JUNIPER NETWORKS

DAY ONE: QUANTUM-SAFE IPSEC VPNS

IPsec VPNs Enhanced with RFC8784 Quantum Cryptography

By Melchior Aelmans, Gert Grammel, Siji Joseph, Sabyasachi Mukhopadhyay, Priyabrata Saha, Ranjan Sinha, Aswin Surendran

DAY ONE: QUANTUM-SAFE IPSEC VPNs

In a classical computing environment, calculating discrete logarithms for sufficiently large prime numbers is computationally infeasible, which makes the Diffie-Hellman algorithm a secure method for key exchange. However, quantum computers use a different approach to computation, based on quantum bits or qubits, which allows them to perform certain calculations much faster than classical computers. One of the computations that quantum computers excel at is the calculation of discrete logarithms, potentially allowing them to break the security of the Diffie-Hellman key exchange algorithm.

Welcome to the world of Shor's Algorithm, superposition, entanglement, and Quantum Key Distribution (QKDs). This book provides a comprehensive understanding of the threats posed by quantum computing to IPsec. The stellar author list includes some of the finest engineers at Juniper Networks as they convey both new knowledge and the right tools needed to protect you and your network from quantum attack.

"Quantum computing promises to herald the next major revolution in computing. Personally, for me, its application to networking and security is the most practical application in the shorter term. Day One: Quantum-Safe IPSEC VPNs *is a great way to impart solid understanding of applying quantum technology to secure communication and safe key exchanges. My thanks to some of our finest leaders who form the author list for the book. Enjoy!"*

Raj Yavatkar, Chief Technology Officer, Juniper Networks

"Feynman, a Nobel Prize winner for his work on QED, said: "I think I can safely say that nobody understands quantum mechanics." Einstein called quantum entanglement (crucial to quantum networking) spukhafte Fernwirkung (spooky action at a distance). Schroedinger spoke of a cat both dead and alive. All this may lead you to think quantum networking is beyond you and you won't even try. WAIT! Here is a practical book that tells you why you need to learn this and it gives you very practical steps on how to get into the technology. It includes hands-on training on integrating quantum-safe methods into your network and it is explained by the best in the business. Give it a shot; you won't regret it."

Kireeti Kompella, SVP Architecture, Juniper Networks

"Quantum technology provides new and innovative tool kits to apply for solutions in multiple fields of science and engineering, in particular computing, security and networking. For me, it's a fascinating discipline that allows you to explore new solutions thinking out of our classical and deterministic box."

Domenico Di Mola, GVP Optical, Juniper Networks

Day One: Quantum-safe IPsec VPNS

By Melchior Aelmans, Gert Grammel, Siji Joseph, Sabyasachi Mukhopadhyay, Priyabrata Saha, Ranjan Sinha, and Aswin Surendran

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Melchior Aelmans, Gert Grammel, Siji Joseph, Sabyasachi Mukhopadhyay, Priyabrata Saha, Ranjan Sinha, Aswin Surendran
Editor in Chief: Patrick Ames
Version History: v1, April 2023

About the Authors

Melchior Aelmans is a seasoned architect with over 20 years of experience in various operations, engineering, and architecture roles in the Cloud- and Service Provider industry. He is currently a member of the Global Service Provider architecture team at Juniper Networks, where he collaborates with Service Providers to design and evolve their networks. Melchior has a particular interest in the intersection of quantum technology and network security and is focused on exploring ways to apply quantum technology in this area since 2018. Aside from his work in quantum technology, Melchior is passionate about routing protocols standards development, routing security and network architectures. He is a frequent presenter and attendee at conferences and meetings, including active participation in standardization organizations such as IETF. As a Steering Committee member at MANRS, Melchior is committed to promoting security, reliability, and stability of the Internet. He also serves as Internet Society Organization Member Advisory Council co-chair, member of the NANOG Program Committee and a board member at the NLNOG Foundation.

Gert Grammel is a Principal Engineer at Juniper Networks where he has been working with operators on automation, optical technology, and evolution of networks. He has 25 years of experience in various management and engineering positions working with Cloud, Data Centers, and Service Providers. He enjoys working on networking technology and control architecture. Since 2020 Gert is spearheading Juniper's activities in exploring the Quantum Internet. He participates in IETF, OpenROADM, ETSI-QKD and ITU-T, is a regular presenter at conferences and chairs the Physical Simulation Environment (PSE) Working Group in www.telecominfraproject.com.

Siji Joseph is a Staff Engineer at Juniper Network working with the protocol testing team. She has over 15 years of experience in testing different protocols and VPN implementations.

Sabyasachi Mukhopadhyay has been in the Networking and Telecommunication Industry for about 25 years, working in both Development and Test teams on diverse technologies like ATM, SDH, MPLS-VPN, SDN. He currently leads the AI/ML team in Juniper CDO org where he Develops AI/ML based features for Service Provider customers. A graduate in Physics/ Optics and Optoelectronics and Masters in AI, his interests spans across Quantum Networks, and the application of AI to quantum. Apart from his work in quantum, Sabyasachi is passionate about AI, NLP and application of AI and NLP to solve critical customer issues.

Ranjan Sinha has been working in the security domain for more than 20 years working on software licensing, encrypted file systems ,and VPNs. He currently leads the IPsec VPN team at Juniper Networks.

Aswin Surendran is a Software Engineer at Juniper Networks. He has been working in the field of Networking and Telecom-munication for over 8 years. He has experience in developing 4G and 5G technologies. His current major focus is on the design and development of Network security using IPsec, SSL, etc.

Priyabrata Saha is a Software Engineer at Juniper Networks working in the IPsec VPN team. With an overall experience of around 3 years, his main area of focus is designing and developing highly efficient and scalable software products. He has worked on multiple security products including but not limited to VPN, IDS, IPS, UTM, etc.

Authors' Acknowledgments

The authors would like to thank, in random order, the following people for their ideas, contributions and support: Kireeti Kompella, Domenico Di Mola, Pavan Kurapati, Raj Yavatkar, Patrick Ames, Neetu Rathee, Ashwin Kovummal, Mark Denny, Johan Andersson, Bruno Rijsman, Yasir Nawaz, and many others who have been of instrumental help.

Melchior Aelmans would like to express his heartfelt appreciation to Patrick Ames for his unwavering dedication to the *Day One* book program and for collaborating with him on numerous books over the years. After publishing this book, Patrick will be embarking on a well-deserved retirement. Patrick, you will be greatly missed. Thank you for everything!

Melchior also wishes to express his gratitude to the RIPE NCC and QuTech for organizing the Quantum Internet hackathon back in 2018. This event marked the beginning of his journey into the world of quantum.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books. *Day One* books feature Juniper Networks technology with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

- Download a free PDF edition at https://www.juniper.net/dayone.
- Purchase the paper edition at Vervante Corporation (www.vervante.com).

Key Quantum Network Resources

The authors highly recommend the following documents if you are interested in reading more about quantum networks.

- Whitepaper - Validation of a Quantum Safe MACsec Implementation
  https://www.juniper.net/content/dam/www/assets/white-papers/us/en/2022/validation-of-quantum-safe-macsec-white-paper.pdf
- IETF draft - Application Scenarios for the Quantum Internet
  https://datatracker.ietf.org/doc/draft-irtf-qirg-quantum-internet-use-cases/
- Blog - Solving the Quantum Computing Security Problem
  https://blogs.juniper.net/en-us/security/solving-the-quantum-computing-security-problem
- Blog - Quantums in Computing and Encryption: Are They the Same?
  https://blogs.juniper.net/en-us/security/quantums-in-computing-and-encryption-are-they-the-same
- Presentation - Demystifying Quantum Key Distribution
  https://www.youtube.com/watch?v=Mv6ne57-iJU
- Podcast – The Quantum Internet
  https://open.spotify.com/episode/0wBd5LsTZ5qTed38sbOcvu
- Presentation - Exploring Quantum Technology for Networking
  https://www.youtube.com/watch?v=1aImelcyS9Q

What You Need to Know Before Reading This Book

The authors expect the reader to have some basic knowledge about the working of current security protocols including IPsec, basic knowledge about quantum computing and quantum effects, and the Junos OS networking operating system.

What You Need to Know About RFC8784

IPsec VPNs rely on the Internet Key Exchange (IKE) protocol for establishing and maintaining security parameters that protect the data traffic. These parameters include encryption and authentication algorithms as well as associated keys along with the lifetime of key material after which new keys must be negotiated.

However, security protocols that rely on asymmetric cryptographic algorithms (public-key cryptography) for establishing keys are susceptible to attacks once quantum computers become powerful enough to solve the math behind such algorithms by executing Shor's algorithm. IKE, specifically version 2, is an example of one such vulnerable protocol as it completely relies on Diffie Hellman (DH) or Elliptic Curve Diffie Hellman (ECDH) operation for generating the shared key material. Note that the symmetric cryptographic algorithms, for example Advanced Encryption Standard (AES), are considered safe provided that the key length is sufficiently large.

To solve this challenge, RFC8784 introduces a method to add an additional secret (key) that is present at, or delivered to, both the initiator and the responder side via some strict out-of-band method (for example Quantum Key Distribution) or post-quantum safe key distribution method. This secret is, in addition to the authentication method that is already provided, within IKEv2.

The additional key material provides quantum resistance to any child Security Associations (SA), therefore, initial negotiated IPsec SA and any subsequent rekeyed IKE and IPsec SAs. This secret is also mixed with peer authentication key so that both sides can cleanly detect mismatches. This secret is called Post-quantum Preshared Keys (PPKs).

RFC8784 does not specify how these PPKs are exchanged. In this book we will look at a few of the options currently available to make IPsec Quantum-safe.

- https://www.rfc-editor.org/rfc/rfc8784.html

Preface

Quantum computing is a rapidly advancing field that promises to revolutionize the way we approach complex computational problems, including those related to cryptography and network security. The potential benefits of quantum computing are vast, but so are the potential risks. The unique properties of quantum computers, such as the ability to perform certain calculations exponentially faster than classical computers, also threaten the security of traditional encryption methods.

One of the most significant threats posed by quantum computing is its ability to break cryptographic protocols that rely on the hardness of certain mathematical problems. Shor's algorithm, a quantum algorithm discovered in 1994, can factor large numbers in polynomial time, which renders many widely used cryptographic schemes vulnerable to

attack. This algorithm represents a significant threat to network security, as it exposes the thread that quantum computers pose to the integrity of the data transmitted over networks.

The emergence of quantum computers highlights the importance of developing new cryptographic protocols and methods that can withstand quantum attacks. One such method for key distribution is Quantum Key Distribution (QKD), which allows two parties to establish a shared secret key that is impossible to intercept or copy without disturbing the quantum state of the system. This protocol offers an unprecedented level of security and is particularly relevant for network security protocols such as IPsec.

As network and security engineers, it is our responsibility to stay ahead of the curve and to anticipate the threats that we may face. This book aims to provide those involved in networking and security a comprehensive understanding of the threats posed by quantum computing to network security, in particular IPsec, and the role that QKD plays in combating those threats.

The Internet Engineering Task Force (IETF) has recently standardized an extension of IKEv2 to allow it to be resistant to a quantum computer; RFC8784 defines strong preshared keys with IPsec, providing a comprehensive framework for the deployment of, amongst other quantum-safe protocols, QKD in network security. This standard represents a significant step forward in the development of quantum-resistant cryptography and its adoption is likely to increase in the coming years.

Our goal with authoring this book is to equip the reader with the knowledge and tools needed to take, perhaps the first, steps in protecting their networks against quantum attacks. We will provide a basic understanding of the foundations of quantum security and the practical applications of QKD in network security and in particular IPsec.

We will also explore the limitations and potential drawbacks of QKD and examine alternative approaches to quantum-resistant cryptography.

We aim to provide a comprehensive understanding of the threats posed by quantum computing to IPsec and the role that quantum-secure key exchange algorithms play in mitigating those threats. We hope that this book will serve as a valuable resource for network engineers and security professionals who wish to stay ahead of the curve and protect their networks against quantum attacks.

Melchior Aelmans, et al., March 2023

Chapter 1

The Big Threat

The advent of quantum computing brings many new possibilities for those forecasting the weather, predicting climate change impact, drug development, financial modeling, and many more applications a 'classical computer' would never be able to do.

Something a Quantum computer can excel in is factoring prime numbers. In fact, it could do this at magnitudes faster than a classical computer if able to run Shor's algorithm. When quantum computers become so powerful that they can effectively run this algorithm it is assumed they will be able to find the prime factors of large numbers very efficiently and by doing so effectively cracking current Public Key Infrastructure (PKI) methods.

Store Now, Decrypt Later

The "Store now, decrypt later" threat refers to the risk of intercepted encrypted data being stored in present time and decrypted in the future by a third party who gains access to the encryption key used to secure the data. Even if he currently is not able to gain access to the encryption key.

If an attacker can obtain the encryption key used to encrypt the data, they can use this key to decrypt the data, potentially gaining access to sensitive information. In some cases, attackers may also use advanced techniques such as brute-force attacks or side-channel attacks to try and obtain the encryption key.

With the development of quantum computing technology, the "Store now, decrypt later" threat is potentially more significant. Quantum computers are designed to operate on a fundamentally different level than classical computers, which could make them capable of breaking many of the encryption algorithms currently used to secure data.

To mitigate the risk of the "Store now, decrypt later" threat, it is essential to implement strong(er) encryption methods and to ensure that encryption keys are securely managed and protected. Additionally, it may be necessary to use post-quantum cryptography methods that are designed to be secure even against the potential future threat of quantum computers.

Shor's Algorithm

Shor's algorithm is a quantum algorithm that can factor large integers and find discrete logarithms efficiently. It was discovered by Peter Shor in 1994 and is considered one of the most significant quantum algorithms, as it has implications for the security of many encryption schemes that rely on the difficulty of factoring large numbers. But first let's get into some of the basic principles of quantum and then we will discuss how Shor's algorithm works.

Basic Principles of Quantum Mechanics

To understand Shor's algorithm, it is important to understand the basic principles of quantum mechanics.

Quantum mechanics is a branch of physics that studies the behavior of particles on a very small scale, such as atoms and subatomic particles. Unlike classical mechanics, which deals with the behavior of objects on a large scale, quantum mechanics requires a new set of rules and principles to describe the behavior of particles on a small scale.

In classical mechanics, we can determine the position and velocity of an object with a high degree of accuracy. However, in quantum mechanics, we cannot simultaneously determine both the position and velocity of a particle. This is due to the uncertainty principle, which states that the more precisely we know one property of a particle, the less precisely we can know the other property.

Another fundamental principle of quantum mechanics is superposition. Superposition states that a particle can exist in multiple states at the same time. For example, an electron can exist in multiple energy states simultaneously, and its wave function, which describes the probability of finding the electron in a particular location, will reflect this superposition.

Superposition is a key component of quantum computing, as it allows for the creation of quantum bits or qubits. Unlike classical bits, which can only be in one state (either 0 or 1), qubits can exist in a superposition of states, which allows for exponential speedup in certain computational task.

Entanglement

One of the most counter-intuitive aspects of quantum mechanics is entanglement. Entanglement occurs when two or more particles become correlated in such a way that the state of one particle is dependent on the state of the other particle. This means that measuring the state of one particle instantaneously affects the state of the other particle, even if they are separated by great distances.

The deep ways that quantum information differs from classical information involve the properties, implications, and uses of quantum entanglement. Entangled states are interesting because they exhibit correlations that have no classical analog. Though Albert Einstein colorfully dismissed quantum entanglement—the ability of separated objects to share a condition or state—as "spooky action at a distance."

Over the past few decades, however, physicists have demonstrated the reality of spooky action over ever greater distances—even from Earth to a satellite in space. This holistic property of compound quantum systems, which involves nonclassical correlations between subsystems, is a potential for many quantum applications: quantum cryptography, quantum teleportation and dense coding.

However, it appeared that this new resource is very complex and difficult to detect. Being usually fragile to environment, it is robust against conceptual and mathematical tools, the task of which is to decipher its rich structure.

Quantum entanglement is the physical phenomenon that occurs when a group of particles are generated, interact, or share spatial proximity in a way such that the quantum state of each particle of the group cannot be described independently of the state of the others, including when the particles are separated by a large distance. The topic of quantum entanglement is at the heart of the disparity between classical and quantum physics: entanglement is a primary feature of quantum mechanics not present in classical mechanics.

Measurements of physical properties such as position, momentum, spin, and polarization performed on entangled particles can, in some cases, be found to be perfectly correlated. For example, if a pair of entangled particles is generated such that their total spin is known to be zero, and one particle is found to have clockwise spin on a first axis, then the spin of the other particle, measured on the same axis, is found to be anticlockwise. However, this behavior gives rise to seemingly paradoxical effects: any measurement of a particle's properties results in an irreversible wave function collapse of that particle and changes the original quantum state. With entangled particles, such measurements affect the entangled system as a whole.

Such phenomena were the subject of a 1935 paper by Albert Einstein, Boris Podolsky, and Nathan Rosen and several papers by Erwin Schrödinger shortly thereafter describing what came to be known as the EPR paradox. Einstein and others considered such behavior impossible, as it violated the local realism view of causality (Einstein referring to

it as "spooky action at a distance") and argued that the accepted formulation of quantum mechanics must therefore be incomplete.

Later, however, the counter-intuitive predictions of quantum mechanics were verified in tests where polarization or spin of entangled particles was measured at separate locations, statistically violating Bell's inequality. In earlier tests, it could not be ruled out that the result at one point could have been subtly transmitted to the remote point, affecting the outcome at the second location. However, so-called "loophole-free" Bell tests have been performed where the locations were sufficiently separated that communications at the speed of light would have taken longer—in one case, 10,000 times longer—than the interval between the measurements.

Quantum entanglement has been demonstrated experimentally with photons, neutrinos, electrons, molecules as large as buckyballs, and even small diamonds. The utilization of entanglement in communication, computation and quantum radar is a very active area of research and development.

Shor's Algorithm Explained

Shor's algorithm works by leveraging the power of quantum superposition and entanglement to perform calculations on large numbers. The algorithm has two main steps: the first step involves finding the period of a function, and the second step involves using the period to factor the number into its prime factors.

Step 1: Finding the Period of a Function

The first step of Shor's algorithm involves finding the period of a function f(x) = a^x mod N, where a is a random number between 1 and N-1, and N is the number to be factored. The period of the function is defined as the smallest positive integer r such that f(x) = f(x+r) for all x. Finding the period of the function is difficult for classical computers, but Shor's algorithm makes use of quantum mechanics to make this calculation more efficient.

The algorithm begins by preparing a quantum state that consists of two registers: the first register contains n qubits that are initially set to 0, and the second register contains m qubits that are initially set to a superposition of all possible values from 0 to 2^m-1. The total number of qubits is therefore n+m.

The algorithm then applies the following operations to the quantum state:

- Apply a Hadamard gate to each qubit in the second register, which puts them into a superposition of all possible values.
- Apply a quantum gate that performs the function f(x) = a^x mod N on the second register.

- Apply a quantum Fourier transform to the second register, which transforms the superposition of all possible values into a superposition of the period of the function.

After these operations are performed, the quantum state is measured, and the result is a superposition of all possible values of the period r. However, since the measurement collapses the quantum state, the result is only one value of r, which is the period of the function.

Step 2: Factoring the Number

The second step of Shor's algorithm involves using the period r to factor the number N. The algorithm uses a classical algorithm called the continued fraction algorithm to find a fraction p/q that approximates r with high precision. Once p and q are known, the factors of N can be computed as gcd(a^(p/2) - 1, N) and gcd(a^(p/2) + 1, N), where gcd stands for greatest common divisor.

The efficiency of Shor's algorithm comes from the fact that the quantum Fourier transform can be performed efficiently on a quantum computer, whereas the classical algorithm for finding the period of a function is much slower.

Limitations of Shor's Algorithm

Although Shor's algorithm is highly efficient for factoring large numbers, it has some limitations. One limitation is that it requires a large number of qubits to be implemented on a quantum computer. The number of logical qubits (as we need error correction, the number of physical qubits will be higher) required to run Shor's algorithm potentially grows with the size of the number to be factored.

For example, to factor a 2048-bit number, which is the standard size of Rivest–Shamir–Adleman (RSA) keys used for encryption, would require approximately 4000 qubits, which is beyond the current state of the art in quantum computing. However, recently researchers claimed to have found a way to reduce the number of qubits down to 372.

It is not efficient for factoring prime numbers or finding discrete logarithms in finite fields, which are also important problems in cryptography.

Despite these limitations, the potential impact of Shor's algorithm on cryptography cannot be ignored. Many cryptographic schemes rely on the difficulty of factoring large numbers, and the discovery of an efficient algorithm for factoring large numbers would render these schemes insecure. The realization of Shor's algorithm on a large-scale quantum computer would therefore have significant implications for the security of mod-ern communication systems.

Chapter 2

Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec) is a suite of protocols that provides secure communication over an IP network. It was developed to overcome the limitations of the original IP protocol, which did not provide any security mechanisms. IPsec provides security at the network layer, which makes it transparent to the applications running on top of the network. IPsec provides authentication, confidentiality, and integrity for IP packets, which makes it an ideal choice for securing overlay virtual private networks (VPNs).

Fundamentals of IPsec

IPsec has two main protocols: the Authentication Header (AH) and the Encapsulating Security Payload (ESP). AH provides authentication and integrity, while ESP provides authentication, integrity, and confidentiality. These two protocols can be used separately or together to provide different levels of security.

Authentication Header (AH)

The Authentication Header (AH) provides authentication and integrity for IP packets. It does this by using a hash algorithm to create a message digest of the packet's header and payload. The message digest is then inserted into the packet's header, which allows the receiver to verify the authenticity of the packet.

The AH protocol is used to protect the packet's header and payload, but not the data itself. This means that the data can be read by anyone who intercepts the packet, but the receiver can be sure that the packet has not been modified in transit. AH is mainly used in situations where confidentiality is not required, but authentication and integrity are important.

Encapsulating Security Payload (ESP)

The Encapsulating Security Payload (ESP) protocol provides authentication, integrity, and confidentiality for IP packets. It does this by encrypting the packet's payload and adding a message digest to the packet's header. The message digest is used to verify the authenticity of the packet, and the encryption provides confidentiality.

ESP can be used with or without authentication, but it is always used with encryption. When used with authentication, the message digest is created using the packet's header, payload, and a secret key. This ensures that the packet has not been modified in transit and that it was sent by the sender who has the secret key.

IPsec Security Associations

IPsec uses Security Associations (SAs) to provide security for IP packets. A Security Association is a set of security parameters that are agreed upon by two devices before they start communicating. The parameters include the security protocol to be used, the encryption algorithm, the key size, and the lifetime of the SA.

An SA is created when two devices exchange Security Parameter Indexes (SPIs). The SPIs are used to identify the SA and the devices involved in the communication. The SPIs are included in the packet's header, which allows the receiver to match the incoming packet with the correct SA.

Internet Key Exchange

Internet Key Exchange (IKE) is a protocol used in IPsec to establish and manage Security Associations (SAs) between two devices. It is responsible for negotiating the security parameters, exchanging keys, and setting up the encryption and authentication algorithms that will be used to protect the IP traffic.

IKE is a complex protocol that operates in two phases. During the first phase, IKE establishes a secure channel between the devices and negotiates the security parameters. During the second phase, IKE uses the agreed-upon parameters to establish the SAs and configure the IPsec security services.

Phase 1 of IKE involves the following steps:

1. The initiator device sends a proposal to the responder device. The proposal includes the encryption algorithm, the hash algorithm, and the Diffie-Hellman group that will be used to generate the shared secret key.

2. The responder device sends a proposal back to the initiator. The proposal includes a counterproposal with the security parameters that the responder prefers.

3. The initiator and responder devices agree on the security parameters and generate the shared secret key using the Diffie-Hellman key exchange algorithm.

4. The initiator and responder devices authenticate each other using digital certificates or a pre-shared key.

5. The secure channel between the devices is established, and the devices move to Phase 2.

Phase 2 of IKE involves the following steps:

1. The initiator device sends a proposal to the responder device, including the security parameters that will be used for the IPsec SAs.

2. The responder device responds with a proposal, including the security parameters that it prefers.

3. The initiator and responder devices agree on the security parameters and establish the IPsec SAs.

4. The devices begin using the IPsec SAs to protect the IP traffic.

IKE is a critical component of IPsec because it provides a secure method for establishing the SAs that are required for IPsec to provide its security services.

In summary, IKE is a protocol that works alongside IPsec to establish and manage the SAs required for secure communication over IP networks. IKE negotiates the security parameters, exchanges keys, and establishes the encryption and authentication algorithms that will be used to protect the IP traffic. Without IKE, IPsec would require a manual configuration of the security parameters, making it less practical and potentially more vulnerable to errors.

Diffie-Hellman key exchange algorithm

The Diffie-Hellman (DH) key exchange algorithm is used by IKE to generate a shared secret key that is used to encrypt and decrypt IPsec traffic. DH is a cryptographic protocol used to establish a shared secret between two parties over an insecure communication channel. It was first introduced by Whitfield Diffie and Martin Hellman in 1976 and is considered one of the first public-key protocols.

The algorithm works by allowing two parties, Alice and Bob, to generate a shared secret without ever transmitting the secret over the communication channel. Instead, both parties use a shared public value and their own private values to compute the secret key. Here is a step-by-step breakdown of the protocol:

1. Alice and Bob agree on a prime number, p, and a primitive root of p, g. These values can be publicly known and can be reused for multiple exchanges.
2. Alice chooses a secret integer, a, and calculates A = g^a mod p. She sends A to Bob.
3. Bob chooses a secret integer, b, and calculates B = g^b mod p. He sends B to Alice.
4. Alice computes the shared secret key as S = B^a mod p.
5. Bob computes the shared secret key as S = A^b mod p.

Both Alice and Bob now have the same secret key, S, which can be used to encrypt and decrypt messages between them using a symmetric encryption algorithm, such as AES.

The security of the Diffie-Hellman key exchange algorithm relies on the difficulty of computing discrete logarithms in modular arithmetic. Specifically, an attacker who intercepts A and B cannot easily calculate (with a 'classical' computer) a or b without solving the discrete logarithm problem, which is believed to be computationally infeasible for sufficiently large prime numbers.

However, the Diffie-Hellman key exchange algorithm is vulnerable to a man-in-the-middle attack, where an attacker intercepts and alters the public values used in the protocol. To prevent this, Alice and Bob can authenticate each other's public values using digital signatures or a trusted third party.

Why IPsec Is Vulnerable to a Quantum Computer Attack?

DH algorithm is based on the difficulty of calculating discrete logarithms. In a classical computing environment, calculating discrete logarithms for sufficiently large prime numbers is computationally infeasible, which makes the Diffie-Hellman algorithm a secure method for key exchange.

However, quantum computers use a different approach to computation, based on quantum bits or qubits, which allows them to perform certain calculations much faster than classical computers. One of the computations that quantum computers excel at is the calculation of discrete logarithms, which could potentially allow them to break the security of the Diffie-Hellman key exchange algorithm.

The vulnerability of the Diffie-Hellman key exchange algorithm to quantum computers is based on Shor's algorithm, which is a quantum algorithm for factoring large numbers and calculating discrete logarithms. Shor's algorithm has been demonstrated on small-scale quantum computers, and it has been shown that a quantum computer with a sufficient number of qubits could break the security of the Diffie-Hellman key exchange algorithm.

One possible solution to this vulnerability is to use post-quantum cryptography, which is a form of cryptography that is designed to be secure against attacks by quantum computers. Post-quantum cryptography uses different mathematical algorithms that are resistant to quantum computing attacks, and research is currently underway to develop post-quantum cryptographic systems that can be used in IPsec and other security protocols.

In conclusion, the Diffie-Hellman key exchange algorithm used by IKE in IPsec is vulnerable to attacks by quantum computers due to the ability of quantum computers to perform calculations much faster than classical computers. This vulnerability could be addressed by using post-quantum cryptography, which is designed to be resistant to quantum computing attacks.

Chapter 3

Quantum-safe Security

Advanced Encryption Standard (AES)

AES is a symmetric key encryption algorithm used to protect sensitive information. It was established in 2001, replacing the older DES (Data Encryption Standard) and 3DES (Triple DES) encryption algorithms. AES is used in various security protocols, including SSL/TLS, SSH, and IPsec. The 256-bit key length version, AES256, is one of the most secure and commonly used encryption standards available today.

AES256 workings

AES256 is a symmetric encryption algorithm, which means that it uses the same key to both encrypt and decrypt data. The algorithm operates on 128-bit blocks of plaintext, which are divided into a 4x4 matrix of bytes. The key length for AES256 is 256 bits, which provides an enormous number of possible key combinations.

The encryption process begins by adding a round key to the plaintext. The round key is created by performing a key expansion routine on the original key. The plaintext is then subjected to a series of rounds, each of which consists of four operations: substitution, transposition, mixing, and adding the round key. These operations are applied repeatedly, creating multiple rounds that obscure the relationship between the original plaintext and the encrypted ciphertext.

In the substitution operation, each byte of the plaintext matrix is replaced by another byte from a fixed lookup table. The lookup table is created by applying a series of mathematical operations to the original key. In the transposition operation, the rows and columns of the plaintext matrix are shuffled to further obscure the plaintext. In the mixing operation, each column of the matrix is multiplied by a fixed matrix to create a new

column, providing additional diffusion of the plaintext. The final step in each round is to add the round key, which is created by performing another key expansion operation on the original key.

This process of repeated rounds continues until the final ciphertext is produced. To decrypt the ciphertext, the same process is applied in reverse, using the same key. The decryption process begins by adding the final round key to the ciphertext, then applying the inverse operations for each round until the original plaintext is obtained.

Security protocols that use AES256

AES256 is used in many security protocols, including SSL/TLS, SSH, and IPsec. In SSL/TLS, AES256 is often used in conjunction with RSA (Rivest-Shamir-Adleman) or Elliptic Curve cryptography for key exchange and authentication. In SSH, AES256 is used to encrypt the data that is transmitted between the client and server. In IPsec, AES256 is used to encrypt the data that is transmitted between two VPN (Virtual Private Network) endpoints.

Why AES256 is not vulnerable to Shor's algorithm

AES256 is a symmetric key encryption algorithm, which means that it uses the same key for encryption and decryption. This makes it immune to Shor's algorithm and other attacks that rely on factoring large numbers, as symmetric key algorithms are not based on mathematical problems that can be solved efficiently by a quantum computer.

Quantum computers work by using quantum bits, or qubits, which can exist in multiple states simultaneously. This allows quantum computers to perform certain computational tasks much faster than classical computers. However, quantum computers are still in their infancy and are currently limited to small numbers of qubits. The largest quantum computer currently available has only 256 qubits, which is not enough to run Shor's algorithm and therefor currently break AES256 encryption.

Furthermore, even if quantum computers become powerful enough to break AES256 encryption, there are other symmetric key encryption algorithms, such as ChaCha20 and Salsa20, that are designed to be resistant to quantum computing attacks. These algorithms use a different approach to encryption which are not easily solved by quantum computers.

In addition to its resistance to quantum computing attacks, AES256 is also designed to be resistant to other types of attacks, including brute force attacks, side-channel attacks, and chosen-plaintext attacks. Brute force attacks attempt to break the encryption by trying every possible key combination, but with 2^256 possible key combinations, AES256 is effectively immune to brute force attacks.

Side-channel attacks attempt to extract the key by analyzing information leaked through power consumption or electromagnetic radiation, but AES256 is designed to be resistant

to these types of attacks as well. Chosen-plaintext attacks attempt to extract the key by analyzing the encryption of specially crafted plaintexts, but AES256 is designed to be secure even in the face of such attacks.

Quantum Key Distribution

Quantum Key Distribution (QKD) is a technology that enables the secure exchange of cryptographic keys between two parties over a public channel. The security of QKD is based on the fundamental principles of quantum mechanics, which state that it is impossible to measure or copy the quantum state of a particle without disturbing it. Therefore, any attempt to eavesdrop on the communication will be detected by the legitimate parties. QKD offers a way to achieve unconditional security that cannot be broken even by future advances in computing technology.

In this chapter, we will discuss the working principle of QKD, its limitations, the BB84 protocol and MDI-QKD to overcome some of the limitations. We will also talk about the technology used for photon detection, the difference between fiber and free-space QKD, and the future prospects of QKD.

Working Principle of Quantum Key Distribution

The main idea behind Quantum Key Distribution (QKD) is to encode information in the quantum states of photons, which are sent over a public channel to the receiver. The sender, usually referred to as Alice, chooses a random sequence of quantum states from a set of orthogonal states, such as polarization or phase, to encode her message. The receiver, usually referred to as Bob, measures the photons in the same basis to retrieve the message.

The key idea behind QKD is that the act of measuring a series of photons in a quantum state disturbs at least the state of one of the photons, making it impossible for an eavesdropper, usually referred to as Eve, to intercept and copy the photons without being detected. If Eve measures a single photon in the correct basis, the state will not be disturbed. The state is only disturbed if Eve measures in the wrong basis. That is why Alice has to randomly choose the encoding basis. Any attempt by Eve to measure the photons will introduce errors in the measurement statistics, which can be detected by Alice and Bob.

The key distribution process begins with Alice generating a random sequence of quantum states, which she sends over the public channel to Bob. Bob measures each received photon in a randomly chosen basis. Alice and Bob communicate their chosen basises only after the measurements have been completed. This process is repeated several times, and Alice and Bob compare a subset of their measurements to detect any errors that may have been introduced by noise or eavesdropping.

If the error rate is within a tolerable limit, Alice and Bob proceed to the key distillation phase, in which they use a protocol such as error correction and privacy amplification to distill a secret key from the shared quantum states. The secret key can then be used by IPsec to encrypt and decrypt messages sent over a classical channel.

BB84 Protocol

One of the most popular QKD protocols is the BB84 protocol, which was proposed by Charles Bennett and Gilles Brassard in 1984. The BB84 protocol uses two complementary sets of orthogonal quantum states, usually polarization or phase, to encode the secret key. The two sets of states are chosen randomly for each transmission, and the receiver is unaware of which set has been used.

The BB84 protocol has four steps:

1. state preparation
2. random basis selection
3. measurement
4. sifting the results
5. error correction and privacy amplification.

In the first step, Alice prepares a random sequence of qubits in one of the two orthogonal bases, either the rectilinear basis (X-basis) or the diagonal basis (Y-basis), and sends them to Bob over the quantum channel.

In the second step, Bob randomly selects a basis in which to measure each qubit, without knowing which basis Alice used to encode it.

In the third step, Bob measures each qubit in the selected basis, obtaining either the correct state or a random one with a certain probability.

In the fourth step, Alice and Bob exchange chosen measurement basis and discard and measurements for which they chose a different basis.

In the fifth step, Alice and Bob compare a subset of their measurement outcomes to detect any errors that may have been introduced by noise or eavesdropping.

If the error rate is within a tolerable limit, they proceed to the key distillation phase, in which they use error correction and privacy amplification to distill a secret key from the shared quantum states.

The result is that a key is now present at both ends which can be consumed by those in need for keys while using the ETSI QKD 014 REST API.

Photon Detection Technology

In QKD, photon detection technology is critical to achieving high detection efficiency, low noise, and high timing resolution. There are several types of photon detectors used in QKD, including avalanche photodiodes (APDs), superconducting nanowire single-photon detectors (SNSPDs), and silicon photomultipliers (SiPMs).

APDs are the most commonly used photon detectors in QKD. They are semiconductor devices that operate in reverse-biased mode, and they can detect single photons with high efficiency and low noise. However, they have a relatively slow response time, which limits the maximum clock rate of the QKD system.

SNSPDs are another type of photon detector that can detect single photons with high efficiency and low noise. They are based on superconducting nanowires that are cooled to cryogenic temperatures, typically below 4 Kelvin. SNSPDs have a faster response time than APDs, and they can operate at higher clock rates.

SiPMs are a relatively new type of photon detector that has gained attention in recent years. They are based on arrays of avalanche photodiodes, and they can detect single photons with high efficiency and low noise. SiPMs have a faster response time than APDs and can operate at higher clock rates. They are also more robust to environmental factors such as temperature and humidity.

Fiber vs. Free-space QKD

QKD can be implemented in either a fiber-based or free-space system, depending on the requirements of the application.

Fiber-based QKD uses standard telecommunication fibers to transmit photons between Alice and Bob. This technology is well-developed and has been demonstrated over long distances, up to hundreds of kilometers.

Free-space QKD, on the other hand, uses lasers and telescopes to transmit photons through the air between Alice and Bob. This technology is less mature than fiber based QKD and is limited by atmospheric turbulence, scintillation, and other environmental factors. However, it offers the potential for high-speed, high-bandwidth communication over long distances, without the need for expensive and cumbersome fiber-optic cables.

Quantum Key Distribution limitations

Despite its promise, QKD is not a panacea for secure communication. There are several limitations and challenges that must be addressed to make QKD a practical and widely adopted technology.

One of the biggest challenges is the limited range of QKD systems. While QKD has been demonstrated over long distances in laboratory settings, practical systems are currently limited to a few hundred kilometers, due to losses in the transmission channel and detector inefficiencies.

Another challenge is the fact that QKD requires sophisticated and expensive hardware, including high-quality photon detectors, precise laser sources, and stable temperature control. This makes it challenging to nowadays deploy QKD systems at scale.

MDI-QKD

MDI-QKD (measurement-device-independent quantum key distribution) is a variant of QKD that eliminates the need for trusted measurement devices, making it more resistant to tampering and hacking. In traditional QKD, the measurement devices used by both

parties are trusted, meaning that an attacker could potentially compromise the security of the system by manipulating the measurement devices. MDI-QKD overcomes this vulnerability by using a different approach to generate the secret key.

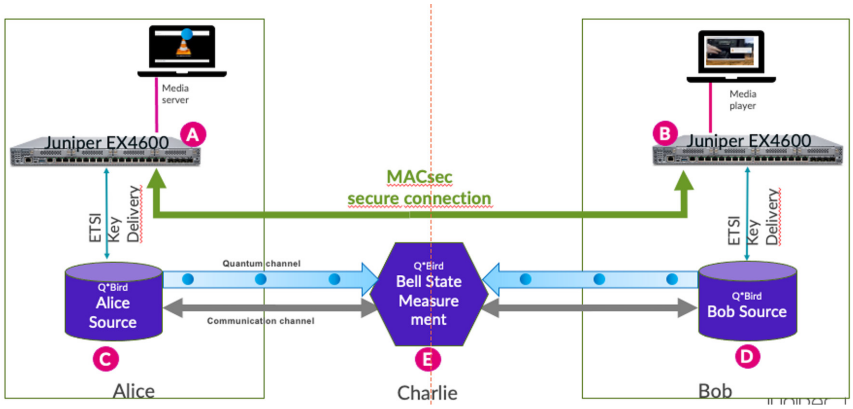

Figure 3.1 *MDI-QKD*

MDI-QKD (measurement-device-independent quantum key distribution) is a variant of QKD that eliminates the need for a trusted node in the path to overcome distance limitation. Also as the key itself is not known a the trusted node in MDI-QKD it is more resistant to tampering and hacking. In traditional QKD, an attacker could potentially compromise the security of the system by manipulating the measurement device at the trusted node. MDI-QKD overcomes this vulnerability by using a different approach to generate the secret key.

The basic idea behind MDI-QKD is to use a third party, referred to as Charlie, to measure the state of the photons Alice and Bob send. Charlie then announces the measurement basises and the measurement values to Alice and Bob, which allows them to compare their measurement results and detect any discrepancies that may indicate the presence of an eavesdropper.

The final step in the MDI-QKD process involves distilling a shared secret key from the measurement results obtained by Alice and Bob. This is done using the same post-processing algorithms used in traditional QKD, exchanging the measurement base, lifting thru the results, error correction and privacy amplification to remove any information that an eavesdropper may have obtained during the transmission.

Post-quantum Cryptography

Post-quantum cryptography (PQC) is designed to be resistant against attacks from quantum computers. Quantum computers have the potential to break many of the current cryptographic systems that are widely used today, including RSA and ECC (Elliptic Curve Cryptography). PQC aims to provide an alternative to these cryptographic systems that is secure against quantum attacks.

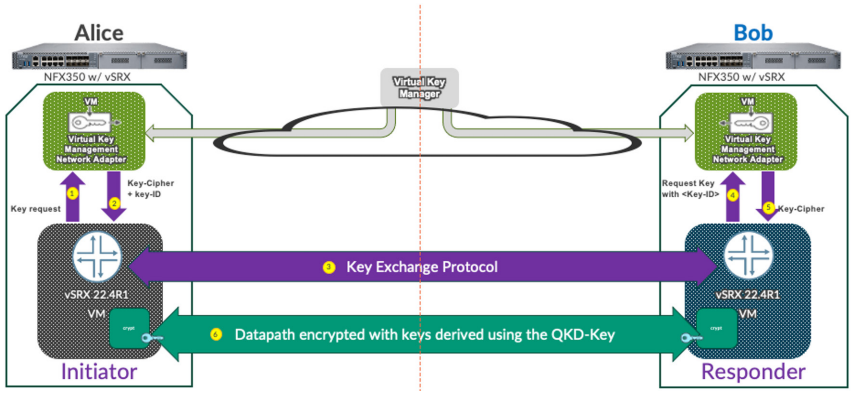

Figure 3.2

Classical Cryptography vs Post-Quantum Cryptography

Classical cryptography is based on mathematical problems that are difficult to solve, such as factoring large numbers and discrete logarithms. These mathematical problems are the basis for cryptographic algorithms such as RSA and ECC, which are widely used today to provide secure communication over the internet.

POC

PQC, on the other hand, is based on mathematical problems that are believed to be difficult to solve even for quantum computers. These mathematical problems are referred to as post-quantum encryption, and they include lattice-based problems, code-based problems, and hash-based problems. By using these post-quantum problems as the basis for cryptographic algorithms, PQC aims to provide a secure alternative to classical cryptography that is resistant to attacks from quantum computers.

Lattice-Based Cryptography

One of the most popular types of PQC is lattice-based cryptography. This approach is based on the mathematical concept of lattices, which are geometric structures that can be used to represent complex mathematical problems. In lattice-based cryptography, the encryption and decryption process involves finding a short vector in a high-dimensional lattice. The short vector can be used to encode the plaintext message, and the decryption process involves finding the original message from the encoded vector.

Lattice-based cryptography has several advantages over classical cryptographic systems. For example, the security of the system is based on the difficulty of finding short vectors in high-dimensional lattices, which is believed to be a difficult problem even for quantum computers. Additionally, lattice-based systems can be used to provide homomorphic encryption, which allows computations to be performed on encrypted data without revealing the plaintext.

Code-Based Cryptography

Another type of PQC is code-based cryptography. This approach is based on error-correcting codes, which are widely used in telecommunications to correct errors in transmitted data.

In code-based cryptography, the encryption and decryption process involves encoding the plaintext message using an error-correcting code. The encoded message is then transmitted over the communication channel, and the decryption process involves decoding the received message using the same error-correcting code.

Code-based cryptography has several advantages over classical cryptographic systems. For example, the security of the system is based on the difficulty of decoding the error-correcting code, which is believed to be a difficult problem even for quantum computers. Additionally, code-based systems can be implemented using relatively simple hardware, which makes them well-suited for resource-constrained environments.

Hash-Based Cryptography

A third type of PQC is hash-based cryptography. This approach is based on cryptographic hash functions, which are one-way functions that can be used to generate a fixed-length digest of an input message.

In hash-based cryptography the encryption and decryption process involves generating a shared secret key using a cryptographic hash function. The plaintext message is then encrypted using the shared secret key, and the decryption process involves decrypting the received message using the same shared secret key.

Hash-based cryptography has several advantages over classical cryptographic systems. For example, the security of the system is based on the difficulty of finding collisions in

the hash function, which is believed to be a difficult problem even for quantum computers. Additionally, hash-based systems have the advantage of being relatively simple to implement and have low computational requirements.

Key Exchange and Digital Signatures

In addition to encryption and decryption, PQC also provides secure key exchange and digital signatures. Key exchange is the process of securely exchanging cryptographic keys between two parties, while digital signatures are used to provide proof of the authenticity of a message. PQC provides several different methods for key exchange, including the Diffie-Hellman key exchange, which is widely used in classical cryptography, and the McEliece key exchange, which is based on code-based cryptography.

Digital signatures in PQC are based on different mathematical problems, including the hash-based Merkle signature scheme, which is based on the Merkle tree data structure, and the lattice-based signature scheme, which is based on the short vector problem in lattices.

Challenges and Limitations

Despite its potential advantages, PQC is still in the research phase, and there are several challenges and limitations that need to be addressed before it can be widely adopted.

One of the main challenges is the lack of standardization. Unlike classical cryptography, which has well-established standards and protocols, PQC is still a relatively new field, and there is a lack of consensus on which algorithms and methods are the most secure and efficient. Another challenge is the performance overhead of PQC algorithms. Many of the PQC algorithms are computationally intensive and require significant computational resources, which can be a limitation in resource-constrained environments.

Finally, there is the challenge of transitioning from classical cryptography to PQC. This transition will require significant changes to the existing infrastructure and protocols, and it may take many years before PQC becomes the dominant cryptographic standard.

It is also worth noting that some PQC candidate protocols have been broken very late in the NIST evaluation process. For example: https://eprint.iacr.org/2022/214.

ETSI QKD 014 specification

The ETSI 014 QKD REST API is a specification developed by the European Telecommunications Standards Institute (ETSI) for the management of QKD networks. This API provides a standardized interface for communication between different QKD network components, such as QKD devices, key servers, and network components such as routers, switches, and firewalls.

The API is in the context of this book mainly used to manage the distribution of the keys and key IDs generated by the QKD devices.

Chapter 4

Pulling IPsec Together with QKD or PQC

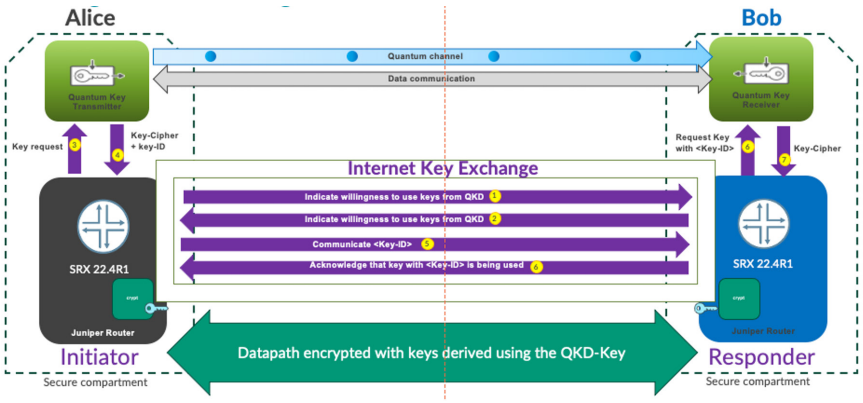

Figure 4.1 *Chapter 4's Implementation*

Leveraging the ETSI-QKD-014 API for Key Exchange

The ETSI standards setting organization defined a REST API to exchange keys between a Key Management Server [KMS] (typically included in the QKD system) and a security application entity [SAE] such as Juniper SRX. The API itself is secured by TLS which we know is not Quantum-secure. For that reason, it is only allowed to be operated in a physically secure compartment as indicated by the dotted boxes. Such compartment can for example be a rack with locked doors or a locked room. In practice this means that the QKD system and the secure application device are in proximity to each other and do not exchange key material over the Internet.

In a typical scenario, each SAE must first identify itself to the local KME to start setting up a quantum secure IPsec tunnel.

1. The two SAEs need to elect an initiator and a responder. In the Juniper implementation this tie-breaking is part of the IKEv2 initialization process and based on the alphabetic order of the SAE names. In the following we assume "SAE-A" in the initiator role and "SAE-B" as responder.

2. The SAE indicates its preference to use QKD-keys to the IPsec peer

3. If the peer also supports the use of QKD-keys, the Key exchange process gets scheduled to fetch keys.

4. SAE_A requests a <key> and <key-ID> from its local KME

5. SAE_A receives a <key> and <key-ID>

6. SAE_A communicates the <key-ID> to SAE_B

7. SAE_B requests the key corresponding to the received <key-ID>

8. The local KME delivers the <key> to SAE_B

9. SAE_B confirms having received the <key> corresponding to the <key-ID>

10. Now SAE_A and SAE_B are ready to use the shared key in their communication.

11. After both sides are in possession of their shared keys, those are mixed with the existing SA key materials to get Quantum safe keys.

IKEv2 Post-Quantum Security support

Quantum-safe IPsec Tunnel support was first introduced for Junos in Release 22.4R1. The feature was initially introduced on select SRX and vSRX models. See Feature Explorer for current feature support information for any Junos OS feature, including Post-Quantum Pre-Shared Key support with RFC878.

The Post-Quantum Pre-Shared Key feature allows you to create quantum-safe IPsec tunnels with minimal hassle. This feature is enabled with minimal user configuration and is enabled in a way that does not have any effect on your existing tunnels.

What 's New in IPsec?

IKE-gateways with a quantum-key-profile

Currently, the existing IKEv2 protocol infrastructure negotiates non-quantum-safe IKE and IPSEC SAs. This new feature [security ike gateway <gw-name> ppk-profile], as per RFC8784, will let you create Quantum-safe IPsec Tunnels. So, you can upgrade without any effect on existing tunnels and with minimum hassle.

Support for adding and managing different types of out-of-band mechanisms for keys via key-profiles

Junos-Key-Manager (JKM) daemon is introduced. This daemon will act as a key store and as a proxy between the crypto application that requires a quantum-safe-key to establish an encrypted and authenticated quantum- safe session with peer(s), and the out-of-band mechanism that lets two peers have a quantum-safe-key.

Different out-of- band mechanisms will have different protocols/methods to communicate and thus the purpose of this daemon is to provide a common uniform API for client applications to communicate with JKM for any out-of-band-mechanism (key-profiles type).

For the time being, two types of out-of-band mechanisms for keys (key-profiles configurations) will be supported: static and quantum-key-manager based.

With Static-key based profiles you need to configure a static key-id and a corresponding key. Thus, if a static-key based profile is used with an IPsec-VPN object, every time when re-auth for existing IKE SA (control channel) needs to be done, the same key and key-id will be used.

With QKD-key based profiles you need access to QKD devices in order to gain access to additional key material. To leverage this profile, you will need to configure all the necessary parameters (see below). Thus, if a QKD-key based profile is used with IPsec, every time when re-auth is done, a different key and key-id will be used, which is more secure.

As of 22.4R1, the ike-key-management daemon is leveraging the infra provided by the junos-key-manager daemon to provide quantum safe SAs. However, other internal or external applications can also use this infra to make their respective services quantum safe as well without reinventing the wheel from scratch.

Supported Juniper Platforms and Software Releases

IPsec with RFC8784 support was first made available on select SRX and vSRX platforms in Junos Release 22.4R1. This might change in the future, for example when new platforms are added or newer Junos OS releases bring support for existing platforms. Always check https://apps.juniper.net/feature-explorer/ for latest feature support.

- The first Junos OS version to support RFC8784 is 22.4R1 which is available via https://support.juniper.net/support/downloads/
- For vSRX make sure to download "vSRX 3.0" versions which can be found here: https://support.juniper.net/support/downloads/?p=vsrx3

The aim of this chapter is to describe how QKD can be leveraged in IPsec based on RFC8784. This chapter will provide a detailed overview of the QKD in IPsec architecture, its components, and its operation.

Quantum safe IPsec Configuration Examples

Let's review all the details and the step-by-step configuration for setting up Quantum safe IPsec VPN tunnels with different out-of-band key mechanisms listed as below:

- Quantum safe IPsec VPN using static key profile for PPKs
- Quantum safe IPsec VPN using quantum-key-manager key profile for PPKs

Requirement

- An optional package "junos-ike" needs to be installed to use the feature-set.
- To verify whether the "junos-ike" package is already installed or not, run "show version | grep "JUNOS ike".
- If the package is not installed, run "request system software add optional://junos-ike" on the supported platforms to add it.

Configuring Quantum safe IPsec VPN with Static Key Profile

Figure 4.2 shows how to configure IPsec VPN with a static key-profile for PPKs to allow data to be securely transferred between two sites.

We configure a route-based IPsec VPN on SRX1 and SRX2 with static key-profile configuration for PPKs. A static key profile consists of a configured key ID and a corresponding configured key. Host1 and Host2 use the VPN to send data securely over the Internet between both hosts. HOST1 and HOST2 can be any device running a simple or robust operating system. Configuration of these devices as per the requirement but make sure that each HOST can communicate with its respective SRX devices beforehand.

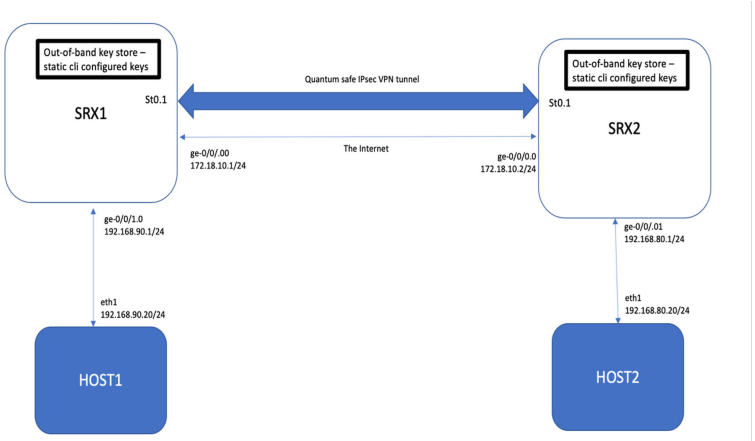

Figure 4.2 Topology for Quantum safe IPsec VPN using static key-profile for PPKs

CLI configurations

To quickly configure SRX1

```
set security key-manager profiles km_profile_1 static key-id ascii-text “test-key-id”
set security key-manager profiles km_profile_1 static key ascii-
text “qjwbdip139u5mcy89m28pcgowerefnkjsdg”
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.2
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.1
set security ike gateway IKE_GW version v2-only
set security ike gateway IKE_GW ppk-profile km_profile_1
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.90.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN establish-tunnels immediately
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.1/24
set interfaces ge-0/0/1 unit 0 family inet address 192.168.90.1/24
set interfaces st0 unit 1 family inet
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
set security zones security-zone vpn interfaces st0.1
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

Step-by-Step Procedure to Configure SRX1

1. Configure a key profile of type static with a key-id and a corresponding key:

Configure an ascii encoded key-id string.

```
set security key-manager profiles km_profile_1 static key-id ascii-text “test-key-id”
```

Configure an ascii encoded key string. The size of the ascii string must be a minimum of 32 characters.

```
set security key-manager profiles km_profile_1 static key ascii-
text "qjwbdip139u5mcy89m28pcgowerefnkjsdg"
```

2. Configure the interfaces:

Configure the external interface

```
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.1/24
```

Configure the secure tunnel interface

```
set interfaces st0 unit 1 family inet
```

Configure the HOST1 facing interface

```
set interfaces ge-0/0/1 unit 0 family inet address 192.168.90.1/24
```

3. Configure the security zones

Configure the untrust security zone with limiting the host-inbound system services to just ike and assign the external interface to the untrust security zone.

```
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
```

Configure the vpn security zone with assigning the secure tunnel interface and other necessary paramters

```
set security zones security-zone vpn interfaces st0.1
```

Configure the trust security zone by limiting the host-inbound system services to just ping and assign the HOST1 facing interface to the trust security zone.

```
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
```

4. Configure the security policies

Configure the security policy for the transit traffic flowing from trust zone to vpn zone

```
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
```

Configure the security policy for the transit traffic flowing from vpn zone to trust zone

```
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

5. Configure the IPsec VPN

Configure an ike proposal with the necessary attributes.

```
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
```

Configure an ike policy with the configured ike proposal and other necessary attributes

```
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text "ipsec-test"
```

Configure an Ike gateway with the defined ike policy and other necessary attributes.

```
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.2
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.1
set security ike gateway IKE_GW version v2-only
```

Configure the previously defined static key-profile to the ike gateway to be used as ppk-profile for PPKs. ppk-profile can only be configured if the ike gateway is configured with [version v2-only].

```
set security ike gateway IKE_GW ppk-profile km_profile_1
```

Configure an ipsec proposal with the necessary attributes.

```
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec proposal IPSEC_PROP lifetime-seconds 2400
```

Configure an ipsec policy with the configured proposal

```
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
```

Configure an IPsec vpn with the previously configured secure tunnel interface, ike gateway, ipsec policy and any other necessary parameters. A Traffic selector is configured to ensure only HOST1 subnet to HOST2 subnet communication is secured using the negotiated IPsec tunnel.

```
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.90.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN establish-tunnels immediately
```

To quickly configure SRX2

```
set security key-manager profiles km_profile_1 static key-id ascii-text "test-key-id"
set security key-manager profiles km_profile_1 static key ascii-
text "qjwbdip139u5mcy89m28pcgowerefnkjsdg"
set security ike proposal IKE_PROP authentication-method pre-shared-keys
```

```
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.1
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.2
set security ike gateway IKE_GW version v2-only
set security ike gateway IKE_GW ppk-profile km_profile_1
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.90.0/24
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.2/24
set interfaces ge-0/0/1 unit 0 family inet address 192.168.80.1/24
set interfaces st0 unit 1 family inet
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
set security zones security-zone vpn interfaces st0.1
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

Step-by-Step Procedure to configure SRX2

1. Configure a key profile of type static with a key-id and a corresponding key

Configure an ascii encoded key-id string.

```
set security key-manager profiles km_profile_1 static key-id ascii-text “test-key-id”
```

Configure an ascii encoded key string. The size of the ascii string must be a minimum of 32 characters.

```
set security key-manager profiles km_profile_1 static key ascii-
text “qjwbdip139u5mcy89m28pcgowerefnkjsdg”
```

2. Configure the interfaces

Configure the external interface

```
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.2/24
```

Configure the secure tunnel interface

```
set interfaces st0 unit 1 family inet
```

Configure the HOST2 facing interface

```
set interfaces ge-0/0/1 unit 0 family inet address 192.168.80.1/24
```

3. Configure the security zones

Configure the untrust security zone with limiting the host-inbound system services to just ike and assign the external interface to the untrust security zone.

```
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
```

Configure the vpn security zone with assigning the secure tunnel interface and other necessary paramters

```
set security zones security-zone vpn interfaces st0.1
```

Configure the trust security zone by limiting the host-inbound system services to just ping and assign the HOST1 facing interface to the trust security zone.

```
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
```

4. Configure the security policies

Configure the security policy for the transit traffic flowing from trust zone to vpn zone

```
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
```

Configure the security policy for the transit traffic flowing from vpn zone to trust zone

```
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

5. Configure the IPsec VPN

Configure an ike proposal with the necessary attributes.

```
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
```

Configure an ike policy with the configured ike proposal and other necessary attributes

```
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
```

Configure an Ike gateway with the defined ike policy and other necessary attributes.

```
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.1
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.2
set security ike gateway IKE_GW version v2-only
```

Configure the previously defined static key-profile to the ike gateway to be used as ppk-profile for PPKs. ppk-profile can only be configured if the ike gateway is configured with [version v2-only].

```
set security ike gateway IKE_GW ppk-profile km_profile_1
```

Configure an ipsec proposal with the necessary attributes.

```
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec proposal IPSEC_PROP lifetime-seconds 2400
```

Configure an ipsec policy with the configured proposal.

```
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
```

Configure an IPsec vpn with the previously configured secure tunnel interface, ike gateway, ipsec policy and any other necessary parameters. A Traffic selector is configured to ensure only HOST2 subnet to HOST1 subnet communication is secured using the negotiated IPsec tunnel.

```
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.90.0/24
```

To quickly configure HOST1 and HOST2

Exact steps to configure the HOSTs is out of scope as it will vary from device to device, but on a generic note, HOST1 must be configured with any IP in the subnet 192.168.90.0/24, other than 192.169.90.1, on its interface connected to SRX1 and HOST2 must be configured with any IP in the subnet 192.168.80.0/24, other than 192.169.00.1, on its interface connected to SRX2.

Verification and Troubleshooting

1. Verify the traffic flow

Ping from HOST1 to HOST2 or vice versa

```
user@HOST1# ping 192.168.80.20 source 192.168.90.20 count 4
PING 192.168.80.20 (192.168.80.20): 56 data bytes
```

```
64 bytes from 192.168.80.1: icmp_seq=0 ttl=64 time=2.151 ms
64 bytes from 192.168.80.1: icmp_seq=1 ttl=64 time=1.710 ms
64 bytes from 192.168.80.1: icmp_seq=2 ttl=64 time=1.349 ms
64 bytes from 192.168.80.1: icmp_seq=3 ttl=64 time=1.597 ms
--- 192.168.80.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.349/1.702/2.151/0.290 ms
```

Data traffic is successfully flowing between the HOSTs

2. Verify ike security associations status

From operational mode, enter the "show security ike security-associations detail" command.

```
user@SRX1> show security ike security-associations detail
IKE peer 172.18.10.2, Index 1, Gateway Name: IKE_GW
  Role: Initiator, State: UP
  Initiator cookie: dee592254e808a2b, Responder cookie: 51f6b1d4a8618332
  Exchange type: IKEv2, Authentication method: Pre-shared-keys
  Local gateway interface: ge-0/0/2.0
  Routing instance: default
  Local: 172.18.10.1:500, Remote: 172.18.10.2:500
  Lifetime: Expires in 1286 seconds
  Reauth Lifetime: Disabled
  IKE Fragmentation: Enabled, Size: 576
  SRG ID: 0
  Remote Access Client Info: Unknown Client
  Peer ike-id: 172.18.10.2
  AAA assigned IP: 0.0.0.0
  PPK-profile: km_profile_1
     Optional: No
     State   : Used
  Algorithms:
   Authentication        : hmac-sha256-128
   Encryption            : aes256-cbc
   Pseudo random function: hmac-sha256
   Diffie-Hellman group  : DH-group-14
  Traffic statistics:
   Input  bytes  :                 1058
   Output bytes  :                 1074
   Input  packets:                    4
   Output packets:                    4
   Input  fragmented packets:       0
   Output fragmented packets:       0
  IPSec security associations: 4 created, 1 deleted
  Phase 2 negotiations in progress: 1
  IPSec Tunnel IDs: 500002
  Negotiation type: Quick mode, Role: Initiator, Message ID: 0
   Local: 172.18.10.1:500, Remote: 172.18.10.2:500
   Local identity: 172.18.10.1
   Remote identity: 172.18.10.2
   Flags: IKE SA is created
  IPsec SA Rekey CREATE_CHILD_SA exchange stats:
   Initiator stats:                                  Responder stats:
    Request Out             : 0                      Request In             : 1
    Response In             : 0                      Response Out           : 1
    No Proposal Chosen In   : 0                      No Proposal Chosen Out : 0
```

```
    Invalid KE In           : 0                     Invalid KE Out         : 0
    TS Unacceptable In      : 0                     TS Unacceptable Out    : 0
    Res DH Compute Key Fail : 0                     Res DH Compute Key Fail: 0
    Res Verify SA Fail      : 0
    Res Verify DH Group Fail: 0
    Res Verify TS Fail      : 0
```

The IKE SA is established with configured attributes. Under the "PPK-profile" output tag, name of the key-profile used for PPKs is displayed. Additionally, it mentions the optional attribute as "No" which means that using PPKs for the establishment of this IKE SA and corresponding IPsec SAs is mandatory and the State attribute displayed "Used" which means PPKs from this key-profile has been used and the IKE SA and corresponding IPSEC SAs are Quantum safe.

3. Verify ipsec security associations status

From operational mode, enter the "show security ipsec security-associations detail" command

```
user@SRX1> show security ipsec security-associations detail

ID: 500002 Virtual-system: root, VPN Name: IPSEC_VPN
  Local Gateway: 172.18.10.1, Remote Gateway: 172.18.10.2
  Traffic Selector Name: ts1
  Local Identity: ipv4(192.168.90.0-192.168.90.255)
  Remote Identity: ipv4(192.168.80.0-192.168.80.255)
  TS Type: traffic-selector
  Version: IKEv2
  Quantum Secured: Yes
  PFS group: N/A
  SRG ID: 0
  DF-bit: clear, Copy-Outer-DSCP Disabled, Bind-interface: st0.1, Policy-name: IPSEC_POL
  Port: 500, Nego#: 0, Fail#: 0, Def-Del#: 0 Flag: 0
  Multi-sa, Configured SAs# 0, Negotiated SAs#: 0
  Tunnel events:
    Thu Mar 30 2023 23:43:42: IPsec SA negotiation succeeds (1 times)
  Location: FPC 0, PIC 0, KMD-Instance 0
  Anchorship: Thread 1
  Distribution-Profile: default-profile
  Direction: inbound, SPI: 0x983a0221, AUX-SPI: 0
                              , VPN Monitoring: -
    Hard lifetime: Expires in 1330 seconds
    Lifesize Remaining:  Unlimited
    Soft lifetime: Expires in 662 seconds
    Mode: Tunnel(0 0), Type: dynamic, State: installed
    Protocol: ESP, Authentication: hmac-sha256-128, Encryption: aes-cbc (256 bits)
    Anti-replay service: counter-based enabled, Replay window size: 64
    Extended-Sequence-Number: Disabled
    tunnel-establishment: establish-tunnels-immediately
    IKE SA Index: 1
  Direction: outbound, SPI: 0x4112746b, AUX-SPI: 0
                              , VPN Monitoring: -
    Hard lifetime: Expires in 1330 seconds
    Lifesize Remaining:  Unlimited
    Soft lifetime: Expires in 662 seconds
    Mode: Tunnel(0 0), Type: dynamic, State: installed
```

```
    Protocol: ESP, Authentication: hmac-sha256-128, Encryption: aes-cbc (256 bits)
    Anti-replay service: counter-based enabled, Replay window size: 64
    Extended-Sequence-Number: Disabled
    tunnel-establishment: establish-tunnels-immediately
    IKE SA Index: 1
```

The IPsec SA is established with configured attributes. Under the "Quantum Secured" output tag, it is mentioned as yes which means the IPsec SA is quantum safe.

4. Verify IPsec SA encryption/decryption statistics

From operational mode, enter the "show security ipsec statistics" command

```
user@SRX1> show security ipsec statistics
ESP Statistics:
  Encrypted bytes:              624
  Decrypted bytes:              624
  Encrypted packets:              4
  Decrypted packets:              4
AH Statistics:
  Input bytes:                    0
  Output bytes:                   0
  Input packets:                  0
  Output packets:                 0
Errors:
  AH authentication failures: 0, Replay errors: 0
  ESP authentication failures: 0, ESP decryption failures: 0
  Bad headers: 0, Bad trailers: 0
  Invalid SPI: 0, TS check fail: 0
  Exceeds tunnel MTU: 0
  Discarded: 0
```

For the 4 ping packets that were sent from HOST1 to HOST2, the count matches the encrypted packets and decrypted packets statistics.

5. Verify key-profile status

From operational mode, enter the "show security key-manager profiles detail" command

```
user@SRX1> show security key-manager profiles detail
Name: km_profile_1, Index: 1, Type: Static
  Configured-at: 30.03.23 (23:22:43)
  Time-elapsed: 1 hrs 16 mins 3 secs
  Request stats:
    Received: 1
    In-progress: 0
    Success: 1
    Failed: 0
```

For the key profile that was used for the establishment of the IKE/IPsec SAs, it displays the number of times a request for corresponding PPK has been made, and it shows the "Received" and "Success" output tag values as 1.

Configuring Quantum-safe IPsec VPNs with quantum-key-manager key-profile

This example shows how to configure IPsec VPN with quantum-key-manager key-profile for PPKs to allow data to be securely transferred between two sites.

We configure a route-based IPsec VPN on SRX1 and SRX2 with static key-profile configuration used for PPKs. A quantum-key-manager key profile consists of parameters which would be required to communicate with a KME/QKD device as per ETSI GS QKD 014 specification. The profile must be configured with an https url to access the server and local-sae-id, but as a recommendation for having the maximum security on the communication, local-certificate and trusted-cas should also be configured for mutual TLS authentication. HOST1 and HOST2 use the VPN to send data securely over the Internet between both hosts.

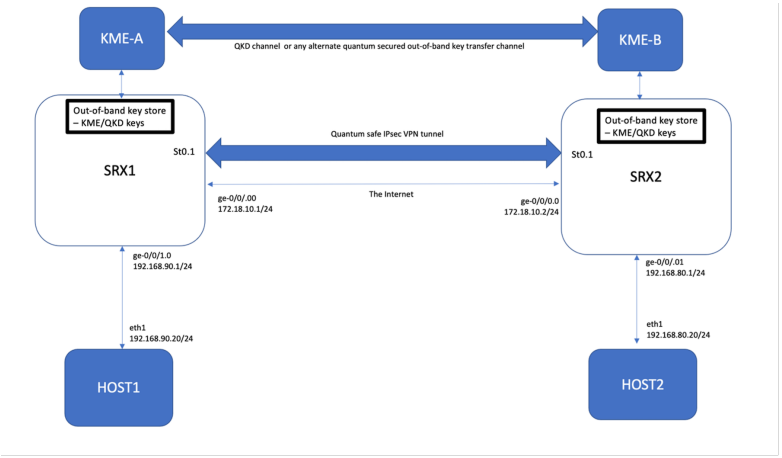

Figure 4.3 *Topology for Quantum-safe IPsec VPN Using Quantum-key-manager key-profile for PPKs*

CLI Configurations

To quickly configure SRX1

```
set security pki ca-profile ROOT_CA_CERT ca-identity RootCA
set security key-manager profiles km_profile_1 quantum-key-manager url https://www.kme_a-qkd-server.
net
set security key-manager profiles km_profile_1 quantum-key-manager local-sae-id SAE_A
set security key-manager profiles km_profile_1 quantum-key-manager local-certificate-id SAE_A_CERT
set security key-manager profiles km_profile_1 quantum-key-manager trusted-cas ROOT_CA_CERT
set security ike proposal IKE_PROP authentication-method pre-shared-keys
```

```
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.2
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.1
set security ike gateway IKE_GW version v2-only
set security ike gateway IKE_GW ppk-profile km_profile_1
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.90.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN establish-tunnels immediately
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.1/24
set interfaces ge-0/0/1 unit 0 family inet address 192.168.90.1/24
set interfaces st0 unit 1 family inet
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
set security zones security-zone vpn interfaces st0.1
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

Step-by-Step Procedure to configure SRX1

1. Configure a key profile of type quantum-key-manager with the must or recommended parameters.

Define the CA certificate. Different parameters for the CA certificate can be configured as per requirement. The corresponding CA certificate should be loaded using “request security pki ca-certificate <enroll | load>” as per requirement.

```
set security pki ca-profile ROOT_CA_CERT ca-identity RootCA
```

Configure the url of the KME server. The url must be IP resolvable. The hostname portion of the url (format:https://[hostname][:port]) can either be a fqdn or an IP address. Irrespective of the type of the hostname, the exact hostname string must be present in the server certificate, either as a Common Name or as one of Subject Alternative Names.

Also, the server should be reachable via the default routing instance.

```
set security key-manager profiles km_profile_1 quantum-key-manager url https://www.kme_a-qkd-server.
net
```

Configure the SAE-ID to be used by the local end. This configuration is not used for communicating wth the KME server but gets used by client applications (in this case IPsec vpn).

```
set security key-manager profiles km_profile_1 quantum-key-manager local-sae-id SAE_A
```

Configure the corresponding certificate for the local SAE-ID. The local certificate should be loaded beforehand using "request security pki local-certificate <enroll | load>" as per requirement. This certificate will be used and sent for client authentication on the server side. The Common Name/Subject Alternative Name used in the certificate must match the string configured for "local-sae-id" as the KME server uses the value from the Common Name or one of the Subject Alternative Names from the certificate to know what the local end SAE-ID, as per implementation of the KME.

```
set security key-manager profiles km_profile_1 quantum-key-manager local-certificate-id SAE_A_CERT
```

Configure the previously defined CA certificate. More than one CA certificate can also be configured as per requirement. The CA certificates must be configured under [security pki ca-profile] and all the required CA certificates should be loaded beforehand using "request security pki ca-certificate <enroll | load>" as per requirement. The CA certificate(s) will be used during server certificate authentication on the local end. The server certificate must be signed using the CAs configured.

```
set security key-manager profiles km_profile_1 quantum-key-manager trusted-cas ROOT_CA_CERT
```

2. Configure the interfaces

Configure the external interface

```
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.1/24
```

Configure the secure tunnel interface

```
set interfaces st0 unit 1 family inet
```

Configure the HOST1 facing interface

```
set interfaces ge-0/0/1 unit 0 family inet address 192.168.90.1/24
```

3. Configure the security zones

Configure the untrust security zone with limiting the host-inbound system services to just ike and assign the external interface to the untrust security zone.

```
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
```

Configure the vpn security zone with assigning the secure tunnel interface and other necessary paramters

```
set security zones security-zone vpn interfaces st0.1
```

Configure the trust security zone by limiting the host-inbound system services to just ping and assign the HOST1 facing interface to the trust security zone.

```
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
```

4. Configure the security policies

Configure the security policy for the transit traffic flowing from trust zone to vpn zone

```
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
```

Configure the security policy for the transit traffic flowing from vpn zone to trust zone

```
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

5. Configure the IPsec VPN

Configure an ike proposal with the necessary attributes.

```
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
```

Configure an ike policy with the configured ike proposal and other necessary attributes

```
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
```

Configure an Ike gateway with the defined ike policy and other necessary attributes.

```
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.2
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.1
set security ike gateway IKE_GW version v2-only
```

Configure the previously defined static key-profile to the ike gateway to be used as ppk-profile for PPKs. ppk-profile can only be configured if the ike gateway is configured with [version v2-only].

```
set security ike gateway IKE_GW ppk-profile km_profile_1
```

Configure an ipsec proposal with the necessary attributes

```
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec proposal IPSEC_PROP lifetime-seconds 2400
```

Configure an ipsec policy with the configured proposal

```
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
```

Configure an IPsec vpn with the previously configured secure tunnel interface, ike gateway, ipsec policy and any other necessary parameters. A Traffic selector is configured to ensure only HOST1 subnet to HOST2 subnet communication is secured using the negotiated IPsec tunnel.

```
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.90.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN establish-tunnels immediately
```

To quickly configure SRX2:

```
set security pki ca-profile ROOT_CA_CERT ca-identity RootCA
set security key-manager profiles km_profile_1 quantum-key-manager url https://www.kme_b-qkd-server.
net
set security key-manager profiles km_profile_1 quantum-key-manager local-sae-id SAE_B
set security key-manager profiles km_profile_1 quantum-key-manager local-certificate-id SAE_B_CERT
set security key-manager profiles km_profile_1 quantum-key-manager trusted-cas ROOT_CA_CERT
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text “ipsec-test”
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.1
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.2
set security ike gateway IKE_GW version v2-only
set security ike gateway IKE_GW ppk-profile km_profile_1
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.90.0/24
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.2/24
set interfaces ge-0/0/1 unit 0 family inet address 192.168.80.1/24
```

```
set interfaces st0 unit 1 family inet
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
set security zones security-zone vpn interfaces st0.1
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

Step-by-Step Procedure to configure SRX2

1. Configure a key profile of type quantum-key-manager with the must or recommended parameters

Define the CA certificate. Different parameters for the CA certificate can be configured as per requirement. The corresponding CA certificate should be loaded using "request security pki ca-certificate <enroll | load>" as per requirement.

```
set security pki ca-profile ROOT_CA_CERT ca-identity RootCA
```

Configure the url of the KME server. The url must be IP resolvable. The hostname portion of the url (format:https://[hostname][:port]) can either be a fqdn or an IP address. Irrespective of the type of the hostname, the exact hostname string must be present in the server certificate, either as a Common Name or as one of Subject Alternative Names. Also, the server should be reachable via the default routing instance.

```
set security key-manager profiles km_profile_1 quantum-key-manager url https://www.kme_b-qkd-server.
net
```

Configure the SAE-ID to be used by the local end. This configuration is not used for communicating wth the KME server but gets used by client applications (in this case IPsec vpn).

```
set security key-manager profiles km_profile_1 quantum-key-manager local-sae-id SAE_B
```

Configure the corresponding certificate for the local SAE-ID. The local certificate should be loaded beforehand using "request security pki local-certificate <enroll | load>" as per requirement. This certificate will be used and sent for client authentication on the server side. The Common Name/Subject Alternative Name used in the certificate must match the string configured for "local-sae-id" as the KME server uses the value from the Common Name or one of the Subject Alternative Names from the certificate to know what the local end SAE-ID, as per implementation of the KME.

```
set security key-manager profiles km_profile_1 quantum-key-manager local-certificate-id SAE_B_CERT
```

Configure the previously defined CA certificate. More than one CA certificate can also be configured as per requirement. The CA certificates must be configured under [security pki ca-profile] and all the required CA certificates should be loaded beforehand using "`request security pki ca-certificate <enroll | load>`" as per requirement. The CA certificate(s) will be used during server certificate authentication on the local end. The server certificate must be signed using the CAs configured.

```
set security key-manager profiles km_profile_1 quantum-key-manager trusted-cas ROOT_CA_CERT
```

2. Configure the interfaces

Configure the external interface

```
set interfaces ge-0/0/0 unit 0 family inet address 172.18.10.2/24
```

Configure the secure tunnel interface

```
set interfaces st0 unit 1 family inet
```

Configure the HOST facing interface

```
set interfaces ge-0/0/1 unit 0 family inet address 192.168.80.1/24
```

3. Configure the security zones

Configure the untrust security zone with limiting the host-inbound system services to just ike and assign the external interface to the untrust security zone.

```
set security zones security-zone untrust host-inbound-traffic system-services ike
set security zones security-zone untrust interfaces ge-0/0/0.0
```

Configure the vpn security zone with assigning the secure tunnel interface and other necessary parameters

```
set security zones security-zone vpn interfaces st0.1
```

Configure the trust security zone by limiting the host-inbound system services to just ping and assign the HOST1 facing interface to the trust security zone.

```
set security zones security-zone trust host-inbound-traffic system-services ping
set security zones security-zone trust interfaces ge-0/0/1.0
```

4. Configure the security policies

Configure the security policy for the transit traffic flowing from trust zone to vpn zone

```
set security policies from-zone trust to-zone vpn policy vpn_out match source-address any
set security policies from-zone trust to-zone vpn policy vpn_out match destination-address any
set security policies from-zone trust to-zone vpn policy vpn_out match application any
set security policies from-zone trust to-zone vpn policy vpn_out then permit
```

Configure the security policy for the transit traffic flowing from vpn zone to trust zone

```
set security policies from-zone vpn to-zone trust policy vpn_in match source-address any
set security policies from-zone vpn to-zone trust policy vpn_in match destination-address any
set security policies from-zone vpn to-zone trust policy vpn_in match application any
set security policies from-zone vpn to-zone trust policy vpn_in then permit
```

5. Configure the IPsec VPN

Configure an ike proposal with the necessary attributes.

```
set security ike proposal IKE_PROP authentication-method pre-shared-keys
set security ike proposal IKE_PROP dh-group group14
set security ike proposal IKE_PROP authentication-algorithm sha-256
set security ike proposal IKE_PROP encryption-algorithm aes-256-cbc
set security ike proposal IKE_PROP lifetime-seconds 3600
```

Configure an ike policy with the configured ike proposal and other necessary attributes

```
set security ike policy IKE_POL proposals IKE_PROP
set security ike policy IKE_POL pre-shared-key ascii-text "ipsec-test"
```

Configure an Ike gateway with the defined ike policy and other necessary attributes.

```
set security ike gateway IKE_GW ike-policy IKE_POL
set security ike gateway IKE_GW address 172.18.10.1
set security ike gateway IKE_GW external-interface ge-0/0/0.0
set security ike gateway IKE_GW local-address 172.18.10.2
set security ike gateway IKE_GW version v2-only
```

Configure the previously defined static key-profile to the ike gateway to be used as ppk-profile for PPKs. ppk-profile can only be configured if the ike gateway is configured with [version v2-only].

```
set security ike gateway IKE_GW ppk-profile km_profile_1
```

Configure an ipsec proposal with the necessary attributes.

```
set security ipsec proposal IPSEC_PROP protocol esp
set security ipsec proposal IPSEC_PROP authentication-algorithm hmac-sha-256-128
set security ipsec proposal IPSEC_PROP encryption-algorithm aes-256-cbc
set security ipsec proposal IPSEC_PROP lifetime-seconds 2400
```

Configure an ipsec policy with the configured proposal.

```
set security ipsec policy IPSEC_POL proposals IPSEC_PROP
```

Configure an IPsec vpn with the previously configured secure tunnel interface, ike gateway, ipsec policy and any other necessary parameters. A Traffic selector is configured to ensure only HOST2 subnet to HOST1 subnet communication is secured using the negotiated IPsec tunnel.

```
set security ipsec vpn IPSEC_VPN bind-interface st0.1
set security ipsec vpn IPSEC_VPN ike gateway IKE_GW
set security ipsec vpn IPSEC_VPN ike ipsec-policy IPSEC_POL
set security ipsec vpn IPSEC_VPN traffic-selector ts1 local-ip 192.168.80.0/24
set security ipsec vpn IPSEC_VPN traffic-selector ts1 remote-ip 192.168.90.0/24
```

To quickly configure HOST1 and HOST2

Exact steps to configure the HOSTs is out of scope as it will vary from device to device, but on a generic note, HOST1 must be configured with any IP in the subnet

192.168.90.0/24, other than 192.169.90.1, on its interface connected to SRX1 and HOST2 must be configured with any IP in the subnet 192.168.80.0/24, other than 192.169.00.1, on its interface connected to SRX2.

Verification and Troubleshooting

Verify the traffic flow

Ping from HOST1 to HOST2 or vice versa.

```
user@HOST1# ping 192.168.80.20 source 192.168.90.20 count 5
PING 192.168.80.20 (192.168.80.20): 56 data bytes count 5
64 bytes from 192.168.80.1: icmp_seq=0 ttl=64 time=0.998 ms
64 bytes from 192.168.80.1: icmp_seq=1 ttl=64 time=1.594 ms
64 bytes from 192.168.80.1: icmp_seq=2 ttl=64 time=1.395 ms
64 bytes from 192.168.80.1: icmp_seq=3 ttl=64 time=1.536 ms
64 bytes from 192.168.80.1: icmp_seq=4 ttl=64 time=1.838 ms

--- 192.168.80.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.998/1.472/1.838/0.277 ms
```

Data traffic is successfully flowing between the HOSTs

2. Verify ike security associations status

From operational mode, enter the "show security ike security-associations detail" command.

```
user@SRX1> show security ike security-associations detail
IKE peer 172.18.10.2, Index 21, Gateway Name: IKE_GW
  Role: Initiator, State: UP
  Initiator cookie: 5a417d46cef3207d, Responder cookie: 57b9a17516bee31b
  Exchange type: IKEv2, Authentication method: Pre-shared-keys
  Local gateway interface: ge-0/0/2.0
  Routing instance: default
  Local: 172.18.10.1:500, Remote: 172.18.10.2:500
  Lifetime: Expires in 3445 seconds
  Reauth Lifetime: Disabled
  IKE Fragmentation: Enabled, Size: 576
  SRG ID: 0
  Remote Access Client Info: Unknown Client
  Peer ike-id: 172.18.10.2
  AAA assigned IP: 0.0.0.0
  PPK-profile: km_profile_1
     Optional: No
     State   : Used
  Algorithms:
   Authentication        : hmac-sha256-128
   Encryption            : aes256-cbc
   Pseudo random function: hmac-sha256
   Diffie-Hellman group  : DH-group-14
  Traffic statistics:
   Input  bytes  :                 783
   Output bytes  :                 831
```

```
   Input  packets:                    2
   Output packets:                    2
   Input  fragmented packets:       0
   Output fragmented packets:       0
  IPSec security associations: 2 created, 0 deleted
  Phase 2 negotiations in progress: 1
  IPSec Tunnel IDs: 500003
    Negotiation type: Quick mode, Role: Initiator, Message ID: 0
    Local: 172.18.10.1:500, Remote: 172.18.10.2:500
    Local identity: 172.18.10.1
    Remote identity: 172.18.10.2
    Flags: IKE SA is created
  IPsec SA Rekey CREATE_CHILD_SA exchange stats:
   Initiator stats:                                  Responder stats:
    Request Out             : 0                      Request In             : 0
    Response In             : 0                      Response Out           : 0
    No Proposal Chosen In   : 0                      No Proposal Chosen Out : 0
    Invalid KE In           : 0                      Invalid KE Out         : 0
    TS Unacceptable In      : 0                      TS Unacceptable Out    : 0
    Res DH Compute Key Fail : 0                      Res DH Compute Key Fail: 0
    Res Verify SA Fail      : 0
    Res Verify DH Group Fail: 0
    Res Verify TS Fail      : 0
```

The IKE SA is established with configured attributes. Under the "Ppk-profile" output tag, name of the key-profile used for PPKs is displayed. Additionally, it mentions the optional attribute as "No" which means that using PPKs for the establishment of this IKE SA and corresponding IPsec SAs is mandatory and the State attribute displayed "Used" which means PPKs from this key-profile has been used and the IKE SA and corresponding IPSEC SAs are Quantum safe.

3. Verify ipsec security associations status.

From operational mode, enter the "show security ipsec security-associations detail" command

```
user@SRX1> show security ipsec security-associations detail

ID: 500003 Virtual-system: root, VPN Name: IPSEC_VPN
  Local Gateway: 172.18.10.1, Remote Gateway: 172.18.10.2
  Traffic Selector Name: ts1
  Local Identity: ipv4(192.168.90.0-192.168.90.255)
  Remote Identity: ipv4(192.168.80.0-192.168.80.255)
  TS Type: traffic-selector
  Version: IKEv2
  Quantum Secured: Yes
  PFS group: N/A
  SRG ID: 0
  DF-bit: clear, Copy-Outer-DSCP Disabled, Bind-interface: st0.1, Policy-name: IPSEC_POL
  Port: 500, Nego#: 0, Fail#: 0, Def-Del#: 0 Flag: 0
  Multi-sa, Configured SAs# 0, Negotiated SAs#: 0
  Tunnel events:
    Fri Mar 31 2023 01:41:52: IPsec SA negotiation succeeds (1 times)
  Location: FPC 0, PIC 0, KMD-Instance 0
  Anchorship: Thread 1
```

```
  Distribution-Profile: default-profile
  Direction: inbound, SPI: 0xd1e1549c, AUX-SPI: 0
                              , VPN Monitoring: -
    Hard lifetime: Expires in 1916 seconds
    Lifesize Remaining:  Unlimited
    Soft lifetime: Expires in 1349 seconds
    Mode: Tunnel(0 0), Type: dynamic, State: installed
    Protocol: ESP, Authentication: hmac-sha256-128, Encryption: aes-cbc (256 bits)
    Anti-replay service: counter-based enabled, Replay window size: 64
    Extended-Sequence-Number: Disabled
    tunnel-establishment: establish-tunnels-immediately
    IKE SA Index: 21
  Direction: outbound, SPI: 0xb5883167, AUX-SPI: 0
                              , VPN Monitoring: -
    Hard lifetime: Expires in 1916 seconds
    Lifesize Remaining:  Unlimited
    Soft lifetime: Expires in 1349 seconds
    Mode: Tunnel(0 0), Type: dynamic, State: installed
    Protocol: ESP, Authentication: hmac-sha256-128, Encryption: aes-cbc (256 bits)
    Anti-replay service: counter-based enabled, Replay window size: 64
    Extended-Sequence-Number: Disabled
    tunnel-establishment: establish-tunnels-immediately
    IKE SA Index: 21
```

The IPsec SA is established with configured attributes. Under the "Quantum Secured" output tag, it is mentioned as yes which means the IPsec SA is quantum safe.

4. Verify IPsec SA encryption/decryption statistics.

From operational mode, enter the "show security ipsec statistics" command

```
user@SRX1> show security ipsec statistics
ESP Statistics:
  Encrypted bytes:              780
  Decrypted bytes:              780
  Encrypted packets:              5
  Decrypted packets:              5
AH Statistics:
  Input bytes:                    0
  Output bytes:                   0
  Input packets:                  0
  Output packets:                 0
Errors:
  AH authentication failures: 0, Replay errors: 0
  ESP authentication failures: 0, ESP decryption failures: 0
  Bad headers: 0, Bad trailers: 0
  Invalid SPI: 0, TS check fail: 0
  Exceeds tunnel MTU: 0
  Discarded: 0
```

For the 5 ping packets that were sent from HOST1 to HOST2, the count matches the encrypted packets and decrypted packets statistics.

5. Verify key-profile status.

From operational mode, enter the "show security key-manager profiles detail" command:

```
user@SRX1> show security key-manager profiles detail
Name: km_profile_1, Index: 3, Type: Quantum-key-manager
  Configured-at: 31.03.23 (01:40:50)
  Time-elapsed: 0 hrs 11 mins 30 secs
  Url: https://www.kme_a-qkd-server.net
  Local-sae-id: SAE_A
  Local-certificate-id: SAE_A_CERT
  Trusted-cas: [ ROOT_CA_CERT ]
  Peer-sae-ids: N/A
  Default-key-size: N/A
  Request stats:
    Received: 1
    In-progress: 0
    Success: 1
    Failed: 0
```

For the key profile that was used for the establishment of the IKE/IPsec SAs, it displays the number of times a request for corresponding PPK has been made, and it shows the "Received" and "Success" output tag values as 1.

Appendices

IPsec VPN PPK-profile Additional Use Cases

An optional flag can be configured under [security ike gateway <gw-name> ppk-profile <key-profile-name>]

Example:

```
set security ike gateway IKE_GW ppk-profile km_profile_1 optional
```

RFC 8784 mentions "implementations have a configurable flag that determines whether this PPK is mandatory". The idea behind having this optional flag is for inter-operability with VPN gateways (can be from other vendors) which do not support RFC 8784 and to establish at least non-quantum safe SAs if quantum safe SAs can't be established.

By default, when ppk-profile is configured, it indicates the negotiated IKE and IPSEC SAs need to be quantum-safe or else abort the negotiation. However, setting the Optional flag at both the endpoints would mean that during negotiation of IKE/IPsec SAs, if due to some reason quantum safe SAs cannot be established, then at least the peers will not abort the ongoing negotiation and rather fallback into negotiating normal IKE/IPSEC SAs.

All the decision making occurs during runtime in a single key exchange session, without first aborting the quantum-safe SA creation session and then initiating a new non-quantum safe SA creation session, thus significantly reducing key-exchange overhead.

Here is a list of scenarios related to the optional ppk-profile feature:

Scenario 1: If Initiator is configured with PPK but with optional flag and Responder is not configured with PPK (maybe it doesn't support RFC 8784), it would result in Normal

IKE and IPSEC SAs getting negotiated.

Scenario 2: If Initiator is configured with PPK without optional flag and Responder is not configured with PPK (maybe it doesn't support RFC 8784), it would result in the negotiation getting aborted.

Scenario 3: When Initiator is not configured with PPK (maybe it doesn't support RFC 8784) and Responder with PPK without optional flag, it would result in Responder aborting the negotiation.

Scenario 4: When Initiator and Responder both are configured with PPK with optional flag, it would result in quantum-safe IKE and IPSEC SAs using the PPK if no error was encountered. PPK will always be used when both the endpoints are configured with PPK-profile (with/without optional flag) and no issue was encountered.

Important Notes for PPK-profile Usage

1. A PPK profile can only be used for IKEv2 based IPsec VPN solutions.

2. In case using IKEv1 is necessary, then configuring IKEv1 with pre-shared key is quantum safe.

3. As of 22.4R1, the feature set is only supported with Site-to-Site IPsec VPN and Auto-VPN configurations.

4. The ike-key-management process will fetch PPK, as per the configured ppk-profile, from the junos-key-manager daemon only during Initial IKEv2 negotiation (to comply with RFC 8784). To have maximum security, it is recommended to perform REAUTH (performing the Initial IKEv2 negotiation rather than IKE rekey) at regular intervals. Reauth frequency can be set via [security ike policy <policy-name> reauth-frequency <number>].

Junos Key Manager Configuration Statements

The Junos Key Manager daemon manages the configuration under [security key-manager]. The key-manager config is structured around configuring key profiles. As of 22.4R1, two types of key profiles are supported – static and quantum-key-manager.

Configuration statements

Profiles

Syntax:

```
profiles {
```

```
    profile-name {
        quantum-key-manager {
            url    url-value;
            local-sae-id   local-sae-id-value;
            local-certificate-id   local-certificate-id-value;
            peer-sae-ids   list-of-peer-sae-ids;
            default-key-size   key-size-value;
            trusted-cas   list-of-ca-profile-ids;
        }
        static {
            key-id [ascii-text | hexadecimal]   key-id-value;
            key [ascii-text | hexadecimal]   key-value;
        }
    }
}
```

Hierarchy level:

[edit security key-manager]

Description: Configure a key profile for out-of-band key retrieval and usage.

Options:

profile-name	Specify name of the key profile.
key-id	Specify the ascii-encoded or hexadecimal-encoded string representing a key-id. This parameter can only be configured for static key profiles.
key	Specify the ascii-encoded or hexadecimal-encoded string representing a secret key for the corresponding key-id. This parameter can only be configured for static key profiles.
url	Specify the url of the KME server. The url must be IP resolvable. The hostname portion of the url (format:https://[hostname][:port]) casleswayball@juniper.netn either be a fqdn or an IP address. Irrespective of the type of the hostname, the exact hostname string must be present in the server certificate, either as a Common Name or as one of Subject Alternative Names. Also, the server should be reachable via the default routing instance. This parameter can only be configured for quantum-key-manager key profiles.
local-sae-id	Specify the SAE-ID to be used by the local end. This configuration is not used for communicating wth the KME server but gets used by client applications. This parameter can only be configured for quantum-key-manager key profiles.
peer-sae-ids	Specify the SAE-IDs for the remote applications that the KME server needs, to generate and send QKD key(s). It is used, in conjunction with loca-sae-id, as identifier(s) by the KME server as to whom to send the generated keys. This parameter can only be configured for quantum-key-manager key profiles and is an optional parameter. If no peer-sae-ids are configured, then the application requesting the keys needs to supply the peer-sae-ids, or else key-request will fail. Note when using quantum-key-manager key-profiles for VPN, no need to configure this parameter.

local-certificate-id	Specify the certificate for the local SAE-ID. The local certificate should be loaded beforehand using "request security pki local-certificate <enroll | load>" as per requirement. This certificate will be used and sent for client authentication on the server side. The Common Name/Subject Alternative Name used in the certificate must match the string configured for "local-sae-id" as the KME server uses the value from the Common Name or one of the Subject Alternative Names from the certificate to know what the local end SAE-ID, as per implementation of the KME. This parameter can only be configured for quantum-key-manager key profiles and is an optional parameter. If local-certificate-id is not configured, then the corresponding certificate won't be sent to the KME server for client authentication.
trusted-cas	Specify one or more CA certificate(s). The CA certificates must be configured under [security pki ca-profile] and all the required CA certificates should be loaded beforehand using "request security pki ca-certificate <enroll | load>" as per requirement. The CA certificate(s) will be used during server certificate authentication on the local end. The server certificate must be signed using the CAs configured. This parameter can only be configured for quantum-key-manager key profiles and is an optional parameter. If trusted-cas is not configured, then server certificate authentication won't be performed.
default-key-size	Specify the default key size that will be used when KME server is requested for QKD key(s). This parameter can only be configured for quantum-key-manager key profiles and is an optional parameter. If default-key-size is not explicitly configured, then the default key size will be 256 bits.

Traceoptions

Syntax:

```
traceoptions {
    file {
        filename;
        size  max-file-size;
        files  number-of-files;
        match  regular-expression;
        [no-world-readable | world-readable];
    }
    level (critical | error | terse | warning | detail | extensive);
     no-remote-trace;
}
```

Hierarchy level:

```
[edit security key-manager]
```

Description:

Configure junos-key-manager tracing options to aid in troubleshooting the key-manager issues. This allows the user to view the detailed application logs for all the operations

performed on a key profile. Key-manager tracing is enabled by default. By default, the application logs are written in /var/log/jkm.

Option:

File–Configure the trace file options.

filename–Name of the file to receive the output of the tracing operation. Enclose the name within quotation marks. All files are placed in the directory /var/log.

Default: jkm

files number–Maximum number of trace files. When a trace file named trace-file reaches its maximum size, it is renamed to trace-file.0, then trace-file.1, and so on, until the maximum number of trace files is reached. The oldest archived file is overwritten.

If you specify a maximum number of files, you also must specify a maximum file size with the size option and a filename.

Range: 2 through 1000 files

Default: 10 files

- match regular-expression–Refine the output to include lines that contain the regular expression.
- size maximum-file-size–Maximum size of each trace file, in kilobytes (KB), megabytes (MB), or gigabytes (GB). When a trace file named trace-file reaches this size, it is renamed trace-file.0. When the trace-file again reaches its maximum size, trace-file.0 is renamed trace-file.1 and trace-file is renamed trace-file.0. This renaming scheme continues until the maximum number of trace files is reached. Then the oldest trace file is overwritten.

If you specify a maximum file size, you also must specify a maximum number of trace files with the files option and filename.

Syntax: x k to specify KB, x m to specify MB, or x g to specify GB

Range: 10 KB through 1 GB

Default: 1024 KB

world-readable | no-world-readable–By default, log files can be accessed only by the user who configures the tracing operation. The world-readable option enables any user to read the file. To explicitly set the default behavior, use the no-world-readable option.

level–Specify the log levels

critical–Log single point failures which needs your immediate attention

error–Log fatal application errors

terse–Log syslog messages

warning–Log recoverable errors

detail–Log all operational information

extensive–Log everything.

no-remote-trace–Set remote tracing as disabled.

Junos Key Manager Operational Commands

Show security key-manager clients

Syntax:

```
show security key-manager clients <index index> <brief | detail>
```

Description:

Display client applications/services currently connected to junos-key-manager and related runtime information.

Options:

- none–Display standard information of all client applications/services, including index numbers.
- index client-index–(Optional) Display information for a particular client application/service for a given index number of the client.
- brief–(Optional) Display standard information about all the client applications/services connected to junos-key-manager. (Default)
- detail–(Optional) Display detailed information about all the client applications/services connected to junos-key-manager .

Output fields:

Index: Index of the client application which is currently connected to JKM.

Name: Name of the client application which is currently connected to JKM.

Pid: Process ID of the client application which is currently connected to JKM.

Connected at: Time at which the client application connected.

Time elapsed: Duration of time since the client application is connected.

Request stats: Statistics related to requests received from the client

Received: Number of requests received from the client

In-progress: Number of requests currently being processed

Success: Number of requests that were successfully processed

Failed: Number of requests that were unsuccessful

Sample output:

```
user@SRX> show security key-manager clients

  Index       Name               Pid         Received     In-progress   Success    Failed

  7           ikemd              66111       3            0             3          0
  8           iked               66055       8            0             3          5
user@SRX> show security key-manager clients detail

Name: ikemd, Index: 7, Pid: 66111
  Connected at: 30.03.23 (23:33:51)
  Time elapsed: 68 hrs 58 mins 12 secs
  Request stats:
    Received: 3
    In-progress: 0
    Success: 3
    Failed: 0

Name: iked, Index: 8, Pid: 66055
  Connected at: 30.03.23 (23:33:51)
  Time elapsed: 68 hrs 58 mins 12 secs
  Request stats:
    Received: 8
    In-progress: 0
    Success: 3
    Failed: 5
```

Show security key-manager profiles

Sytax:

```
show security key-manager profiles <index index> <brief | detail>
```

Description:

Display key profiles currently configured/managed by junos-key-manager and related runtime information.

Options:

- none–Display standard information of all key profiles, including index numbers.
- index profile-index - (Optional) Display information for a particular key profile for a given index number.
- brief–(Optional) Display standard information about all the key profiles managed by junos-key-manager. (Default)
- detail–(Optional) Display detailed information about all the key profiles managed by junos-key-manager.

Output fields:

Index: Index of the key profile which is currently configured.

Name: Name of the key profile which is currently configured

Type: Type of the configured key profile

Configured at: Time at which the key profile was configured

Time elapsed: Duration of time since the key profile was configured

Url: URL for KME (KME management entity) server

Local sae id: The SAE (Secure Application entity) ID for the local application which the KME server needs to generate and send QKD key(s).

Local cert id: The certificate id for the corresponding certificate when client authentication needs to be done with the KME server

Peer sae ids: The SAE IDs for the remote applications that the KME server needs, to generate and send QKD key(s)

Trusted cas: CA(s) that will be considered as valid when server certification verification is performed.

Default key size: The default key size that will be used when KME server is requested for QKD key(s)

Request stats: Statistics related to requests received for the key profile

Received: Number of requests received for the key profile

In-progress: Number of requests currently being processed

Success: Number of requests that were successfully processed

Failed: Number of requests that were unsuccessful

Sample output:

```
user@SRX> show security key-manager profiles

  Index        Name               Received     In-progress    Success      Failed

  3            km_profile_1       3            0              3            0


user@SRX> show security key-manager profiles detail

Name: km_profile_1, Index: 3, Type: Quantum-key-manager
  Configured-at: 31.03.23 (01:40:50)
  Time-elapsed: 67 hrs 1 mins 35 secs
  Url: https://www.kme_a-qkd-server.net:5000
  Local-sae-id: SAE_A
  Local-certificate-id: SAE_A_CERT
```

```
Trusted-cas: [ ROOT_CA_CERT ]
Peer-sae-ids: N/A
Default-key-size: N/A
Request stats:
  Received: 3
  In-progress: 0
  Success: 3
  Failed: 0
```

Show security key-manager statistics

Syntax:

show security key-manager statistics <brief>

Description:

Display global statistics information for junos-key-manager.

Options:

- none –Display standard statistics information.
- brief–(Optional) Display standard statistics information for junos-key-manager. (Default)

Output fields:

Statistics based on profile type: Stats categorized based profile type

Profile type: Type of configured profile

Config-count: Number of configured profiles for a type

Received: Number of requests received for a profile type

In-progress: Number of requests currently being processed.

Success: Number of requests that were successfully processed

Failed: Number of requests that were unsuccessful

Statistics based on operation type: Stats categorized based request type

Request type: Type of request supported.

Received: Number of requests received for a profile type.

In-progress: Number of requests currently being processed.

Success: Number of requests that were successfully processed.

Failed: Number of requests that were unsuccessful.

Request Error Stats: Stats for various request status codes

Failed: Display count of all requests that failed due to unknown/generic error.

Invalid operation type: Display count of all requests that failed due to Invalid operation type.

Profile not found: Display count of all requests that failed due to Profile not found.

Invalid argument(s) for profile type: Display count of all requests that failed due to some invalid argument provided as part of the request. Invalid argument means an argument that is not supported for the request/operation type.

External key resource request failed: Display count of all requests that failed due to some issue during preparing/making the request to the external key source.

External key resource response failed: Display count of all requests that failed due to some issue with the response from the external key source.

Aborted by profile change: Display count of all requests that failed due to ongoing requests getting aborted due to change/delete of key profile.

Aborted by client: Display count of all requests that failed due to client application/service aborting ongoing requests.

Timed out: Display count of all requests that failed due to no response and the request getting timed out.

Sample output:

```
user@SRX> show security key-manager statistics

  Statistics based on profile type:

    Key-profile-type         Config-count    Received     In-progress    Success      Failed

    Static                   0               4            0              4            0
    Quantum-key-manager      1               11           0              4            7


  Statistics based on request type:

    Request-type                   Received     In-progress    Success      Failed

    Get-profile-config             4            0              4            0
    Get-profile-status             0            0              0            0
    Get-profile-keys               11           0              4            7
    Get-profile-keys-with-id       0            0              0            0


  Request Error Stats:

  Failed                                : 0
  Invalid operation type                : 0
  Profile not found                     : 0
```

```
  Invalid argument(s) for profile type   : 0
  External key resource request failed   : 0
  External key resource response failed  : 7
  Aborted by profile change              : 0
  Aborted by client                      : 0
  Timed out                              : 0
```

Clear security key-manager clients

Syntax:

```
clear security key-manager clients <index index> <statistics>
```

Description:

Clear runtime state of client applications/services including aborting all in-progress requests, clear statistics information and disconnecting from client.

Options:

- none–Clear runtime state of all clients
- index client-index–(Optional) Clear runtime state of a particular client application/service for a given index number.
- statistics–(Optional) Clear only the statistics information of client application(s)/service(s).

Output fields: This command produces no output.

Clear security key-manager profiles

Syntax:

Clear security key-manager profiles <index index> <statistics>

Description:

Clear runtime state (if applicable as per profile type) of key profiles including aborting all in-progress requests, clear statistics information. For Quantum-key-manager profile-type, clearing the runtime state will mean closing any existing sessions with the KME server.

Options:

- none–Clear runtime state of all key profiles
- index profile-index–(Optional) Clear runtime state of a particular key profile for a given index number.
- statistics–(Optional) Clear only the statistics information of client key profile(s).

Output fields: This command produces no output.

Clear security key-manager statistics

Syntax:

```
clear security key-manager statistics
```

Description:

Clear global statistics information for junos-key-manager.

Options:

- none–Clear all statistics

Output fields: This command produces no output.

Request security key-manager profiles get profile-config

Syntax:

```
request security key-manager profiles get profile-config <name profile-name>
```

Description:

Get the config of a key profile.

Options:

- name –Name of the key profile

Output fields:

When you enter this command, you are provided feedback on the status of your request. Output fields will differ based on the key profile type.

Sample output:

```
user@SRX> request security key-manager profiles get profile-config name km_profile_1

   - Response:
     - Status: SUCCESS
     - Name: km_profile_1
     - Type: Static


user@SRX> request security key-manager profiles get profile-config name km_profile_1

  - Response:
    - Status: SUCCESS
    - Name: km_profile_1
    - Type: Quantum-key-manager
    - Url: https://www.kme_a-qkd-server.net
    - Local-sae-id: SAE_A
    - Local-certificate-id: SAE_A_CERT
    - Trusted-cas: [ ROOT_CA_CERT ]
    - Peer-sae-ids: N/A
```

```
    - Default-key-size: N/A
```

In case of failure:

```
user@SRX> request security key-manager profiles get profile-config name km_profile_2

  - Response:
    - Status: FAILED
```

Request security key-manager profiles get profile-status

Syntax:

```
request security key-manager profiles get profile-status <name profile-name> <peer-sae-id peer-sae-
id-value>
```

Description:

Get the status of a key profile

Options:

- name profile-name –Name of the key profile
- peer-sae-id peer-sae-id-value – SAE-ID of the remote end application. This is only applicable for quantum-key-manager profile type. If peer-sae-ids field is configured for the given key profile, then this parameter can be skipped. If more than one peer-sae-ids in configured in the key-profile config, then the first peer-sae-id in the list will be used to run the command.

Output fields:

When you enter this command, you are provided feedback on the status of your request. Output fields will differ based on the key profile type.

Sample output:

```
user@SRX> request security key-manager profiles get profile-status name km_profile_1

   - Response:
 - Status: SUCCESS
 - Name: km_profile_1
 - Type: Static
 - Configured-at: 01.12.21 09:11:30
 - Time-elapsed: 3 hrs 10 mins 20 secs
 - Key-size: 512 bits

user@SRX> request security key-manager profiles get profile-status name km_profile_1 peer-sae-
id SAE_B

  - Response:
    - Status: SUCCESS
```

```
    - Name: km_profile_1
    - Type: Quantum-key-manager
    - Configured-at: 31.03.23 (01:40:50)
    - Time-elapsed: 68 hrs 5 mins 26 secs
    - Key-size: 256 bits
    - Source-KME-ID: kme-1
    - Target-KME-ID: kme-2
    - Master-SAE_ID: SAE_A
    - Slave-SAE_ID: SAE_B
    - Stored-key-count: 25000
    - Max-key-count: 100000
    - Max-key-per-request: 128
    - Max-key-size: 1024 bits
    - Min-key-size: 64 bits
    - Max-SAE-ID-count: 0
```

In case of failure:

```
user@SRX> request security key-manager profiles get profile-status name km_profile_2

  - Response:
    - Status: FAILED
```

Request security key-manager profiles get profile-keys

Syntax:

```
request security key-manager profiles get profile-keys <name profile-name> <peer-sae-id peer-sae-id-
value> <key-count key-count-value> <key-size key-size-value>
```

Description:

Get the key(s) for a key profile.

Options:

- name profile-name–Name of the key profile
- peer-sae-id peer-sae-id-value – SAE-ID of the remote end application. This parameter can be specified more than once to provide multiple peer-sae-ids. This is only applicable for quantum-key-manager profile type. If peer-sae-ids field is configured for the given key profile, then this parameter can be skipped. If more than one peer-sae-ids in configured in the key-profile config, then the command will be run using all the peer-sae-ids in the list.
- key-size key-size-value - Specify the size of key(s) in bits. This is only applicable for quantum-key-manager profile type. The specified key size must be a multiple of 8. 256 bits is the minimum key size that can be specified which is the default key size (unless it is changed via default-key-size configuration) used if the parameter is not specified.

- key-count key-count-value - Specify the number of keys. This is only applicable for quantum-key-manager profile type. The default key count is 1 if the parameter is not specified.

Output fields:

When you enter this command, you are provided feedback on the status of your request. Key-ids will be in ascii format and Keys will be in hexadecimal format.

Sample output:

```
user@SRX> request security key-manager profiles get profile-keys name km_profile_1

  - Response:
    - Status: SUCCESS
    - Name: km_profile_1
    - Type: Static
    - Key-size: 256 bits
    - Key-count: 1
    - Key-ids:
      - test-key-id
    - Keys:
      - abcd01e289dea1258fe37bbd8ccae26f

user@SRX> request security key-manager profiles get profile-keys name km_profile_1 peer-sae-id SAE_B

  - Response:
    - Status: SUCCESS
    - Name: km_profile_2
    - Type: Quantum-key-manager
    - Key-size: 256 bits
    - Key-count: 1
    - Key-ids:
      - 5f11a71d-ef23-4902-ac6a-14c1c289fae7
    - Keys:
      - f3dc9f10a80e82820a91f2c8dc1946fe33649eb0a40cd64bfb0e6edac7bbaac0
```

In case of failure:

```
user@SRX> request security key-manager profiles get profile-keys name km_profile_1 peer-sae-id SAE_C

  - Response:
    - Status: FAILED
```

Request security key-manager profiles get profile-keys-with-id

Syntax:

```
request security key-manager profiles get profile-keys-with-id <name profile-name> <peer-sae-
id peer-sae-id-value> <key-id key-id-value>
```

Description:

Get the key(s) for given key id(s) for a key profile.

Options:

- Name profile-name - Name of the key profile
- peer-sae-id peer-sae-id-value – SAE-ID of the remote end application. This parameter can be specified more than once to provide multiple peer-sae-ids. This is only applicable for quantum-key-manager profile type. If peer-sae-ids field is configured for the given key profile, then this parameter can be skipped. If more than one peer-sae-ids in configured in the key-profile config, then the first peer-sae-id in the list will be used to run the command.
- key-id key-id-value - Specify key-id. This parameter can be specified more than once to provide multiple key-ids.

Output fields:

When you enter this command, you are provided feedback on the status of your request. Key-ids will be in ascii format and Keys will be in hexadecimal format.

Sample output:

```
user@SRX> request security key-manager profiles get profile-keys-with-id name km_profile_1 key-
id “test-key-id”

  - Response:
    - Status: SUCCESS
    - Name: km_profile_1
    - Type: Static
    - Key-size: 256 bits
    - Key-count: 1
    - Key-ids:
      - test-key-id
    - Keys:
      - abcd01e289dea1258fe37bbd8ccae26f

user@SRX> request security key-manager profiles get profile-keys-with-id peer-sae-id SAE_A name km_
profile_1 key-id 5f11a71d-ef23-4902-ac6a-14c1c289fae7

  - Response:
    - Status: SUCCESS
    - Name: km_profile_1
    - Type: Quantum-key-manager
    - Key-size: 256 bits
    - Key-count: 1
    - Key-ids:
      - 5f11a71d-ef23-4902-ac6a-14c1c289fae7
    - Keys:
      - f3dc9f10a80e82820a91f2c8dc1946fe33649eb0a40cd64bfb0e6edac7bbaac0
```

In case of failure:

```
user@SRX> request security key-manager profiles get profile-keys name km_
```

```
profile_1 peer-sae-id SAE_B key-id 5f11a71d-ef23-4902-ac6a-14c123wr"

  - Response:
    - Status: FAILED
```

Restart key-manager

Syntax:

```
restart key-manager <gracefully | immediately | soft>
```

Description:

Restart or reinitialize the junos-key-manager daemon.

Options:

- none–Restarts the daemon
- gracefully – Gracefully restart (SIGTERM) the process.
- Immediately - Immediately restart (SIGKILL) the process
- soft - Soft reset (SIGHUP) the process

Output fields:

This command produces no output.