดำเนินการคำนวณต่อ” แต่บอกไม่ได้ว่าไม่ได้ถูกดัดแปลงมาจาก Eve ผู้ไม่พึงประสงค์ หรือที่จริงแล้วอาจจะมีชุดของ

กลุ่มจุดที่ไม่ปลอดภัยบนตัว curve นอกจากที่กล่าวมาก็เป็นไปได้ (เช่น Eve สามารถหาวิธีที่ทำให้ค่าของ j-invariant วนกลับมาเท่ากับ curve เริ่มต้น E_0 หรือไม่ขึ้นกับกุญแจลับด้วยวิธีอื่นได้) ซึ่งทั้งหมดที่กล่าวมานั้นอยู่นอกเหนือไปจากขอบเขตที่กำหนดไว้ของงานนี้ โดยตัวเป้าหมายหลักในงานนี้คือยืนยันให้ได้ว่าทั้งสองฝ่ายทำตามเงื่อนไขอย่างถูกต้องที่ได้กำหนดไว้ในโปรโตคอลโดยอิงจากสมบัติแบบเดิม เพียงแต่เป็นการตรวจสอบบนกุญแจสาธารณะอีกรูปแบบหนึ่งที่ถูกบีบอัด และไม่เสียเวลาในการคำนวณเพื่อตรวจสอบมากเกินไปเท่านั้น

บทที่ 4
ผลการศึกษาค้นคว้า

ในบทนี้จะพูดถึงผลลัพธ์ที่เกิดขึ้นบนตัวโปรโตคอลหลังจากที่ได้เพิ่มกระบวนการตรวจสอบกุญแจสาธารณะเข้าไป โดยเราจะมีการอภิปรายผลการศึกษาได้ดังนี้

4.1 ผลลัพธ์การค้นคว้า

จากการศึกษาเราพบว่าการตรวจสอบกุญแจสาธารณะในรูปแบบที่ถูกบีบอัดจนเหลือเพียงค่า $pk_A = (bit, t_1, t_2, t_3, A, r_1, r_2)$ และ $pk_B = (bit, t_1, t_2, t_3, A, s, r)$ สามารถทำได้จริง โดยส่วนการตรวจสอบของค่า A นั้นยังคงใช้เวลานาน เพราะเป็นการตรวจสอบบน F_{p^2} และต้องใช้การสุ่มจุด P ขึ้นมา รวมถึงกระบวนการจำนวนมากบน elliptic curve ที่คูณด้วยจำนวนเลขขนาดใหญ่ l^e ซึ่งถือเป็น bottle-neck ของกระบวนการตรวจสอบในขณะนี้ และไม่ได้มีการพัฒนาจากกระบวนการเดิมมากนัก เพียงแต่ต้องเพิ่มการตรวจสอบมากขึ้นให้รองรับกับรูปแบบของกุญแจสาธารณะในปัจจุบันคือในเรื่องของ QR, QNR

ในขณะที่ส่วนของการตรวจสอบ s, r หรือ r_1, r_2 นั้นใช้เวลาเพียงเล็กน้อย เพราะแม้จะเป็นการคำนวณที่เกี่ยวข้องกับตัวเลขขนาดใหญ่ แต่เป็นเพียงกระบวนการบนจำนวนเต็มอยู่ โดยเราไม่ได้เสียเวลาในการคำนวณมากนัก และกระบวนการตรวจสอบก็เป็นเพียงแค่การตรวจสอบค่าของความเป็น QR, QNR ว่าถูกต้องหรือไม่โดยใช้การคูณและการหารเพียงไม่กี่ครั้ง

ส่วนสุดท้ายก็คือการตรวจสอบค่า bit, t_1, t_2, t_3 ที่เป็นการตรวจสอบสมบัติของจุดที่เดิมเราจะต้องใช้การคูณด้วย l^e กับจุดสองจุดในการตรวจสอบ ซึ่งพอเราปรับค่ามาแล้วแม้ว่าจะมีค่าที่ต้องตรวจสอบมากขึ้น แต่กระบวนการตรวจสอบเกิดขึ้นเพียงการตรวจสอบความสามารถของการหารลงตัว การตรวจสอบว่าค่าไม่เท่ากัน และสมบัติของความ congruence ของค่า t_i ใด ๆ เท่านั้น ทำให้สามารถคำนวณได้อย่างรวดเร็วเมื่อค่าเหล่านั้นเป็นสมาชิกของ Z_n ซึ่งถือเป็นสิ่งที่เราพัฒนามาได้ดีกว่าวิธีตรวจสอบแบบดั้งเดิมอย่างมากเทียบกับการตรวจสมบัติเหล่านั้นบนจุด P, Q

4.2 เทียบจำนวนรอบ CPU ที่ใช้กับตัวซอฟต์แวร์เดิมที่ไม่มีการตรวจกุญแจสาธารณะ

เราทำการนำเสนอจำนวนรอบ CPU เปรียบเทียบระหว่างกระบวนการภายในโปรโตคอลที่เกิดขึ้นเทียบกับการที่ไม่มีการตรวจสอบใด ๆ เลยในตอนแรก โดยหน่วยที่ใช้ในตารางจะอยู่ในหลักล้านรอบของ

CPU (Million CPU clock cycles เช่น 1.3M = 1,300,000 รอบของ CPU) ซึ่งผลลัพธ์จากการทดสอบสามารถดูได้ในตารางที่ 4.1 เป็นผลมาจากการเฉลี่ยจำนวนรอบของ CPU ที่ใช้จากกระบวนการของโปรโตคอลจำนวน 100 รอบ

Protocol	**Operation**	**No validation**	**Have validation on**			
			Curve	**Basis parameter**	**Point**	**All**
SIKE	Key Generation	683.8 M	667.0 M	668.8 M	676.0 M	667.6 M
	Encapsulation	871.7 M	939.6 M	856.8 M	849.7 M	945.2 M
	Decapsulation	801.9 M	874.7 M	799.3 M	785.0 M	881.0 M
	Total	**2357.4 M**	**2481.3 M**	**2324.9 M**	**2310.7 M**	**2493.8 M**
SIDH	Alice's key generation	656.0 M	641.8 M	641.3 M	693.4 M	690.7 M
	Bob's key generation	555.7 M	550.8 M	567.1 M	549.2 M	554.4 M
	Alice's shared secret computation	235.3 M	288.1 M	241.1 M	235.0 M	334.3 M
	Bob's shared secret computation	295.7 M	349.4 M	302.0 M	295.7 M	399.0 M
	Total	**1742.7 M**	**1830.1 M**	**1751.5 M**	**1773.3 M**	**1978.4 M**

ตารางที่ 4.1 ผลการเปรียบเทียบรอบ CPU เมื่อเพิ่มวิธีตรวจสอบกุญแจสาธารณะ

ผลลัพธ์ที่คาดการณ์ไว้คือส่วนของการทำ key generation ทั้งของโปรโตคอล SIDH, SIKE ไม่ควรจะมีการเปลี่ยนแปลงมากนัก ขึ้นกับค่าความแปรปรวนในรอบการคำนวณของโปรโตคอลเท่านั้น ที่เป็นไปตามที่คาดการณ์ไว้ โดยที่ใน SIKE จะอยู่ที่ประมาณ 660 ถึง 680 ล้านรอบ ในขณะที่ส่วนของ SIDH $Alice$ ใช้ประมาณ 640 ถึง 690 ล้านรอบ ในขณะที่ Bob ใช้ประมาณ 540 ถึง 560 ล้านรอบ

ส่วนถัดมาคือการคำนวณหาความลับร่วมกัน ที่เราพบว่าหากมีการตรวจเพียงค่าที่ใช้ในการสร้าง basis R_1, R_2 หรือตรวจเพียงแค่ bit, t_1, t_2, t_3 ที่เรานำมาใช้สร้างจุดนั้นแทบจะไม่เกิดผลกระทบต่อจำนวนรอบการทำงานเดิม โดยอยู่ในช่วงของความแปรปรวนของรอบ CPU ที่ใช้ แต่หากเราเพิ่มการ

ตรวจสอบ Curve parameter A เข้าไป จะส่งผลต่อรอบการทำงานอย่างมาก โดยที่ในกระบวนการ encapsulation นั้นเพิ่มมาประมาณ 60-70 ล้านรอบ ในขณะที่ส่วนของ decapsulation นั้นเพิ่มมาที่ประมาณ 60-70 ล้านรอบเช่นเดียวกัน ในขณะที่ถ้าเรามองในส่วนของโปรโตคอล SIDH จำพบว่าการคำนวณของ $Alice$ และ Bob นั้นเพิ่มมาประมาณ 50 ล้านรอบเช่นเดียวกัน

สุดท้ายคือเมื่อรวมทุกกระบวนการเข้าด้วยกัน เราพบว่าผลกระทบของโปรโตคอล SIDH นั้นเพิ่มมาถึง 230 ล้านรอบ คิดเป็นประมาณ 15% ของรอบการทำงาน CPU แบบเดิม ในขณะที่ถ้ามองทั้งโปรโตคอล SIKE นั้นเพิ่มมาที่ประมาณ 150 ล้านรอบเท่านั้น หรือประมาณ 6% ของรอบการทำงาน CPU ในแบบเดิม โดยเราคาดว่าอาจจะเกิดจากความแปรปรวนในเรื่องของกระบวนการในส่วนของ public key encryption และในส่วนของ key encapsulation นั้นเองที่ทำให้มีความคลาดเคลื่อนถึงระดับนี้

4.3 จำนวนรอบ CPU ในการสร้างความลับร่วมกัน 100 รอบ เมื่อกุญแจสาธารณะผิดจากข้อกำหนด

ในส่วนที่แล้วเราได้วัดผลว่าการเพิ่มตัวส่วนของการตรวจสอบกุญแจสาธารณะนั้นส่งผลต่อรอบ CPU เฉลี่ยในการใช้โปรโตคอลแต่ละครั้งอย่างไรบ้างเมื่อกุญแจสาธารณะนั้นยังถูกต้องเหมือนกรณีทั่วไปอยู่ แต่ในส่วนนี้เราจะพูดถึงว่าในกรณีที่กุญแจสาธารณะที่ได้รับนั้นผิดไปจากรูปแบบที่ได้กำหนดไว้ ทำให้เราสามารถหยุดโปรโตคอลล่วงหน้าแทนที่จะต้องทำงานต่อไปทั้งรอบได้จะช่วยเพิ่มความเร็วในการทำงานสำหรับการสร้างความลับร่วมกันหลายครั้งหรือไม่ หรือก็คือใช้ประเมินผลว่าเราควรคาดการณ์ว่าตัวโปรโตคอลจะเผชิญหน้ากับผู้ไม่ประสงค์ดีคิดเป็นเปอร์เซ็นต์เท่าใดแล้วจึงคุ้มที่จะใช้ตัวกระบวนการตรวจสอบกุญแจสาธารณะ

ในส่วนของเปอร์เซ็นต์กุญแจสาธารณะที่ผิดรูปแบบนั้น ในตัวทดสอบเราใช้การตรวจสอบอย่างง่าย คือ สำหรับรอบการทำงานที่ i เราทำการเช็คว่า $0 \equiv i \bmod n$ โดยที่ $n \in \{20, 10, 5, 4, 3, 2\}$ สำหรับการทดสอบของรูปแบบ 5, 10, 20, 25, 33, 50% ตามลำดับ เช่นสำหรับแถวของ 10% ทุกรอบการทำงานทดสอบครั้งที่ 0, 10, 20, ..., 90 จะเป็นรอบที่ตัวกุญแจสาธารณะถูกปรับแต่งนั่นเอง

เราจะแบ่งเป็นสองกรณีคือ กรณีที่ $Alice$ ส่งกุญแจสาธารณะที่ผิดมาให้ ทำให้ตรวจเจอได้ตั้งแต่ส่วนของกระบวนการ encapsulation กับอีกกรณีคือ Bob ส่งกุญแจสาธารณะที่ผิดมาให้ ทำให้ตรวจเจอได้ในส่วนของกระบวนการ decapsulation ซึ่งทั้งสองกรณีนี้ทำให้ประโยชน์ที่ได้จากการตรวจสอบกุญแจสาธารณะในเรื่องของความเร็วที่มากขึ้นนั้นแตกต่างกันเล็กน้อย โดยที่การตรวจเจอตั้งแต่ขั้นตอน encapsulation จะทำให้ได้ผลลัพธ์ที่ดีกว่า

4.3.1 จำนวนรอบ CPU กรณีที่กุญแจสาธารณะของ $Alice$ ถูกแก้ไข

ในหัวข้อนี้คือการตรวจเจอข้อผิดพลาดที่ขั้นตอน encapsulation สำหรับ SIKE โดยผลลัพธ์ของจำนวนรอบ CPU สามารถดูได้ในตารางที่ 4.2

Protocol	Percentage of invalid public key	No validation	With Validation
SIKE	0%	235,109 M	246,124 M
	5%		241,766 M
	10%		234,876 M
	20%		221,516 M
	25%		219,590 M
	33%		206,311 M
	50%		190,065 M

ตารางที่ 4.2 ผลลัพธ์การเปรียบเทียบรอบ CPU เมื่อกุญแจสาธารณะของ $Alice$ ไม่ถูกต้อง

4.3.2 จำนวนรอบ CPU กรณีที่กุญแจสาธารณะของ Bob ถูกแก้ไข

ในหัวข้อนี้คือการตรวจเจอข้อผิดพลาดที่ขั้นตอน decapsulation สำหรับ SIKE โดยผลลัพธ์ของจำนวนรอบ CPU สามารถดูได้ในตารางที่ 4.3

Protocol	Percentage of invalid public key	No validation	With Validation
SIKE	0%	235,109 M	246,124 M
	5%		245,119 M
	10%		240,298 M
	20%		231,930 M
	25%		228,122 M
	33%		219,798 M
	50%		208,889 M

ตารางที่ 4.3 ผลลัพธ์การเปรียบเทียบรอบ CPU เมื่อกุญแจสาธารณะของ Bob ไม่ถูกต้อง

สุดท้ายเราเลยสามารถสรุปได้ว่าในกรณีทั่วไปที่กุญแจสาธารณะนั้นถูกปรับแต่งมาให้ผิดจากรูปแบบที่กำหนดไว้ ถ้าเราคาดการณ์ว่าตัวระบบของเรามีโอกาสมากกว่า 20% ที่จะเจอผู้ไม่ประสงค์ดีมาโจมตีระบบก็ควรที่จะเริ่มใช้กระบวนการตรวจสอบกุญแจสาธารณะแล้ว เพราะทำให้ตัวระบบสามารถทำงานได้เร็วขึ้น ในทั้งสองกรณีที่ตรวจพบในขั้นตอนที่ต่างกัน หรือในกรณีที่โดยมากแล้วเรามักจะทำตัวเป็น server หลัก ที่ส่วนใหญ่จะกำหนดให้ตัวเราเองนั้นเป็น $Alice$ และผู้ที่มาติดต่อกับเราเป็น Bob ทำให้การตรวจสอบว่าผู้ที่มาติดต่อกับ server เรานั้นมีการแก้ไขกุญแจสาธารณะหรือไม่จะตรวจพบในส่วนของขั้นตอน decapsulation ที่ผลลัพธ์จะเริ่มดีกว่ารูปแบบที่ไม่มีการตรวจสอบเลยเมื่อเราพบกุญแจสาธารณะที่ผิดรูปแบบมากกว่า 20% ขึ้นไปเท่านั้น ถ้าเราอยากพัฒนาให้เร็วขึ้นกว่านี้ก็สามารถตั้งค่าให้ตัวเรานั้นเป็น Bob เองก็ได้เช่นกัน ซึ่งในกรณีนี้จะตรวจพบในขั้นตอน encapsulation ทำให้ใช้ประโยชน์ของการตรวจสอบกุญแจสาธารณะในกรณีที่ตัวกุญแจสาธารณะถูกแก้ไขตั้งแต่ 10% ขึ้นไปนั่นเอง

บทที่ 5
ข้อสรุปและข้อเสนอแนะ

ในบทนี้จะพูดถึงข้อสรุปที่ได้จากการทำการศึกษาค้นคว้าและวิจัยในหัวข้อนี้ รวมถึงอภิปรายปัญหาที่พบเจอระหว่างการศึกษาค้นคว้า รวมไปถึงข้อเสนอแนะที่ผู้วิจัยคาดว่าน่าจะมีประโยชน์ในการศึกษาครั้งต่อไปในหัวข้อนี้สำหรับผู้วิจัยท่านอื่นที่ต้องการพัฒนาตัวโปรโตคอลให้ดียิ่งขึ้น

5.1 ข้อสรุปจากการศึกษา

การตรวจสอบกุญแจสาธารณะนั้นโดยรวมแล้วไม่ได้ทำให้โปรโตคอลทำงานช้าลงมากนัก เนื่องจากส่วนที่เกี่ยวของจะมีเพียงแค่การคำนวณความลับร่วมกันเท่านั้น ทีจะเกิดขึ้นในส่วนของ encapsulation, decapsulation เราวัดแล้วพบว่าเพิ่มมาประมาณ 70-80 ล้านรอบ CPU เท่านั้น แต่พอเทียบลงไปแค่ส่วนของการคำนวณ shared secret กลับพบว่าเพิ่มมาถึงประมาณ 100 ล้านรอบ ซึ่งเป็นไปได้สองอย่างคือ อาจจะเกิดจากความแปรผันของตัวโปรโตคอลนี้เอง ซึ่งอาจจะต้องทดสอบวัดรอบมากขึ้น เช่น 1,000 หรือ 10,000 ครั้ง (ในตัวรายงานนั้นเป็นผลลัพธ์เฉลี่ยจากการใช้งาน 500 รอบ) เพื่อให้ได้ค่าเฉลี่ยที่แม่นยำขึ้น หรือมองได้อีกอย่างคือกระบวนการนี้นั้นเป็นเพียงส่วนเล็กน้อยของภาพรวมโปรโตคอลซึ่งมีการคำนวณอื่นอีกมาก เช่นการแก้ปัญหา discrete logarithm ที่มองว่าเป็นส่วนที่เสียเวลานานสุด ทำให้แม้เราจะเพิ่มส่วนนี้เข้ามาก็ไม่ได้มีผลต่อความช้าลงของโปรโตคอลมากนัก

อีกข้อสรุปหนึ่งที่ได้พบคือจากการแยกตรวจทีละหัวข้อ เราพบว่าการตรวจสอบ curve parameter A นั้นเพิ่มจำนวนรอบ CPU มามากที่สุด ในขณะที่การตรวจสอบอื่นนั้นแทบจะไม่เพิ่มมาจากเดิมที่ไม่มีการตรวจสอบเลยเสียด้วยซ้ำ เพราะการตรวจค่า A เป็นกระบวนการตรวจสอบเดียวในนี้ที่ต้องยุ่งเกี่ยวกับสมาชิกที่เป็นจุดอยู่บน elliptic curve และกระบวนการทางคณิตศาสตร์จำนวนมากบนนั้น ซึ่งเป็นสิ่งที่เราคาดเดาได้ว่าน่าจะเป็นส่วนหลักที่ทำให้การตรวจสอบกุญแจสาธารณะนี้ทำได้ช้า ดังนั้นถ้าหากมีความกังวลเรื่องประสิทธิภาพความเร็วในการทำงานของโปรโตคอลที่เพิ่มวิธีการตรวจสอบกุญแจสาธารณะนี้ เราอาจจะตรวจเพียงแค่ส่วนที่เอามาสร้าง basis และค่าทีเกี่ยวข้องกับจุดก็เพียงพอก็ได้ โดยอาจจะละการตรวจค่า A ไปโดยแลกกับการลดความปลอดภัยลงระดับหนึ่ง แต่ได้ความเร็วในการทำงานที่เทียบเท่ากับรูปแบบเดิมที่ไม่มีการตรวจสอบกุญแจสาธารณะ

และข้อสรุปสุดท้ายคือ สำหรับการจะนำวิธีการนี้มาใช้จริงนั้นสามารถนำมาใช้ได้เลยในกรณีที่เราคาดเดาไว้ล่วงหน้าแล้วว่าตัวระบบที่เราจะนำโปรโตคอล SIKE/SIDH ไปใช้นั้นมีโอกาสหากคิดเป็นเปอร์เซ็นต์แล้วจะโดนการโจมตีที่ผ่านทางกุญแจสาธารณะที่ผิดรูปแบบเป็นเท่าใดบ้าง ซึ่งในกรณีทั่วไปคือ

การที่ตัวระบบเราเลือกเป็น $Alice$ จะสามารถใช้วิธีการตรวจสอบนี้ได้อย่างมีประสิทธิภาพเลยแม้โอกาสที่ถูกโจมตีจะมีเพียงประมาณ 10% ซึ่งเป็นผลลัพธ์ที่ได้มาจากในหัวข้อที่ 4.3 นี้เอง โดยยิ่งโอกาสถูกโจมตีมากก็ยิ่งทำให้กระบวนการนี้มีประสิทธิภาพมากขึ้นด้วย ดังที่เห็นได้ว่าสำหรับ SIKE นั้นแล้วจำนวนรอบ CPU ลดลงมาได้เหลือเพียง 70% หากเทียบกับระบบที่ไม่มีการตรวจสอบกุญแจสาธารณะสำหรับกรณีที่โอกาสถูกโจมตีผ่านทางกุญแจสาธารณะคือ 50% หรืออีกตัวอย่างหนึ่งคือจำนวนรอบ CPU ลดลงเหลือเพียง 85% ในกรณีที่โอกาสถูกโจมตีคือ 25% นั่นเอง

5.2 ปัญหาและอุปสรรคที่พบ

ปัญหาหลักที่พบคือ กว่าจะเริ่มเข้าใจกระบวนการขั้นตอนการทำงานของโปรโตคอล รวมไปถึงวิธีการตรวจสอบแบบเดิมว่าสมบัติที่ต้องมีการตรวจสอบคืออะไรบ้างนั้นต้องผ่านการอ่านงานวิจัยและหนังสือเพื่อปูพื้นฐานอย่างมาก ทำให้กว่าจะได้เริ่มลงมือทำว่าควรจะเริ่มจากตรงไหนนั้นเสียเวลาไปมากพอแล้ว และแม้ว่าจะเตรียมความรู้พื้นฐานมามากก็ยังเจอคำศัพท์และทฤษฎีที่ค่อนข้างทำความเข้าใจได้ยากจนกระทั้งได้เห็นตัวอย่างจริงว่าสามารถนำบททฤษฎีเหล่านั้นมาประยุกต์ใช้อย่างไรได้บ้าง

อีกปัญหาที่พบคือ ตัวโปรโตคอลในส่วนของ code นั้นทำความเข้าใจได้ยากมาก เพราะเต็มไปด้วยการเพิ่มประสิทธิภาพในหลายส่วน เช่น การบีบอัดข้อมูลที่ส่งให้อยู่ในรูปของ octet string ทำให้ตอนแรกการจะอ่านค่าหลายส่วนในโปรโตคอลนั้นทำได้ยาก และตัวข้อมูลเหล่านี้นั้นไม่ได้มีระบุในตัวงานวิจัยต่างอื่นแต่อยู่เพียงในรายงานฉบับเดียวคือ SIDH-spec [19] ที่บอกว่ามีการทำอะไรบ้าง ซึ่งตัวรายงานนั้นก็ไม่ได้ลงละเอียดในระดับที่จะเข้าใจทุกอย่างใน code ได้ แต่ก็ช่วยเป็นแนวทางให้สามารถเริ่มดำเนินการปรับแต่งแก้ไขโปรโตคอลให้สามารถทำงานได้ตามที่ต้องการในบางส่วน

ปัญหาสุดท้ายคือ ไม่สามารถทำการทดสอบได้ตามทุกอย่างที่คาดการณ์ไว้ โดยตอนแรกตั้งใจไว้ว่าจะทดสอบกับกุญแจสาธารณะที่ผิดรูปแบบบางเฉพาะได้ เช่นการแก้ค่า curve parameter โดยตรงให้กลายเป็น ordinary curve หรือการปรับให้ค่าที่มาจากตารางไม่ได้เป็นไปตามข้อกำหนดในส่วนของ QR, QNR สำหรับ basis parameter และสมบัติของค่า t_i สำหรับจุดที่เราใช้ตรวจสอบนั้นเอง แต่กระบวนการที่ใช้ในการแก้ไขกุญแจสาธารณะให้ผิดรูปแบบคือแค่ลองสุ่มค่าเช่น เลื่อน bit, shift bit ที่ถูก encode ในรูปของตัว octet string ในตัวโปรโตคอลนั้นเอง ซึ่งทำให้กลายเป็นกุญแจสาธารณะที่ไม่ถูกรูปแบบได้เช่นกันเพียงแต่ว่าต่างกับที่คาดการณ์ไว้ในตอนแรกว่าต้องการแก้ทีละส่วนโดยตรงเลย

5.3 ข้อเสนอแนะ

ควรเตรียมตัวให้มั่นใจก่อนว่าจะสามารถทำงานวิจัยชิ้นนี้ได้จริง เพราะตอนแรกที่คิดว่าจะทำนั้นไม่ได้คาดการณ์ว่าจะมีความยากในระดับนี้ การกำหนดขอบเขตงานวิจัยให้ชัดเจนตั้งแต่แรกเริ่มว่าจะทำอะไรบ้าง หรืออะไรที่สามารถทำได้บ้างนั้นเป็นสิ่งที่สำคัญ

รายการอ้างอิง

[1] Miller, Victor S. "The Weil pairing, and its efficient calculation." Journal of cryptology 17.4 (2004): 235-261.

[2] Washington, Lawrence C. Elliptic curves: number theory and cryptography. CRC press, 2008.

[3] Jao, David, and Luca De Feo. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies." International Workshop on Post-Quantum Cryptography. Springer, Berlin, Heidelberg, 2011.

[4] Sutherland, Andrew V. "Identifying supersingular elliptic curves." LMS Journal of Computation and Mathematics 15 (2012): 317-325.

[5] Costello, Craig, Patrick Longa, and Michael Naehrig. "Efficient algorithms for supersingular isogeny Diffie-Hellman." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2016.

[6] Azarderakhsh, Reza, et al. "Key compression for isogeny-based cryptosystems." Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. 2016.

[7] Galbraith, Steven D., et al. "On the security of supersingular isogeny cryptosystems." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2016.

[8] De Feo, Luca. "Mathematics of isogeny based cryptography." arXiv preprint arXiv:1711.04062 (2017).

[9] Costello, Craig, et al. "Efficient compression of SIDH public keys." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2017.

[10] Hofheinz, Dennis, Kathrin Hövelmanns, and Eike Kiltz. "A modular analysis of the Fujisaki-Okamoto transformation." Theory of Cryptography Conference. Springer, Cham, 2017.

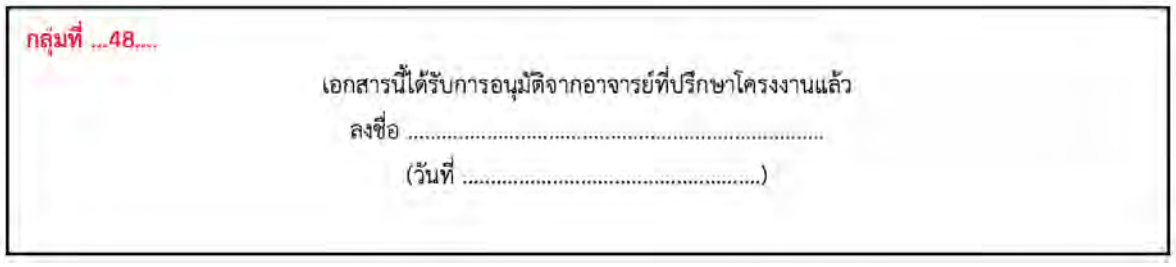
[11] Galbraith, Steven D., and Frederik Vercauteren. "Computational problems in supersingular elliptic curve isogenies." Quantum Information Processing 17.10 (2018): 265.

[12] Zanon, Gustavo HM, et al. "Faster key compression for isogeny-based cryptosystems." IEEE Transactions on Computers 68.5 (2018): 688-701.
[13] Urbanik, David, and David Jao. "SoK: The problem landscape of SIDH." Proceedings of the 5th ACM on ASIA Public-Key Cryptography Workshop. 2018.
[14] Costello, Craig, et al. "Improved Classical Cryptanalysis of the Computational Supersingular Isogeny Problem." IACR Cryptology ePrint Archive 2019 (2019): 298.
[15] Martindale, Chloe, and Lorenz Panny. How to not break SIDH. Cryptology ePrint Archive, Report 2019/558, 2019, https://eprint.iacr.org/2019/558.
[16] Costello, Craig. "Supersingular Isogeny Key Exchange for Beginners." International Conference on Selected Areas in Cryptography. Springer, Cham, 2019.
[17] David Jao. SIDH-Spec[Online]. Available from: https://sike.org/files/SIDH-spec.pdf, 2019.
[18] Brown, Ezra, Eric Errthum, and David Fu. "Weil Pairing vs. Tate Pairing in IBE systems." (2003).
[19] David Jao, et al. SIKE – Supersingular Isogeny Key Encapsulation[Online]. Available from: https://sike.org, 2020.

ภาคผนวก

ภาคผนวก ก

แบบเสนอหัวข้อโครงงาน รายวิชา 2301399 Project Proposal


กลุ่มที่ ...48....

เอกสารนี้ได้รับการอนุมัติจากอาจารย์ที่ปรึกษาโครงงานแล้ว

ลงชื่อ .....

(วันที่ .....)

The Project Proposal of Course 2301399 Project Proposal

Academic Year 2019

Project Title (Thai) การตรวจสอบกุญแจเข้ารหัสสาธารณะบนโปรโตคอลการแลกเปลี่ยนกุญแจแบบ SIDH

Project Title (English) Public Key Validation on SIDH Key Exchange Protocol

Project Advisor Wutichai Chongchitmate

By Disorn Chaladtanyakij ID 5933625023

Computer Science Program, Department of Mathematics and Computer Science

Faculty of Science, Chulalongkorn University

Background and Rationale

Public key exchange (also known as key establishment) is a way for both parties (named Alice and Bob) to exchange their own public key (pk_A, pk_B) constructed from each parties secret key (sk_A, sk_B) which is then used to generate their shared secret for establishing a secure communication channel that no other eavesdropper (named Eve) can obtain a copy of shared secret. This shared secret is then used to generate secret key for symmetric-key cryptography system such as block ciphers which utilize single key to encrypt and decrypt messages. Such protocol has been invented long ago called Diffie-Hellman key exchange. However, there is a problem when public key of each parties was modified by

malicious users. This modified key might led to Eve able to obtain Alice and Bob shared secret or one of their secret keys. It could also potentially slow down the server because it has to perform lots of arithmetic operations on invalid inputs. Furthermore, either Alice or Bob public key could break the security of key exchange scheme. If their secret keys is not generated properly or generated from weak parameters (says, prime (p, q) must be large in order for Integer factorization to be hard problem, Bob decided to use (2, 3)). This is why there must be a way for Alice and Bob to validate key whenever they are receiving a key from another party.

The protocol we are interested in is called Supersingular isogeny Diffie-Hellman key exchange (SIDH). It is based on the supersingular isogeny walk problem, given Supersingular Elliptic Curves E_1 and E_2 with the same number of points. Finding the map that takes E_1 to E_2 is suggested to take $O(p^{1/4})$ for Classical computer and $O(p^{1/6})$ for Quantum computers [8]. This means that choosing prime p with 768-bit for SIDH will have a security for Quantum computers comparable to 128-bit security system.

The first proposal of how SIDH protocol would work started in 2011 with [8], then in 2016 Microsoft Researcher improved its performance with various techniques and suggested a way to validate public key of SIDH in [9](Section 9). This process of validation consists of validate the curve parameter A, point (x_P, x_Q) test for full order l^e and Weil Pairing check of points P and Q. The following papers during 2016 - 2017 focused more on compression of public key which changed the public key in the form of (x_P, x_Q, x_R) into $(bit, t_1, t_2, t_3, A, s, r)$ ([4], [5] and [6]). The compressed key can be decompressed back and then used the same validation techniques as normal public key. But this method requires (A, s, r) to be ephemerally generated from Alice and Bob in the first place before they retrieve uncompressed key back. This suffered the same problem that Eve could intercept and modified (A, s, r) to be malicious input for Public key decompression Algorithm[3](Section 1.5.2).

The public key validation that we have talked in previous section however, has seen no used since 2017 with the introduction of Supersingular Isogeny Key Encapsulation (SIKE) [2]. This is the transformation of SIDH with indirect key validation[12] by Dennis Hofheinz,

Kathrin Hövelmanns, Eike Kiltz in [10]. The main purpose of this scheme is to ensure that the shared secret that is generated at the end will be the same when public key were honestly generated as specified by the protocols. Otherwise Alice and Bob will end up with different shared secret and requires to generate new shared secret. This scheme also provides resistant against some attacks that were once a threat to SIDH scheme [11].

It makes sense that direct public key validation is not needed anymore with the introduction of SIKE. Since the purpose of SIDH at first is to establish shared secret and we can just re-establish shared secret one more time if there were a detection of malicious users trying to attack SIKE. Also the cost of validation techniques is non-negligible in the first place, convincing that we should only do indirect public key validation.

However, we observe that this view only work with non-compressed public key counterpart of the SIDH/SIKE. Since the cost of public key decompression is one of the bottlenecks in compressed SIKE scheme. If we were to compute all of the arithmetic operations under the key encapsulation mechanism on an invalid public key, this could potentially slow down a server since it only knows that shared secret is not the same for Alice and Bob at the end of SIKE scheme.

With this observation, we concluded that direct public key validation is still needed for the compressed public key counterpart, and the validation also need to work with just $(bit, t_1, t_2, t_3, A, s, r)$ received from public key and does not have to go through key decompression algorithm. We also need to check that the runtime of the algorithm we proposed is better than leaving the invalid input go through key encapsulation mechanism and how the proposed technique slow down the protocol compares to no direct public key validation.

Objectives

To design an algorithm for both parties to validate whether or not their compressed public key received from another party is valid, then implement this algorithm to existed SIDH/SIKE protocol in order to monitor performance impact from this algorithm and compare to a full run of key encapsulation mechanism.

Scope

The proposed algorithm aims to validate a public key in the compressed form: whether $(bit, t_1, t_2, t_3, A, s, r)$ is valid or not. This procedure happens twice: first during encapsulation mechanism where Bob receives Alice's public key (pk_A) before the encryption algorithm and second during decapsulation mechanism where Alice receives Bob's public key (pk_B) before the decryption algorithm with this following restriction to keep in mind.

1. The baseline code is from [2] Optimized implementation, this consist of parameter setting, field operation, key encapsulation mechanism and key exchange procedure. The proposed algorithm will be compatible with this implementation.
2. The proposed algorithm runs and tests on UNIX based Operating System (Linux, macOS).
3. The protocol requires CMake version 3.5 or later and GMP version 6.1.2 or later with any C99-compatible compiler (Test on Clang and GCC).
4. The impact to performance metric (CPU clocks count) by this algorithm should not increase more than a factor of 2 against no validation variance.
5. The accuracy of validation on public key in compressed form with proposed algorithm should be similar to validate on public key in normal form with algorithm in [9].

Project Activities

Activity	2019					2020			
	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	April
1. Study related work.									
2. Study math related to isogeny graphs and elliptic curves									
3. Test original validation.									

4. Analyze algebraic structure and design new validation algorithm.									
5. Implement new algorithm.									
6. Monitor new algorithm impact.									
7. Prepare document.									

Benefits

Benefits for the student

1. Learn more about computational algebra.
2. Learn more about arithmetic computation on C language.
3. Gain experience on implementing something on cryptographic protocol.
4. Gain experience on having to work with multiple people in fields of cryptography.

Benefits of the project

1. Proposed an algorithm for direct public key validation for Compressed SIDH Key exchange.
2. Might introduce a new scheme which does not rely on key encapsulation to remain safe against active attack while also able to validate the key as we proposed.

Equipment

1. Personal Notebook: Apple MacBook Pro 13” Early 2015
    a. Intel® Core™ i5-5257U 2.70 GHz
    b. 8 GB 1866 MHz LPDDR3 RAM
    c. 256 GB PCIe Solid State Drive
    d. Integrated Intel Iris Graphics 6100

2. Personal Computer

. Intel® Core™ i5-9400F 2.90 GHz 6 Core 6 Thread

a. 16 GB 2400 MHz DDR4 RAM

b. 250 GB PCIe NVMe M.2 Solid State Drive

c. NVIDIA GeForce GTX 1070 Ti 8GB GDDR5

Budget

1. MacBook Pro 2015 Battery replacement		4700฿
	Total	4700฿

References and Bibliography

[1] Post-Quantum Cryptography, Round 2 Submissions[Online]. Available from: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 22 October 2019

[2] David Jao, et al. SIKE – Supersingular Isogeny Key Encapsulation[Online]. Available from: https://sike.org, 12 November 2019

[3] David Jao. SIDH-Spec[Online]. Available from: https://sike.org/files/SIDH-spec.pdf, 17 April 2019

[4] Azarderakhsh, Reza, et al. "Key compression for isogeny-based cryptosystems." Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. ACM, 2016.

[5] Costello, Craig, et al. "Efficient compression of SIDH public keys." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2017.

[6] Zanon, Gustavo HM, et al. "Faster key compression for isogeny-based cryptosystems." IEEE Transactions on Computers 68.5 (2018): 688-701.

[7] Naehrig, Michael, and Joost Renes. "Dual Isogenies and Their Application to Public-key Compression for Isogeny-based Cryptography."

[8] Jao, David, and Luca De Feo. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies." International Workshop on Post-Quantum Cryptography. Springer, Berlin, Heidelberg, 2011.

[9] Costello, Craig, Patrick Longa, and Michael Naehrig. "Efficient algorithms for supersingular isogeny Diffie-Hellman." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2016.

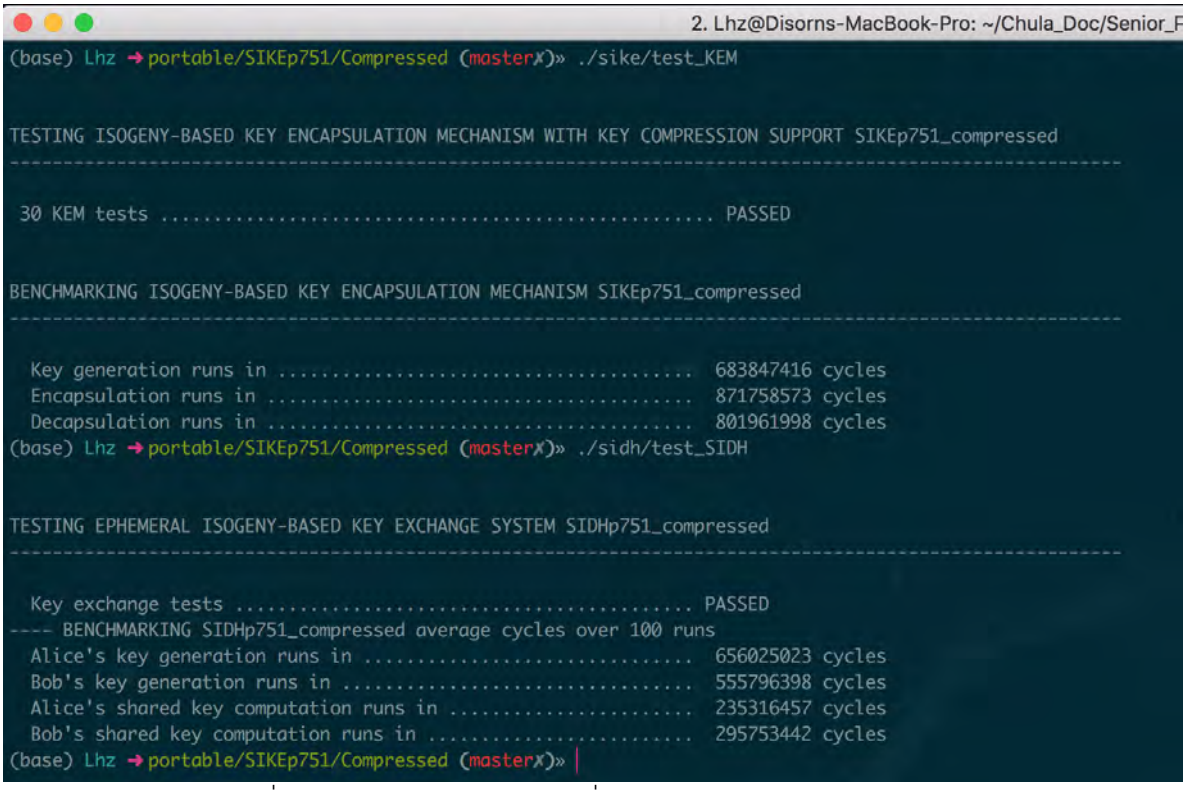
[10] Hofheinz, Dennis, Kathrin Hövelmanns, and Eike Kiltz. "A modular analysis of the Fujisaki-Okamoto transformation." Theory of Cryptography Conference. Springer, Cham, 2017.

[11] Galbraith, Steven D., et al. "On the security of supersingular isogeny cryptosystems." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, 2016.

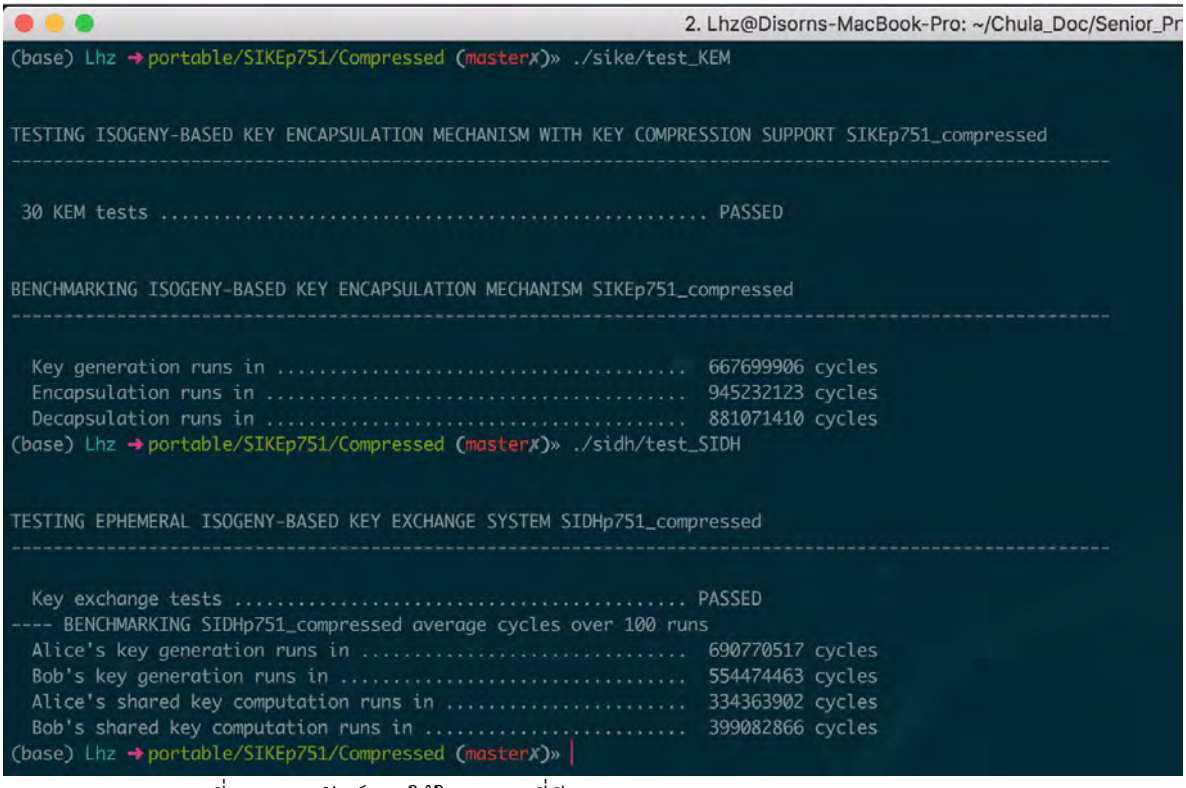
[12] D. Kirkwood, B. C. Lackey, J. McVey, M. Motley, J. A. Solinas, and D. Tuller. Failure is not an option: Standardization issues for post-quantum key agreement[Online]. Available from: http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm, April 2015

ภาคผนวก ข

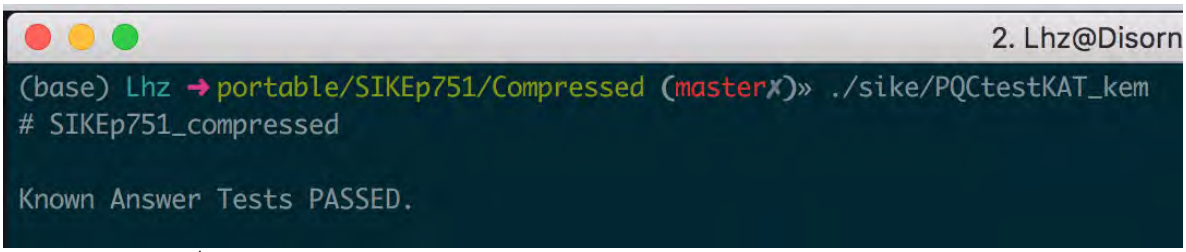
ตัวอย่างการใช้งานโปรโตคอล SIDH/SIKE



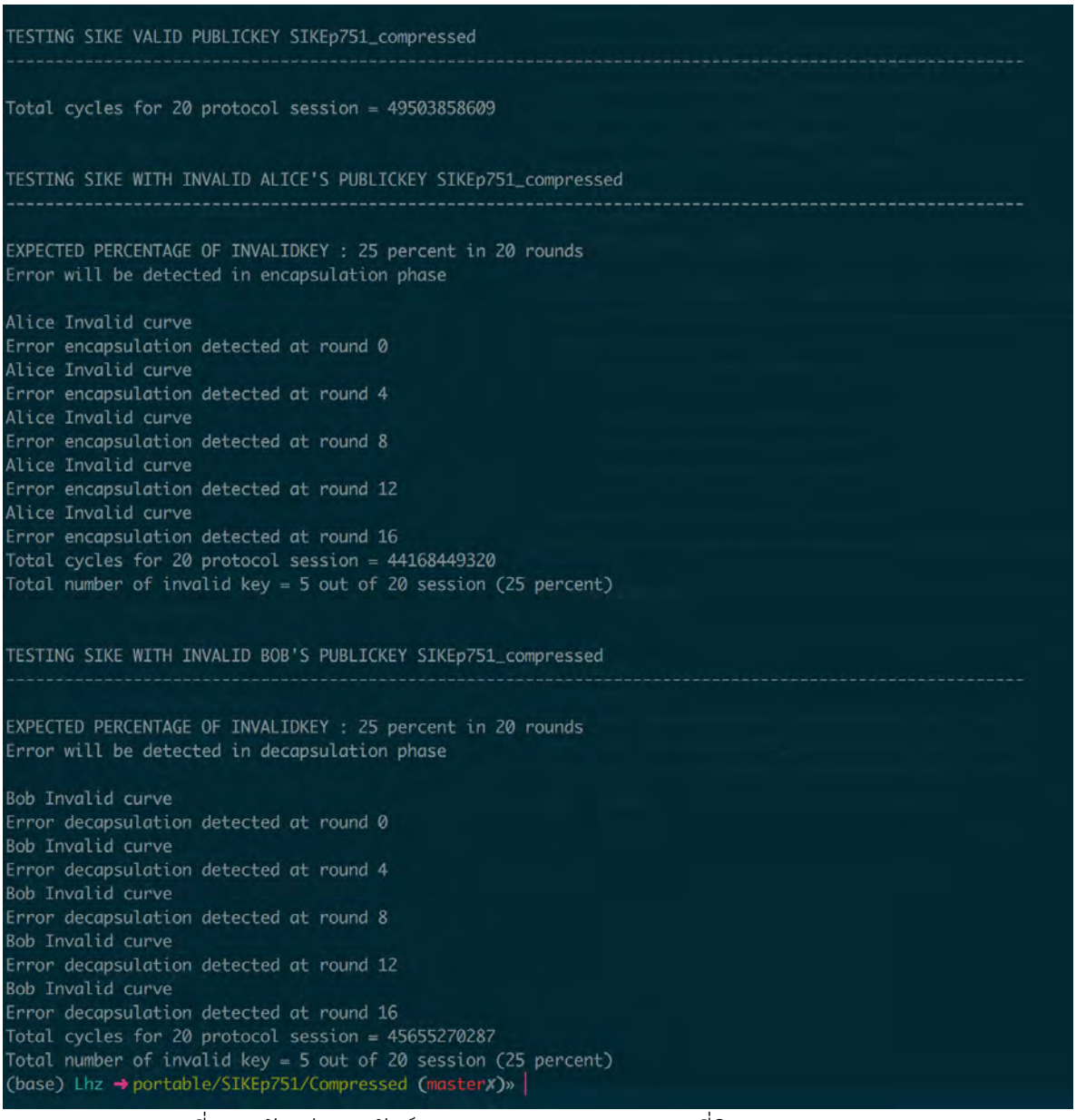
ภาพที่ ข.1 ผลลัพธ์การใช้ในระบบที่ไม่มีการตรวจสอบกุญแจสาธารณะ



ภาพที่ ข.2 ผลลัพธ์การใช้ในระบบที่มีการตรวจสอบกุญแจสาธารณะทุกประเภท



ภาพที่ ข.3 ตรวจสอบความถูกต้องของโปรโตคอลด้วย Known answer test (KAT)



ภาพที่ ข.4 ตัวอย่างผลลัพธ์การทดสอบกุญแจสาธารณะที่ผิดประเภท (20 รอบ)

ประวัติผู้เขียน

Mr. Disorn Chaladtanyakij
นาย ดิศรณ์ ฉลาดธัญญกิจ
เบอร์โทรศัพท์ 0878033375
อีเมล: randgris@protonmail.ch
รหัสนิสิต 5933625023
สาขาวิชา วิทยาการคอมพิวเตอร์ ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย