POLITECNICO DI TORINO

Master's degree course in Cybersecurity

Master's Degree Thesis

Post-Quantum IPsec Gateway: Policy Decision Point

Supervisors
Prof. Antonio Lioy
Dott. Flavio Ciravegna

Candidate
Simone SAMBATARO

Academic Year 2024-2025

Summary

This thesis investigates how to protect existing IPsec-based networks from emerging quantum threats without requiring an immediate upgrade of all endpoints. It focuses on environments where legacy systems and operational constraints make a direct migration to post-quantum cryptography difficult, and instead introduces a policy-driven gateway that can offer quantum-resilient protection at the network edge, working as a translator between the two "worlds". The work first reviews the impact of quantum computing on contemporary cryptography, the NIST PQC standardisation process, and the design of IPsec, IKEv2 and strongSwan, in order to motivate the need for crypto-agility and centrally managed cryptographic policies.

Building on this background, the thesis proposes a post-quantum IPsec gateway that terminates classical IKEv2 tunnels on one side and re-establishes connectivity towards post-quantum capable peers on the other, under the control of an external Policy Decision Point (PDP) implemented with Open Policy Agent (OPA). The strongSwan-based gateway acts as a Policy Enforcement Point (PEP), consulting the PDP before accepting IKE SAs, installing CHILD SAs or performing rekeys, and enforcing service-specific security levels through structured policy classes. While a decision logger, metrics exporter and dashboards give observability and current time status and security level trends.

The prototype is validated through functional tests that exercise the complete negotiation workflow, demonstrating correct policy enforcement, downgrade resistance and fail-closed behaviour in the presence of non-compliant proposals. Performance experiments provide a multi-layered assessment that jointly examines timing behaviour, network overhead and fragmentation across the four defined security levels and at each stage of secure tunnel establishment. The results show that, although decoupling policy decisions from the data plane and incorporating post-quantum certificates and signature exchanges introduces a measurable bottleneck, the additional latency is justified by the improved security posture and the deliberate verification time at each step, both of which are crucial in the intended deployment context. Overall, the findings indicate that a policy-centric, post-quantum IPsec gateway is technically feasible and offers a practical migration strategy for organisations seeking to place long-lived or hard-to-upgrade services behind a quantum-safe perimeter, while paving the way for broader adoption of quantum-resistant protocols in the future.

Acknowledgements

This thesis project was conducted in collaboration with my colleague, Leonardo Rizzo.

List of Figures

List of Tables

Listings

Contents

Chapter 1

Introduction

Within today's landscape, quantum computing has recently gained significant attention as an alternative paradigm for information processing, leveraging quantum computers' ability to create and manipulate quantum bits (qubits) to carry out computations [1] [2]. Two fundamental properties define qubits: *Superposition*, which allows a qubit to exist in a linear combination of multiple basis states simultaneously, thereby enabling parallel processing across a vast computational space and *Entanglement*, whereby a group of qubits share a single quantum state, allowing them to coordinate and perform complex computations more efficiently. Exploiting these effects, a quantum computer can act on many value combinations at once, tackling problems that outstrip the practical reach of classical and even high performance systems, commonly referred to as supercomputers.

While these capabilities bring clear advances, they also create risks for cybersecurity [3]. Today's public-key cryptosystems rely on the concept of NP problems, notably Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC), specifically on the presumed intractability of factoring large integers and solving discrete logarithms with classical resources. Quantum algorithms, most prominently Shor's algorithm, could solve these problems efficiently, undermining the security of those schemes [4]. As a proof of concept, it has been estimated that breaking a RSA 2048-bit key could be feasible with about 20 million qubits in roughly 8 h [5].

Although large scale quantum computers are not yet widely deployed, it is crucial to put countermeasures in place ahead of practical availability. In this scenario, the development of quantum-resistant solutions, known as PQC[6], becomes essential. Beginning in 2016, the NIST launched a PQC standardisation program. After several evaluation rounds, in 2022 it narrowed the field to four candidates for standardisation [7], and finally, on August 2024, NIST published the first three finalised FIPS standards, marking the first quantum-resistant cryptography standards to be issued [8].

This competition has involved industry and government organisations, including the *National Security Agency (NSA)*, providing advice that emphasises the need to switch to PQC, especially concerning software signing and traditional networking frameworks. This involves enhancing the protocols that facilitate secure connectivity, Transport Layer Security (TLS) for web security and IPsec for network layer protection.

For IPsec, the actual baseline protocol machinery is defined by RFC-4301 [9] for the security architecture and RFC-7296 for the IKEv2 protocol [10]. Recent updates introduce post-quantum aware mechanisms: RFC-9242 [11] and RFC-9370 [12] specify hybrid and PQ-ready exchanges so that IKEv2 can combine classical and post-quantum key encapsulation while preserving interoperability with existing deployments.

Migration, however, is not just a matter of "turning on" new algorithms. As early field trials (e.g. Cloudflare's hybrid TLS KEM deployments) show, organisations need time to validate end-to-end behaviour with PQC: measuring handshake latency, assessing appliance overhead, stress testing large messages and reassembly paths. Furthermore, the presence of middleboxes, hardware offload constraints and legacy endpoints complicates roll outs.

All of this suggests that *cryptographic agility* is a strategic necessity. Systems must be modular enough to switch primitives without requiring intrusive rewrites, ideally isolating changes behind stable APIs and capability negotiation at the protocol layer, as PQC schemes are still an emerging field. In a rapidly changing and possibly ever changing environment, policy-driven control is essential.

The centrally established rules that choose cryptography suites, key sizes, and acceptable risk levels for each peer, application or network segment are the foundation of this effort.

Instead than addressing cryptographic algorithms as the primary design dimension, this thesis takes a policy-centric approach, establishing policy as the governing layer that coordinates the behaviour of IPsec systems. In particular, it offers a *crypto-agile post-quantum gateway* that provides PQC protections for legacy systems that are unable to migrate natively. The proposed gateway "translates" network traffic, facilitating secure interoperability with post-quantum-ready counterparts in alignment with centrally established cryptographic policies.

Chapter 2

Quantum technology and cybersecurity

In the early twentieth century, *quantum mechanics* emerged as a theory that departs from classical determinism and governs phenomena at microscopic scales. The guiding principles are the *uncertainty principle*, which limits the simultaneous precision with which conjugate observables (e.g. position and momentum) can be known, and *superposition*, whereby a system may occupy multiple classically exclusive states until measurement collapses the state.

In the standard probabilistic interpretation, a particle is described by a wavefunction $\Psi(x,t)$ whose squared modulus yields the probability density $P(x,t)$ of finding the particle at position x and time t. In addition, a group of particles is defined *entangled*, when the quantum state of each particle in the group cannot be described independently of the state of the others. This marks a defining departure from classical composition: the joint state of a composite quantum system generally cannot be factorised into independent subsystem states, giving rise to non-classical correlations that persist across large separations. In such ensembles, the state of each constituent is inseparable from that of the others, the system realises a single, global quantum state whose correlations endure even when the particles are far apart. Complementing this picture, the *no-cloning theorem* forbids the creation of an independent, identical copy of an arbitrary unknown quantum state, a limitation with far reaching consequences for quantum communication and computation.

2.1 Quantum computing: principles, promise and practical limits

Unlike conventional computers that process information as binary digits streams of electrical or optical signals representing logical 0s and 1s, quantum computers operate on *qubits*, physical systems such as electrons, photons or superconducting circuits maintained in precisely controlled quantum states. Realised through technologies like superconducting loops at cryogenic temperatures or ions confined in ultra-high-vacuum traps, qubits exhibit distinctly non-classical behaviour that can be harnessed for computation.

Quantum Computing (QC) exploits three quintessential quantum phenomena to manipulate information in ways that classical architectures cannot:

- **quantum superposition**: quantum bits can occupy coherent combinations of 0 and 1 at once, so ensembles of qubits span exponentially large state spaces that algorithms can explore during computation;
- **quantum entanglement**: in composite quantum systems, the joint state cannot be factored into independent states of the parts. As a result, measurements on one qubit constrain

the admissible states of others, even across large separations, yielding correlations with no classical analogue. This resource is engineered in quantum circuits to coordinate multi-qubit operations and enable algorithmic speedups.

- **quantum interference**: by engineering constructive and destructive interference of probability amplitudes, quantum circuits amplify the probability amplitudes associated with correct outcomes while suppressing the others, which yields speedups for specific problem families (e.g. structured search, factoring and selected simulation tasks).

QC is *not* a universal accelerator: performance gains are problem dependent and arise only when algorithms can exploit quantum structure. Moreover, contemporary devices are noise limited. *Decoherence*, the loss of phase information due to unwanted interaction with the environment, together with gate and readout errors, forces the use of fault tolerance, whereby a single logical qubit is encoded across many physical qubits. As a consequence, large-scale, general purpose quantum advantage remains a medium term objective, while near term utility is expected in domains such as quantum simulation (e.g. chemistry and materials) [2], constrained optimisation and selected data analysis workflows where hybrid quantum-classical pipelines are effective. Minimising decoherence is essential. Even tiny thermal or vibrational perturbations ("noise") can destroy phase coherence and eject qubits from superposition prematurely. Error rates further necessitate *fault tolerance*, in which one logical qubit is encoded across many physical qubits, with leading surface code schemes this overhead is typically in the hundreds-to-thousands of physical qubits per logical qubit. [1]

QC is *not* a general-purpose accelerator. Its benefits are problem dependent and arise only when algorithms can exploit specifically quantum structure. Beyond the gate-model paradigm, specialised approaches such as Quantum Annealing (QA), target combinatorial optimisation directly. Commercial QA systems (e.g. D-Wave's superconducting flux-qubit processors with $\mathcal{O}(10^3)$ physical qubits) implement adiabatic dynamics, but face practical scaling limits from cryogenic infrastructure to fabrication variability. Looking forward, roadmaps emphasise fault tolerance and scale: for example, plans call for systems with hundreds of *logical* qubits by the end of the decade and thousands in the early 2030s, together with advances in error correction aimed at pushing sustained circuit depths into the 10^8–10^9 gate range [13]. Alternative hardware lines (e.g. topological platforms based on Majorana modes) aim to reduce error rates at the device level, but remain in the research and prototyping stage [14].

Overall, current trends in QC point towards larger-scale devices, higher fidelities and progressively more expressive programming abstractions. In the near term, a form of *quantum utility* is beginning to materialise, in which today's noisy hardware can already deliver reliable solutions to selected problems that go beyond straightforward brute-force classical simulation, particularly in areas such as quantum simulation and constrained optimisation. In contrast, sustained, fault-tolerant *quantum advantage* is expected to depend on the next generation of fully error-corrected architectures that are presently under development.

2.2 The quantum threat to cryptography

Contemporary cryptographic systems, especially those relying on asymmetric primitives, face an inherent *expiration date* under the imminent arrival of large-scale quantum computers.

The *Quantum Threat Timeline Report 2024*, published by the Global Risk Institute, outlines projections for the first Cryptographically-Relevant Quantum Computer (CRQC), estimating that within the next 15–30 years such devices may be capable of breaking encryption schemes now widely deployed [15]. As the report emphasises, the threat is not just theoretical. Organisations must begin preparations today because:

- the development, validation, standardisation and wide deployment of post-quantum algorithms is a multi-year process;
- adversaries may already adopt a "Harvest Now, Decrypt Later (HNDL)" strategy, capturing encrypted data today in hopes of decrypting it when quantum resources become available;

- sensitive data collected now may remain valuable years into the future, making deferred decryption attacks especially damaging.

2.2.1 X-Keyscore: a real-world example of data harvesting

To illustrate the reality of large-scale data collection, we could consider the NSA's X-Keyscore surveillance system, revealed in disclosures by Edward Snowden, which enables broad interception and analysis of global Internet traffic, including emails, web browsing and metadata [16]. Although not a cryptanalysis tool itself, X-Keyscore exemplifies how intelligence frameworks already possess infrastructure capable of massive data harvesting, which could be repurposed in a post-quantum era to mount retrospective decryption attacks.

By integrating extensive data collection platforms with prospective quantum decryption capabilities, an adversary could utilise X-Keyscore like systems to maintain extensive archives of encrypted content, awaiting the moment when quantum advantage renders decryption feasible.

2.3 Quantum impact on current technologies

The advent of QC represents a paradigm shift for modern cryptography, undermining the mathematical assumptions upon which current security systems rely. Contemporary cryptographic mechanisms draw their strength from the computational infeasibility of solving certain mathematical problems using classical architectures. Asymmetric schemes such as RSA, DH key exchange, and ECC depend on the presumed hardness of integer factorisation and discrete logarithm problems, whereas symmetric ciphers and hash functions derive their robustness from the impracticality of exhaustive key search.

While these problems remain intractable under classical computational models, the emergence of large-scale quantum computers would overturn such guarantees. Quantum algorithms can, for specific classes of problems, provide exponential speed-ups over their classical counterparts, as exemplified by Shor's algorithm [4] for factorisation and discrete logarithms, while others, such as Grover's algorithm [17] for unstructured search, offer quadratic improvements. These advances render much of today's public-key cryptography vulnerable to future quantum adversaries.

2.3.1 Shor's algorithm

Shor's algorithm is known for its ability to endanger public-key cryptography, since it can efficiently factor large numbers and solve the Discrete Logarithm Problem (DLP), the foundation of security for many public-key algorithms [4].

The algorithm proceeds in two stages:

1. a classical reduction that maps the Factoring problem to an *Order-Finding* problem, lowering the complexity of the original task;
2. a quantum subroutine that solves the Order-Finding problem efficiently.

The *Order-Finding problem* consists in determining the period of a modular exponential function.

Formally, given the exponential function a^x the modular exponential function is defined as the remainder of the division among the exponential function and an integer N:

$$F_N(x) = a^x \bmod N.$$

The *order* r of the modular exponential is the least positive integer such that:

$$a^r \bmod N = 1.$$

The strategy over which the Order-Finding solution is based on consists in computing the function $F_N(x)$ for many values of x in parallel, aiming to detect the period in the function sequence's values. Using randomisation, integer factorisation can be reduced to finding the order of a suitably chosen element. A high-level description of the factoring procedure is:

1. pick a random integer x coprime to N;
2. compute the order r of x modulo N;
3. compute $\gcd(x^{r/2} - 1, N)$.

The quantum algorithm to compute the order can be found in [4]. The computed greatest common divisor fails to produce a non-trivial factor of N only in the exceptional cases

$$r \bmod 2 = 0 \qquad \text{or} \qquad x^{r/2} \equiv -1 \pmod{N}.$$

When x is chosen uniformly at random, the procedure yields a non-trivial factor of N with probability at least

$$p = 1 - 2^{1-k},$$

where k denotes the number of distinct odd prime factors of N.

The asymptotic time complexity of Shor's algorithm is commonly quoted as

$$O\big((\log N)^3\big) = O(n^3),$$

where N is the integer to factor and n is the bit-length of N. Consequently, a sufficiently large and capable quantum computer could efficiently solve integer factorisation and the DLP, thus compromising systems that depend on these problems.

It is important to emphasise that while practical large-number factorisations via real quantum hardware are not yet available, proof-of-principle demonstrations and small-number implementations have been reported [18]. Hence, Shor's algorithm is both theoretically and experimentally established, even though the construction of a large-scale machine capable of factoring cryptographically relevant integers remains an open engineering challenge.

2.3.2 Grover's algorithm

In contrast, *Grover's algorithm*, threatens symmetric key primitives, or, more generally, provides a speed up for searching an unstructured database with respect to the classical algorithms. Informally, the task that the algorithm aims to solve can be expressed as follows.

Given an abstract function $f(x)$ that accepts search items x, an item x_0 is a solution to the search task if

$$f(x_0) = 1$$

otherwise,

$$f(x_0) = 0.$$

The *Search Problem* consists in finding any item such that $f(x_0) = 1$.

Based on this problem, the purpose of the algorithm can be seen as inverting a "black-box" function $y = f(x)$ by amplifying the amplitude of the marked solutions across the search space. Hence, the algorithm enables the search for specific solutions across all possible input combinations.

As stated by the European Telecommunications Standards Institute (ETSI) [19], if a problem has the following four properties:

- the only way to solve the problem is by repeatedly answering and checking the answers;
- the number of candidates equals the number of possible inputs;
- each candidate evaluation requires comparable time (uniform cost);
- there is no additional structure or heuristic that favours certain candidates over others.

The time required for a quantum computer to solve problems with these four properties is proportional to the square root of the number of inputs.

Classically, finding a marked item in the worst case requires $O(N)$ oracle queries, and so $f(x)$ has to be evaluated a total of $N-1$ times for a space of size N (e.g. in the symmetric algorithms and hash functions contexts, this can be seen as all the possibilities that must be evaluated to respectively brute force the secret key or a pre-image). This is reduced to $O(\sqrt{N})$ queries using Grover's technique [17], which offers a quadratic speedup for brute force search jobs.

2.3.3 Cryptographic schemes affected

Asymmetric schemes

Any cryptosystem, security protocol, and product relying on the presumed classical hardness of integer factorisation or discrete logarithms becomes vulnerable in the presence of a sufficiently powerful quantum computer. This includes RSA, DH , and ECC [20].

For RSA, the knowledge of the public key (N, e) allows an attacker that can factor N to compute the private exponent d, satisfying:

$$e \cdot d \equiv 1 \pmod{\varphi(N)}$$

where

$$\varphi(N) = (p-1)(q-1)$$
$$N = pq$$

Once N is factored into primes, it is straightforward to compute the private key. Furthermore, publishing the public key would be equivalent to posting the private key as well.

Not only would all data encrypted with this method be vulnerable, but no message could be guaranteed to be secure, effectively destroying the purposes of the encryption and digital signatures.

Unlike RSA, DH and ECC based schemes rest on the Discrete Logarithm Problem hardness. *Diffie-Hellman* is an asymmetric cipher widely deployed that uses the aforementioned properties to transmit keys securely over a public network. Rather than exploiting the classic DLP, algorithms that employ Elliptic Curve Cryptography (e.g. ECDH) rely on the hardness of computing the Elliptic Curve Discrete Logarithm for their security.

The difficulty of breaking these cryptosystems is based on the difficulty in determining the integer r such that:

$$g^r \equiv x \pmod{p}$$

which can be expressed as:

$$r \equiv \log_g x \pmod{p}$$

The integer r is called the DLP of x to the base g.

A suitable adaptation of Shor's algorithm can efficiently solve Discrete Logarithms over these groups (and over Elliptic Curves) [21]. Due to ECC employing shorter keys for identical classical security, smaller quantum computers may potentially compromise ECC prior to breaching comparable RSA key sizes.

Typical deployments that would be affected include:

- Public Key Infrastructure (PKI);
- Secure Software Distribution;
- Key Exchange over Public Channels;
- Virtual Private Networks (VPNs);
- Secure Web Browsing (e.g. SSL/TLS);
- Secure Boot.

Symmetric schemes

For symmetric cryptography, the advent of quantum computing is regarded as a comparatively minor threat [20]. This is because the fundamental primitives on which these algorithms rely are not based on computational problems that can be efficiently solved by Shor's algorithm. The only known quantum threat in this domain arises from *Grover's algorithm*, which allows to obtain a quadratic speed-up over classical exhaustive methods.

Consequently, a quantum computer would not dramatically reduce the time required to discover a symmetric key compared to classical machines. The same level of security can therefore be maintained by simply doubling the key length, assuming that the symmetric algorithm does not depend on structural properties that can be exploited by quantum computation.

Formally, for a cipher employing an n-bit key, a quantum computer would need approximately $\sqrt{2^n} = 2^{n/2}$ evaluations to complete an exhaustive key search.

Accordingly, Advanced Encryption Standard (AES) remains a highly resilient symmetric primitive in the quantum era when deployed with keys of at least 192 or 256 bits, which translate to approximate security levels of 96 and 128 bits, respectively.

Moreover, Grover's algorithm is known to parallelise poorly. As reported in [22], even when multiple quantum computers operate in parallel, the improvement over the classical brute force approach remains marginal.

Table 2.1 summarises the classical and quantum security strengths of several widely adopted cryptographic schemes in light of the discussed quantum algorithms [23].

Table 2.1. Comparison of classical and quantum security levels for the most used cryptographic schemes

Crypto scheme	Key size	Classic strength (bits)	Quantum strength (bits)
RSA-1024	1024	80	0
RSA-2048	2048	112	0
ECC-256	256	128	0
ECC-384	384	256	0
AES-128	128	128	64
AES-256	256	256	128

Hash functions

The family of hash functions is subject to threats analogous to those faced by symmetric ciphers. Adversaries typically pursue two primary goals: finding a *collision*, i.e. two distinct inputs that yield the same digest, or discovering a *pre-image*, i.e. an input that maps to a given hash output.

As previously discussed, Grover's algorithm effectively reduces the computational cost of a pre-image search: a quantum adversary can identify a pre-image by exploring approximately the square root of the classical search space. Consequently, an n-bit hash function offers only about $n/2$ bits of resistance against an adversary equipped with an ideal quantum search capability.

The hash output length must thus be doubled in order to preserve the same degree of security as in the classical context.

Collision attacks follow a different complexity model. Classically, the Birthday Paradox [24] implies that a collision on an x-bit hash can be expected after about $2^{x/2}$ evaluations. Brassard observed that Grover style quantum techniques can be combined and adapted to attack collisions more efficiently than the naive birthday strategy for certain black-box functions. His approach targets an r-to-one function F, i.e. a function for which each output has on average r pre-images, and trades quantum memory for time. Concretely, for collision search the two algorithms compare as follows:

$$\begin{aligned} \text{Grover}: &\quad \text{time } = O(N^{1/2}),\ \text{space } = O(\log N), \\ \text{Brassard}: &\quad \text{time } = O\big((N/r)^{1/3}\big),\ \text{space } = O(N^{1/3}) \end{aligned}$$

where N denotes the size of the domain being searched. For cryptographic hash functions (modelled as near random mappings), the Brassard-style method therefore achieves a better time/space trade-off than Grover alone by allocating substantial quantum memory. Informally, if a quantum adversary can maintain a lookup structure of size $\Theta(N^{1/3})$ and use Grover as a subroutine, collision finding can be carried out in roughly $O(N^{1/3})$ time and space rather than $O(N^{1/2})$ time with only logarithmic space.

Translating these discoveries into practical hash length standards creates a prudent basis for post-quantum collision-resistant design. Upon demonstrating that, the Brassard technique reduces the effective collision search cost from $2^{x/2}$ to roughly $2^{x/3}$ within the presumed quantum memory framework. As a result, a commonly accepted conservative standard is to allocate hash outputs of no less than $3b$ bits to guarantee b-bit collision resistance against adversaries who can leverage these quantum advantages.

Taken together with the Grover based pre-image considerations, these analyses explain why many legacy hash constructions are unsuitable for a post-quantum environment, and why SHA-2/SHA-3 variants with substantially longer outputs remain preferable for quantum-resilient deployments.

The overall quantum impact on various widely adopted cryptographic algorithms is summarised in table 2.2.

Table 2.2. Summary of the quantum impact on different cryptographic algorithms

Cryptographic algorithm	Type	Purpose	Quantum impact
AES	Symmetric key	Encryption	Larger keys needed
SHA-2, SHA-3	—	Hash functions	Larger output needed
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH	Public key	Signatures, key exchange	No longer secure
DSA	Public key	Signatures, key exchange	No longer secure

2.4 Migration to a quantum-safe state

2.4.1 Migration challenges

According to the NIST, the migration to PQC cannot be accomplished as a simple drop-in replacement [25].

The diversity of contexts in which this transition must occur, ranging from Internet of Things (IoT) devices and embedded systems, to complex applications and network protocols, introduces significant challenges. Indeed, PQC algorithms differ substantially from current cryptographic standards in terms of key sizes, ciphertext and signature lengths, as well as computational and communication requirements, many of which may be incompatible with existing infrastructures. In addition, several candidates introduce novel requirements, such as state management, necessitating modifications to current frameworks. These algorithms may be suitable for certain

environments but misaligned with others. Furthermore, each new cryptographic scheme proposed for standardisation requires extensive cryptanalysis, as mandated by the ongoing NIST evaluation process. Consequently, it is crucial to recognise that no single algorithmic replacement will be sufficient. Rather, a diverse set of quantum-resistant solutions will need to coexist, each selected according to specific operational requirements and deployment contexts.

The migration process must therefore be approached holistically, taking into account three key dimensions: performance, security and implementation.

Performance considerations

Since PQC algorithms typically demand greater computational power, memory, storage and communication resources, assessing their performance in various deployment scenarios represents a critical research area.

Before these schemes can be safely adopted in production environments, extensive studies are required to develop efficient and practical implementations. For instance, in networking environments, the increased key and signature sizes associated with PQC schemes can introduce additional latency and bandwidth overhead in secure communication protocols such as TLS and during IPsec key establishment, thereby directly affecting overall system performance. Optimising these parameters while preserving scalability and security remains a major engineering challenge. Similarly, in constrained environments such as IoT devices, limitations on processing, memory and battery capacity further complicate deployment. Until lightweight and hardware-efficient implementations become available, the widespread adoption of PQC will remain limited in such constrained environments, as well as across legacy systems.

Security considerations

The integration of new public-key primitives has direct implications not only for performance but also for overall system security. Unlike RSA or ECC, PQC schemes involve different trade-offs among configurable parameters, such as key length, ciphertext size and computation time. A key challenge, therefore, lies in balancing these factors across diverse operational domains, as inappropriate configurations may inadvertently expand the attack surface.

Moreover, many PQC candidates, such as Multivariate Cryptographic Schemes, are still undergoing intensive cryptanalysis compared to the long studied classical algorithms. A particularly important research direction concerns the identification of vulnerabilities in different protocols and adversarial models. Among these, *side-channel vulnerabilities* represent a critical concern. Side-channel attacks exploit information leakage from an implementation, such as timing behaviour, power consumption or memory access patterns, to infer sensitive data. Since most PQC algorithms introduce novel computation and communication patterns, understanding and mitigating such leaks is a central open problem. Accordingly, designing implementations resilient to this type of attack exploitation remains an active and essential area of research.

Implementation considerations

The implementation of cryptographic algorithms is itself a cornerstone of *secure system design*. Even mathematically sound algorithms can become insecure if implemented incorrectly. This challenge extends beyond simple software coding, it also involves translating abstract mathematical formulations into platform specific architectures. Subtle factors like data representation, memory layout, and buffer handling can introduce security flaws that are difficult to detect.

As previously stated, problems are more noticeable in embedded systems, where implementation complexity is increased by heterogeneity in device capabilities (e.g. memory, power availability, processor architecture). Security risks are further increased by the fact that these devices are frequently physically accessible and so susceptible to tampering.

Finally, it should be emphasised that the reference implementations bundled with NIST candidate submissions are not intended for direct production deployment. Algorithms that have already been standardised are a partial exception. However, they should still be adopted prudently, given the comparatively limited public cryptanalysis to date. Nonetheless, initiating a phased migration remains essential. Therefore, practitioners are encouraged to rely on well-maintained, security-hardened libraries, such as the *liboqs* library from the Open Quantum Safe project [26], to facilitate experimentation and integration of PQC algorithms across specific platforms.

2.4.2 Cryptographic agility

As previously stated, continual advances in cryptanalysis and computing steadily erode the practical security of deployed primitives, so schemes once regarded as robust may become inadequate.

New cryptographic primitives and schemes may also be required to mitigate emerging weaknesses or to achieve more favourable trade-offs between assurance and performance. It is therefore sensible to treat obsolescence as an intrinsic stage in the cryptographic life cycle and to engineer systems that explicitly anticipate, and can gracefully accommodate, such change.

In this spirit, *cryptographic agility* denotes the capability to replace and adapt cryptographic mechanisms across software, hardware, firmware, protocols and supporting infrastructures, while preserving security, interoperability and service continuity [27].

Recent guidance from NIST frames crypto-agility as a cross-cutting set of practices spanning: protocol design, application interfaces and enterprise governance. Intended to enable timely transitions without unacceptable operational disruption [28]. This perspective is elaborated in the NIST Cybersecurity White Paper on Crypto Agility, which surveys historic transitions, identifies common pitfalls, outlines strategies for protocol negotiation, key life-cycle management and enterprise risk governance.

Crypto-agility concepts are based on the following pillars. Robust software engineering practices encapsulate cryptographic functionality behind stable, policy-driven interfaces, thereby allowing algorithms and parameter sets to be substituted without necessitating modifications to data paths or business logic. Modular architectures and well specified negotiation mechanisms enable implementations to introduce new cipher suites, including post-quantum options and hybrid compositions, while preserving interoperability. It is essential, however, that negotiation processes are integrity protected to prevent downgrade attacks and that deployments incorporate telemetry mechanisms to verify successful migration to the preferred suites. Moreover, operational governance, encompassing asset inventories, configuration baselines, deprecation strategies, conformance testing and rollback procedures, constitutes the organisational framework necessary to manage cryptographic transitions at scale and to ensure alignment with evolving standards and regulatory requirements [28].

2.4.3 Hybrid schemes

The Internet Engineering Task Force (IETF) highlights in [29] the necessity of employing *hybrid cryptographic schemes*, where traditional and post-quantum algorithms are combined following defined structural patterns. The rationale behind such combinations lies in the need, or in some cases, the regulatory requirement for security protocols to leverage both classes of algorithms simultaneously. This approach provides resistance against quantum adversaries while preserving the well understood security assurances of traditional cryptographic primitives. Moreover, hybrid deployments facilitate the practical integration of PQC into existing infrastructures, supporting a smoother migration process.

A Post-Quantum Traditional Hybrid (PQ/T) scheme can be viewed as a *Multi Algorithm Cryptographic Scheme* in which at least one component is quantum-resistant and at least one remains purely classical. More generally, a *Multi Algorithm Scheme* refers to any cryptographic construction that combines multiple algorithms serving the same purpose, such as key establishment or digital signatures. As a result, the security guarantees of a hybrid scheme are intrinsically tied to the security properties of its individual constituent algorithms.

Hybrid approaches acquire particular significance in the context of *retrospective decryption*, also known as the Harvest Now, Decrypt Later attack model. The adoption of PQ/T hybrid schemes thus provides an immediate defensive measure against such future decryption attempts.

As delineated in the aforementioned [29], the IETF further discusses how PQC certificates should be handled within transitional deployments that enable hybrid configurations. In particular, hybrid certificates are defined as those containing public keys from two or more component algorithms, at least one traditional and one post-quantum. These keys can either be embedded as a *composite public key* or provided as distinct, individually signed public key fields. Within this framework, a *Composite Cryptographic Element* is defined as an element that incorporates multiple *cryptographic elements* of the same type, collectively forming a multi algorithm construct.

Although distinct certificates could theoretically be maintained for each component algorithm, employing a single hybrid certificate can streamline hybrid authentication protocols. However, as emphasised by the IETF, the use of PQ/T hybrid certificates does not automatically guarantee *hybrid authentication of identity*. For example, an end-entity certificate containing a composite public key but signed using a single traditional algorithm may provide hybrid authentication of the message origin, yet it would not constitute hybrid authentication of the sender's identity.

A recent extension of these concepts is provided in the Internet-Draft *"Composite ML-DSA for use in X.509 Public Key Infrastructure"* [30], which defines hybrid combinations of the post-quantum `ML-DSA` algorithm [31] with traditional signature schemes such as `RSASSA-PSS`, `RSASSA-PKCS1-v1.5`, `ECDSA`, `Ed25519`, and `Ed448`. These hybrid constructs are designed for applications that rely on `X.509` or `PKIX` data structures, offering additional resilience against potential cryptanalytic or implementation level weaknesses in either algorithmic family while ensuring compliance with regulatory and assurance requirements.

2.5 Post-Quantum Cryptography

2.5.1 Post-Quantum families

Post-Quantum Cryptography represents the branch of cryptography designed to remain secure against both classical and quantum adversaries. Unlike traditional public-key systems, whose security relies on the previously discussed mathematical problems, PQC algorithms are based on mathematical principles considered resilient to quantum assaults. These constructions are generally classified into five main families of primitives, all ultimately derived from the concept of *trapdoor functions*.

A trapdoor function is a mathematical function that is computationally straightforward to evaluate in one direction, yet infeasible to invert without privileged information. The following sections summarise the principal families of PQC algorithms that have been selected for standardisation and still undergoing active evaluation and refinement.

Lattice-based cryptography

Lattice-based cryptography is a form of public-key cryptography that mitigates the structural weaknesses of RSA by basing its security on the presumed hardness of lattice problems.

A *lattice* may be described as a discrete, regularly repeating set of points in an n-dimensional Euclidean space, formed by all integer linear combinations of a chosen basis of vectors. In contrast to factorisation-based schemes, lattice-based cryptography replaces arithmetic over integers with operations on matrices and linear algebraic structures.

The security of such schemes derives primarily from the computational difficulty of the *Shortest Vector Problem* (SVP), which requires identifying the shortest non-zero vector in a lattice given an arbitrary basis, a task proven to be NP-hard. More formally, lattice-based cryptosystems rely on the *worst-case to average-case reduction* principle: breaking the scheme in practice would imply the existence of an algorithm capable of solving the corresponding lattice problem in all

instances. In most cases, this involves approximating hard lattice problems, such as SVP, within polynomial factors, a task widely conjectured to be intractable [32]. Consequently, the security of every key within these schemes is as strong in the easiest case as in the hardest.

Moreover, these schemes are typically computationally efficient and offer considerable flexibility, as their parameters can be adjusted to suit a wide range of deployment scenarios. At the same time, poor parameter choices can unintentionally erode security guarantees and broaden the attack surface. Additionally, to note is that, there is currently no known quantum algorithm that yields a substantial advantage over classical methods for the underlying lattice problems, which further strengthens their candidacy as post-quantum primitives.

Code-based cryptography

Code-based cryptography is grounded in the theory of error-correcting codes [33]. Such codes enhance reliability by embedding redundancy into transmitted data, enabling recovery even when bit errors occur on the channel. Within this family, *Goppa codes* have been studied extensively and underpin a number of robust constructions.

A secure scheme built on Goppa codes requires the secrecy of both the encoding and decoding transformations, while exposing only a *masked* public encoding function.

This public function enables the linkage of a message to a specific set of codewords, concealing the underlying framework of the secret code. While access to the secret decoding function is required in order to recover the original plaintext from the encoded communication.

In this category of cryptosystems, the ciphertext is depicted as a codeword intentionally combined with controlled errors, which can only be rectified with the possession of private decoding information.

Despite their theoretical strength, code-based systems suffer from extremely large key sizes, which obstruct their practical implementation. Although they frequently produce compact signatures, their overall efficacy is typically inferior to that of other post-quantum primitives.

Multivariate polynomial cryptography

Multivariate cryptography derives its hardness from solving systems of multivariate quadratic equations over finite fields [34].

In this setting, the trapdoor takes the form of a quadratic polynomial mapping that converts an otherwise linear system into a non-linear one. To achieve security, designers conceal specific algebraic structure within the polynomial system such that, to an external observer, the resulting equations are computationally indistinguishable from random instances.

While these approaches can produce compact signatures, they typically entail slow decryption and very large public keys, reflecting inherent inefficiencies in the decryption process.

Hash-based signatures

Hash-based cryptography offers digital signature schemes constructed entirely from cryptographic hash functions [35]. Because hash functions are well understood and not dependent on number theoretic assumptions, such constructions remain robust against both classical and quantum adversaries. Notably, even quantum computers cannot drastically improve upon the generic attack complexity against secure hash functions.

A primary constraint of hash-based signatures is their frequent dependence on *One-Time Signature* (OTS) schemes, wherein each key pair can be utilised to sign only a singular message. The reuse of keys undermines security by disclosing adequate information for an attacker to replicate signatures.

To overcome this limitation, modern constructions arrange one-time keys in a binary tree, allowing a single master key pair to support many signatures, with the total capacity determined by the depth of the tree.

In these tree-based constructions, the value stored at each internal node is obtained by hashing its two child nodes, while the hash at the root serves as the public key. For each individual signature, a distinct OTS key is chosen from the leaves, and the final signature comprises both the OTS signature output and an authentication path that links the selected leaf back to the root. Schemes of this kind are usually categorised as either **stateful** or **stateless**. In the stateful case, the signer must maintain accurate records of key usage to avoid reusing one-time keys, which can become operationally demanding at scale. Stateless schemes remove this bookkeeping requirement, but do so at the price of significantly larger signatures.

Isogeny-based cryptography

Isogeny-based cryptography is among the most mathematically intricate branches of post-quantum cryptography. Its security is grounded in the hardness of problems involving *isogenies*, that is, structure-preserving morphisms between elliptic curves.

Central to this area is the *Supersingular Isogeny Problem*, which, given two supersingular elliptic curves, asks for an isogeny that maps one onto the other. Current evidence suggests that this problem is computationally infeasible for both classical and known quantum algorithms, as no efficient sub-exponential quantum algorithm is available to solve it [36]. Consequently, isogeny-based schemes offer one of the few post-quantum options that retain key and ciphertext sizes on a similar scale to traditional elliptic-curve cryptography, making them particularly appealing in bandwidth-constrained environments.

2.5.2 NIST selected algorithms

This section provides a general overview of the cryptographic objects considered within the NIST *Post-Quantum Cryptography* standardisation process.

The PQC algorithms target three principal applications: public-key encryption, Key Encapsulation Mechanisms (KEMs) and digital signatures.

A *Public Key Encryption* (PKE) scheme consists of three core algorithms:

- a *key generation* algorithm that outputs a pair of public and private keys;
- an *encryption* algorithm that, given a message and a public key, produces a ciphertext;
- a *decryption* algorithm that, given the ciphertext and the corresponding private key, recovers the original plaintext.

For correctness, the decryption algorithm must always reproduce the original message when the appropriate private key is used.

A KEM provides a mechanism for two parties to establish a shared symmetric key over an untrusted communication channel. It operates by encapsulating a session key into a fixed-size ciphertext, from which the corresponding symmetric encryption material can subsequently be recovered.

Formally, a KEM consists of three algorithms:

- a *key generation* algorithm producing a pair of public and private keys;
- an *encapsulation* algorithm that uses the public key to generate a ciphertext and a symmetric key;
- a *decapsulation* algorithm that, given the ciphertext and the private key, recovers the same symmetric key.

Correctness requires that encapsulation and decapsulation yield identical session keys when the legitimate private key is applied.

A DSS also relies on three algorithms:

- *key generation*, which produces a public-private key pair;
- *signature generation*, which takes a message and a private key to compute a digital signature;
- *verification*, which takes a signature, a message, and a public key, and outputs a boolean value indicating validity.

For correctness, a correctly generated signature verifies under the right public key.

Following three evaluation rounds (2022), NIST announced the first set of algorithms to be standardised:

- **PKE/KEM:** CRYSTALS-Kyber;
- **DSS:** CRYSTALS-Dilithium, Falcon and SPHINCS+.

Beyond these, the benefits of Extended Merkle Signature Scheme (XMSS) need consideration: its hash-based design is increasingly regarded as both robust and conceptually straightforward for quantum-safe use.

In parallel with its primary selections, NIST has also retained a set of *alternate candidates* that exhibit promising security properties but warrant further investigation, particularly in terms of performance and implementation. These have progressed to a *fourth evaluation round* for continued assessment.

Acceptance Criteria

The NIST call for proposals required submissions to meet specific security guarantees. For PKE and KEM schemes, the main criterion was ***Indistinguishability under Adaptive Chosen Ciphertext Attack* (IND-CCA2)**, ensuring that adversaries cannot distinguish ciphertexts even after adaptively querying a decryption oracle. This property maintains confidentiality under chosen-ciphertext attacks, preventing adversaries from generating new valid ciphertexts.

For DSS, NIST mandates ***Existential Unforgeability under Adaptive Chosen Message Attack* (EUF-CMA)** security. This ensures that no adversary, even one with access to a signing oracle, can forge a valid signature on any previously unseen message.

Evaluation Criteria

Security strengths in the NIST PQC process are defined in terms of *security levels*, a concept derived from NIST Special Publication 800–57 [23]. Unlike classical schemes, where resistance could be conveniently expressed in terms of key length, the advent of quantum attacks requires an abstract quantification based on equivalent computational effort. Each level corresponds to the estimated cost of performing a specific reference attack against well understood symmetric primitives such as AES or SHA–2/3, both under classical and quantum cost models. The five levels adopted by NIST are summarised in Table 2.3, together with the reference hardness assumptions and motivating attack types (Grover's and Brassard's algorithms).

These levels are used as a baseline for categorising post-quantum schemes to ensure that their security strength remains consistent with classical expectations even under the quantum threat model. Level I, for example, corresponds to the effort required to perform a Grover optimised key search over AES–128, while Level V represents parity with the full exhaustive search over AES–256.

Performance was the second key metric, incorporating computational efficiency, communication overhead and implementation costs. This includes the time for key generation, encryption or signing operations, the bandwidth required to transmit public keys and ciphertexts or signatures.

Nevertheless, it is worth noting that, for KEMs, where ephemeral key pairs are often generated per session to ensure forward secrecy, generation cost is more relevant than in signature schemes.

Table 2.3. NIST PQC Security Levels - equivalence with classical symmetric primitives and quantum attack complexity.

Level	At least as hard	Attack model	Quantum complexity
I	AES–128	exhaustive key search	$O(2^{64})$ (Grover)
II	SHA–256	collision search	$\sim O(2^{85})$ (Brassard)
III	AES–192	exhaustive key search	$O(2^{96})$ (Grover)
IV	SHA–384	collision search	$\sim O(2^{128})$ (Brassard)
V	AES–256	exhaustive key search	$O(2^{128})$ (Grover)

NIST Standardised Algorithms

On August 13, 2024, NIST released the first three standards resulting from the *Post-Quantum Cryptography* standardisation process:

- **ML-KEM (FIPS 203)** [37] – *Module-Lattice-based Key Encapsulation Mechanism (ML-KEM)* (formerly *CRYSTALS-Kyber*);
- **ML-DSA (FIPS 204)** [38] – *Module-Lattice-based Digital Signature Algorithm (ML-DSA)* (formerly *CRYSTALS-Dilithium*);
- **SLH-DSA (FIPS 205)** [39] – *Stateless Hash-based Digital Signature Algorithm (SLH-DSA)* (formerly *SPHINCS+*).

Module-Lattice-based Key Encapsulation Mechanism (ML-KEM)

ML-KEM is an IND-CCA2 secure KEM, whose security foundations lie in well-studied problems from lattice-based cryptography. It demonstrates excellent performance across software, hardware, and hybrid environments, achieving efficient key generation, encapsulation and decapsulation. Like other structured lattice constructions, its public key and ciphertext sizes are on the order of a few kilobytes.

Table 2.4 summarises key and ciphertext sizes (in bytes) compared with traditional key exchange schemes for a 256 bits (i.e. 32 bytes) encapsulated key.

Table 2.4. Comparison among ML-KEM and traditional KEM/KEX schemes: public key, private key and ciphertext sizes (in bytes).

Security Level	Algorithm	Public key size	Ciphertext size
Traditional	P256_HKDF_SHA256	65	65
Traditional	P521_HKDF_SHA512	133	133
Traditional	X25519_HKDF_SHA256	32	32
1	ML-KEM–512	800	768
3	ML-KEM–768	1184	1088
5	ML-KEM–1024	1568	1588

The three configurations correspond to the following quantum-resistant security levels:

- ML-KEM-512: equivalent to AES-128 security;

- ML-KEM-768: equivalent to AES-192 security;
- ML-KEM-1024: equivalent to AES-256 security.

Considering the expected capabilities of quantum adversaries (e.g. Grover's algorithm), ML-KEM-768 is generally recommended as it achieves a balanced trade-off between classical and post-quantum security.

Digital Signature Schemes

The standardised digital signature algorithms are *ML-DSA* and *SLH-DSA*. Although *Falcon* was among the NIST Round 3 finalists, it was not ultimately selected for standardisation, as discussed in Section 2.5.2.

All mentioned standardised schemes comply with the EUF-CMA security criterion and utilise the *hash and sign* methodology, whereby messages are hashed prior to the generation of signatures.

Module-Lattice-based Digital Signature Algorithm (ML-DSA)

ML-DSA is a lattice-based signature scheme proven secure under chosen-message attacks, relying on the computational hardness of module-lattice problems. This security notion means that an adversary having access to a signing oracle cannot produce a signature of a message whose signature he has not seen yet, nor produce a different signature of a message that he already saw signed.

It offers flexible parameter sets allowing trade-offs between key size, signature size and computational performance. Together with Falcon, ML-DSA was among the most efficient Round 3 candidates suitable for widespread deployment, with even better performance than classic ones.

Stateless Hash-based Digital Signature Algorithm (SLH-DSA)

SLH-DSA is a stateless hash-based signature scheme. It combines one-time signatures, few-time signatures and tree structures to build a general purpose digital signature framework.

Specifically, it employs both:

- a few-time signature scheme, Forest of Random Subsets (FORS);
- a multi-time signature scheme, the Extended Merkle Signature Scheme (XMSS), constructed using the hash-based one-time signature scheme Winternitz One-Time (WOTS+)

Its security of SLH-DSA rests solely on the properties of the underlying hash function. Because of the way signatures are constructed, key generation and verification are considerably faster than signing. Internally, the scheme uses randomised message compression via a keyed hash function capable of processing inputs of arbitrary length.

Two alternative design variants are proposed:

- a faster signature mode at the expense of larger signatures;
- a smaller signature mode at the cost of slower signing.

More aggressive trade-offs may also be selected depending on application requirements.

Specifically, the public keys are very short, but the signatures are relatively large. One significant drawback lies in the complexity of the scheme: its sophisticated structure makes implementation error-prone, complicating both security analysis and deployment. It is essential to acknowledge that the algorithm's design imposes a limit on the number of signatures: beyond a certain number, there is a non-negligible probability that the signatures leak enough information

to allow forgery. To keep this risk acceptably low, the total number of signatures that can be securely generated must be bounded (for instance, NIST requires support for at least 2^{64} signatures) [40].

In table 2.5 the key and signature sizes for SLH-DSA variants are shown alongside those of other PQC and traditional algorithms (e.g. RSA-2048, ECC P-256). The entries for SLH-DSA include both "s" (small) and "f" (fast) versions, denoting trade-off options.

Table 2.5. Comparison among PQC DSS and traditional counterparts (in bytes)

Sec. Level	Algorithm	Pub. key size	Priv. key size	Sign. size
Traditional	RSA-2048	256	256	256
Traditional	P-256	64	32	64
1	SLH-DSA-SHA256-128f	32	64	17088
2	ML-DSA-44	1312	2560	2420
3	ML-DSA-65	1952	4032	3309
3	SLH-DSA-SHA256-192f	48	96	35664
5	SLH-DSA-SHA256-256f	64	128	49856
5	ML-DSA-87	4627	4896	4627

Extended Merkle Signature Scheme (XMSS)

XMSS is a stateful hash based digital signature scheme capable of producing a fixed, though potentially large, number of signatures per key pair [40]. Its security relies solely on the properties of cryptographic hash functions rather than number-theoretic assumptions. While state tracking is required, XMSS offers compact implementations, natural side-channel resistance and relatively short signatures compared with other hash-based constructions. The multi tree variant of XMSS mitigates key generation latency and enhances scalability, making it a practical option for constrained or embedded environments.

Implementation flaws in post-quantum algorithms

The evolution of the PQC standardisation process illustrates that algorithmic innovation and solid implementation frequently conflict. Certain schemes once considered promising have revealed critical design or operational weaknesses under scrutiny, prompting withdrawals or delayed standardisation.

The fall of SIKE and the Hertzbleed attack Schemes in the *isogeny-based* family, such as Supersingular Isogeny Key Exchange (SIKE), provide a DH style key agreement on the isogeny graph of elliptic curves. Rather than exponentiation in finite fields or scalar multiplication in Elliptic Curve Diffie-Hellman (ECDH), each party computes a secret isogeny whose kernel is generated by a private point, publishes the resulting isogenous curve together with the images of basis points, and both parties arrive at the same j-invariant to derive a shared key.

Initially celebrated for their compact key sizes and elegant mathematical foundations, these constructions were seen as potential candidates for long term quantum resistance.

However, SIKE, jointly implemented by Microsoft and Cloudflare as a "constant time" library to resist timing attacks, was later found vulnerable to a new class of side-channel exploit known as *Hertzbleed* [41]. This attack leverages dynamic CPU frequency scaling (e.g. Intel Turbo Boost, AMD Ryzen Zen 2 and Zen 3) to extract cryptographic keys remotely by analysing variations in execution frequency rather than execution time.

In addition to practical implementation difficulties, cryptanalytic attacks that exploited structural weaknesses in SIDH ultimately prompted the authors to withdraw the SIKE proposal from the NIST PQC competition. The research team documented both their findings and the precise cause of the scheme's failure in a final technical report [42], an act that has since been recognised as contributing to greater collective insight and transparency within the PQC community.

The Falcon problem Falcon, a lattice based signature scheme and former finalist in the NIST competition, achieves compact keys and efficient verification through the use of Fast Fourier Transform (FFT) arithmetic over NTRU lattices. Despite its performance advantages, Falcon's reliance on floating-point operations poses significant implementation risks: non-constant time arithmetic introduces potential side-channel leakage. Consequently, NIST has delayed the publication of *FIPS 206*, the intended Falcon based digital signature standard (provisionally titled *FN-DSA*, short for FFT aver NTRU-Lattice-based Digital Signature Algorithm), until secure and deterministic constant-time mitigations are demonstrated.

Alternative KEMs: BIKE and HQC Among code-based KEM proposals, Bit-flipping Key Encapsulation (BIKE) and Hamming Quasi-Cyclic (HQC) have advanced as alternative candidates to the lattice based ML-KEM. Both are built upon error correcting codes, but rely on distinct mathematical assumptions. BIKE is grounded in Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes and offers simple, low latency operations with compact public keys. However, its security relies heavily on the hardness of decoding random linear codes, an assumption that has been under continual examination.

By contrast, HQC employs structured quasi-cyclic codes and a noise injection mechanism that enhances resistance to known decoding and reaction attacks. It provides a conservative design with predictable performance and resistance to side-channel analysis. For these reasons, NIST has initiated its standardisation as *FIPS 207* [43], presenting HQC as a mathematically diverse and implementation friendly complement to ML-KEM.

Chapter 3

Internet Protocol Security (IPsec)

This section provides an overview of one of the oldest security protocols [44], which emerged from the necessity to protect network infrastructure from unauthorised monitoring and control of the network traffic, as well as to secure end-user-to-end-user communication through authentication and encryption mechanisms.

The analysis begins with a review of the current IPsec specifications, RFC-4301 and RFC-7296 respectively for the security architecture and for Internet Key Exchange protocol, providing an overview of its core mechanisms, architectural principles and the evolution from its earlier versions. Particular attention is given to the updates introduced in subsequent specifications, namely RFC-9242 and RFC-9370, which refine key exchange procedures and cryptographic negotiation to accommodate post-quantum requirements, reflecting the primary efforts to solve practical difficulties such as increasing signature, certificate volumes and network packet overhead.

3.1 IPsec architecture

IPsec is the IETF architecture for Layer 3 security in IPv4/IPv6 designed to create a Secure VPN (S-VPNs) over unstrusted networks and create end-to-end secure packets flows. Generally speaking, an S-VPN can be defined as a solution that enhances the security and privacy of user communications by protecting network packets through cryptographic means. When robust cryptographic algorithms are employed, the only remaining practical attack vector is to disrupt the communication channel itself (Denial of Service (DoS)). Prior to encapsulation, packets are safeguarded using the following mechanisms:

- **Message Authentication Code (MAC):** ensures data integrity and source authentication;
- **encryption:** provides confidentiality by protecting the payload against unauthorised access;
- **sequence numbering:** mitigates replay attacks by enforcing packet order validation.

This is achieved through the definition of two specific packet types:

- **AH:** provides integrity, authenticity and protection against replay attacks;
- **ESP:** offers functions similar to AH, with the addition of payload confidentiality.

It is crucial to emphasise that confidentiality can only be provided for the payload. In fact, it is never possible to encrypt the (external) header, otherwise, intermediate systems would be unable to process the packets.

There is also a specific protocol for key management and exchange for use with IPsec. It is named Internet Key Exchange, used to create and distribute keys in IP networks. The IPsec security services are the following:

Authentication of IP packets The authentication of the IP packets is achieved by computing a keyed digest using a shared key, which ensures:

- **data integrity:** the recipient can *detect* any packet manipulation;
- **sender authentication:** a formal proof of the sender's identity;
- **(partial) protection against *replay* attacks:** challenges emerge due to operations at Layer 3, where packets may be lost or duplicated.

Confidentiality of IP packets Providing *data privacy*, with payload encryption with a symmetric algorithm and a shared key.

Peer Authentication during SA creation Peer authentication is achieved by a pre-shared key or a digital signature, with key agreement reached just after the authentication procedure complete.

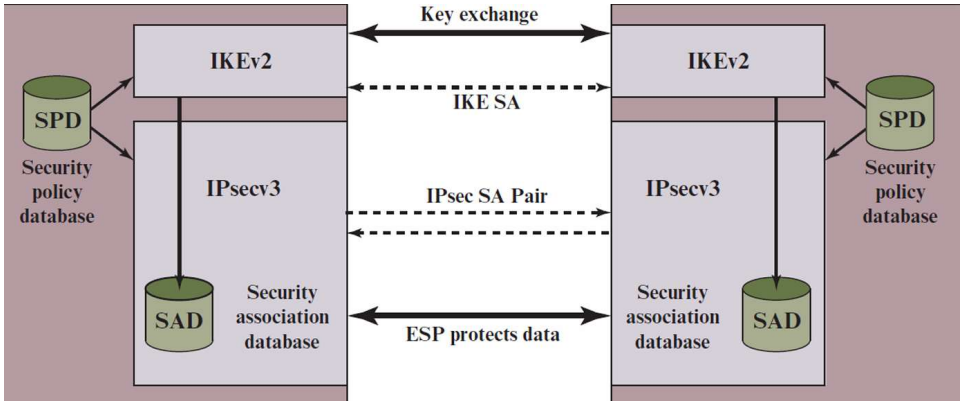

Figure 3.1. IPsec architecture overview

3.2 Security Association (SA)

The above enumerated security features are associated with the concept of Security Association, which is an **one-way logical connection** between a sender and a receiver that affords security services to the traffic carried on it. Each SA is associated with different security services. To achieve full protection for a bidirectional packet flow between two nodes, two SAs are needed.

A SA is uniquely identified by three parameters.

- **Security Parameters Index (SPI)**: 32-bit unsigned integer designated for the SA with local significance alone. The SPI is carried in AH and ESP headers, to enable the receiving system to select the SA under which a received packet will be processed;
- **IP destination address**: address of the destination endpoint of the SA, which may be an end-user system or a network system such as a firewall or router;
- **security protocol identifier**: field from the outer IP header indicates whether the security association is an AH or ESP.

Theoretically, independent security features and algorithms might be employed for each direction. Nevertheless, it is standard practice to employ the same kind of protection, even while utilising two distinct SAs.

Security Associations are administered via two local databases:

- **Security Policy Database (SPD)**: a catalogue of security policies applied to packet flows, either preconfigured (e.g. manually) or provisioned by an automated system (e.g. Internet Security Policy System (ISPS));
- **Security Association Database (SAD)**: a runtime store of active SAs and their attributes (e.g. algorithms, keys, parameters) used to produce protected traffic for each SA.

A SA is further characterised by a set of parameters stored within the **Security Association Database**, each of which defines the operational state and cryptographic context of the secure channel. The already mentioned *SPI*, a *sequence number counter*, also 32 bits in length, is employed to populate the sequence number field in AH or ESP headers, ensuring ordered delivery and potentially replay protection. Associated with it is the *sequence counter overflow* flag, which indicates whether exceeding the sequence limit should trigger an auditable event and suspend packet transmission under the affected SA. Moreover, an *anti-replay window* mechanism provides additional defence against replay attacks by determining whether inbound packets are duplicates of previously received ones. Each SA also specifies the cryptographic material in use: for AH, this includes authentication algorithms, keys and their lifetimes. For ESP, both encryption and authentication parameters are defined alongside keying material and initialisation values. Furthermore, every SA possesses a defined *lifetime*, expressed either as time interval or byte count, after which it must be replaced with a new association (and newSPI) or terminated, as prescribed by policy. Finally, the SAD entry records the operational mode of IPsec, either *transport* or *tunnel*, and the observed *Path Maximum Transmission Unit (MTU)*, which denotes the largest packet size that can be transmitted without fragmentation, together with its ageing variables.

In complex network environments, multiple entries may correspond to a single SA, or conversely, several SAs may be linked to a single **Security Policy Database** entry. Each entry in the SPD is defined by a set of *selectors*, which specify the IP and upper-layer protocol fields used to determine how traffic should be processed. These selectors act as filtering rules that classify outbound packets and map them to the appropriate SA for transmission. Typical selectors include the *remote* and *local IP addresses*, each of which may represent a single host, a list or range of addresses, or a wildcard address. The latter options are particularly relevant when multiple systems share the same SA, such as in scenarios where endpoints reside behind a firewall or gateway. The *next-layer protocol* selector corresponds to the protocol field in the IP header, referred to as *Protocol* in IPv4 and *Next Header* in IPv6, which identifies the protocol operating over IP. Depending on the configuration, this field may specify an individual protocol number, the keyword `ANY`, or, in the case of IPv6, `OPAQUE`. When AH or ESP is employed, the IP protocol header directly precedes the corresponding authentication or encapsulation header within the packet.

In addition to these network layer selectors, an *identity selector* or *name* may be included to represent a user identifier provided by the operating system. Although not contained within IP or upper layer headers, such identifiers are available when IPsec operates within the same host environment as the user application. Finally, *local and remote port numbers*, whether defined as single values, lists or wildcards, enable more granular control over transport layer traffic, supporting selective protection of specific application flows within the overall IPsec policy infrastructure.

3.3 Processing model for IPsec packets

IPsec provides a high degree of granularity in discriminating between traffic that is afforded IPsec protection and traffic that is allowed to bypass it.

When an outbound packet emerges from the TCP/IP stack and is queued for transmission on Layer 2, it is first intercepted by the IPsec subsystem. A t this stage, the system consults the SPD to determine the applicable security policy, deciding whether protection is required and, if so, which mechanisms must be applied. Depending on the outcome, the packet is either passed directly to the data-link layer or subjected to IPsec processing.

If protection is mandated and the packet initiates a flow for which no suitable SA has yet been established, the system triggers the creation of a new SA. On the contrary, when a matching

SA already exists, the IPsec module retrieves the corresponding entry from the SAD, including the selected cryptographic algorithms, keys and operational parameters. The packet is then encapsulated and/or authenticated according to the chosen protocol (ESP or AH). After the required security transformations have been applied, the resulting protected packet is handed off to Layer 2 for transmission over the physical network.

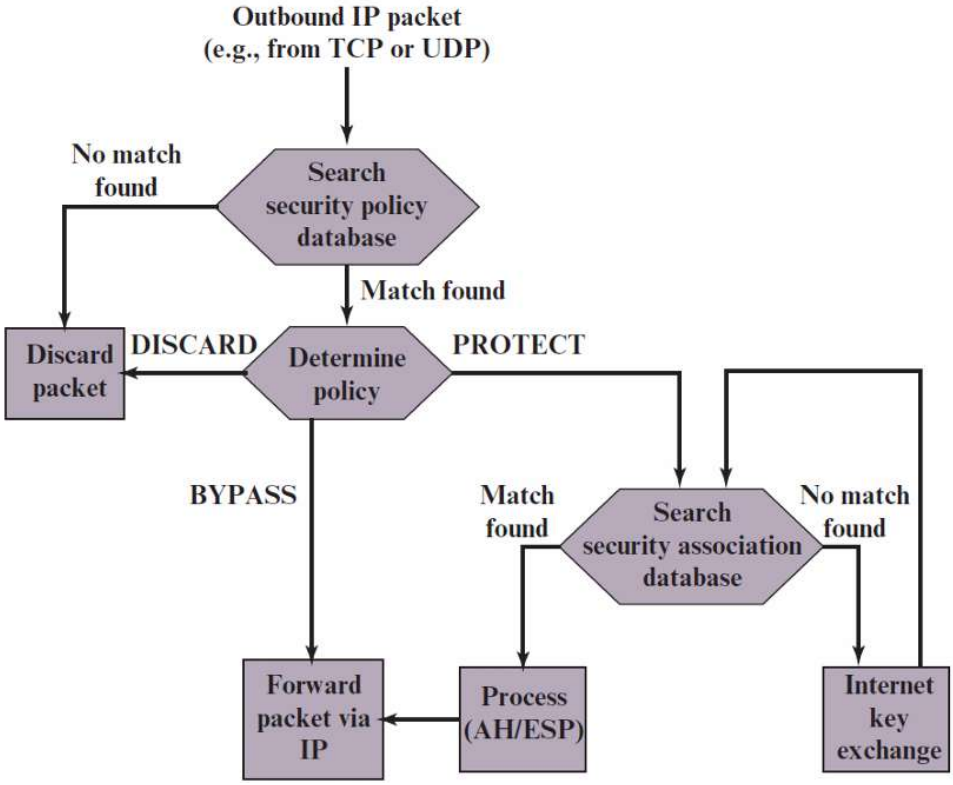

Figure 3.2. IPsec outbound packets processing

3.4 Operating modes

The protocol operates in two distinct modes: **transport** and **tunnel**. These modes govern the extent of encapsulation and protection applied to a packet, the identity of the communicating endpoints and the suitability for host-to-host versus network oriented deployments. The following subsection introduces each mode and outlines their principal semantics and deployment considerations.

3.4.1 Transport mode

Transport mode is utilised for **end-to-end security**, mostly implemented by hosts rather than gateways, except for traffic sent to the gateway itself (e.g. SNMP, ICMP). The original packet, as represented in fig. 3.3, is split into two parts, with a new header placed between the IPv4 header and the TCP/IP one. Consequently, the IPv4 header will signify that it is conveying IPsec rather than TCP/UDP. Within the IPsec header, an additional field will indicate the specific payload being transmitted. Among the advantages, it is computationally efficient, whereas it lacks protection for header variable fields.

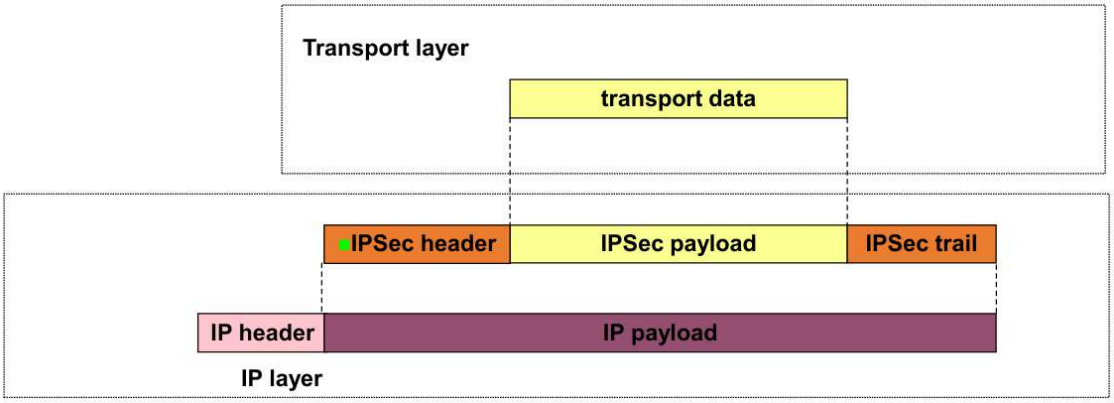

Figure 3.3. Tunnel mode

3.4.2 Tunnel mode

Tunnel mode IPsec is employed to establish a VPN, generally between gateways. The appropriate name is *gateway*, which functions as a contact point between a presumed secure network and another network that is considered not secure. In this modality the original packet, with the end-to-end header, is encapsulated within a tunnel. The tunnel is established by the sending gateway, which secures the network of the corresponding destination. The transmitting gateway implements IPsec on this IP in the IP packet.

The primary advantage is the full protection of the packet, including mutable fields inside the header, yet, it is computationally intensive. While less prevalent, IPsec in tunnel mode may be utilised for end-to-end communication, but it is generally employed between border elements of extensive networks. This configuration is commonly note to as a *site-to-site* VPN, as the organisations involved are generally complete networks.

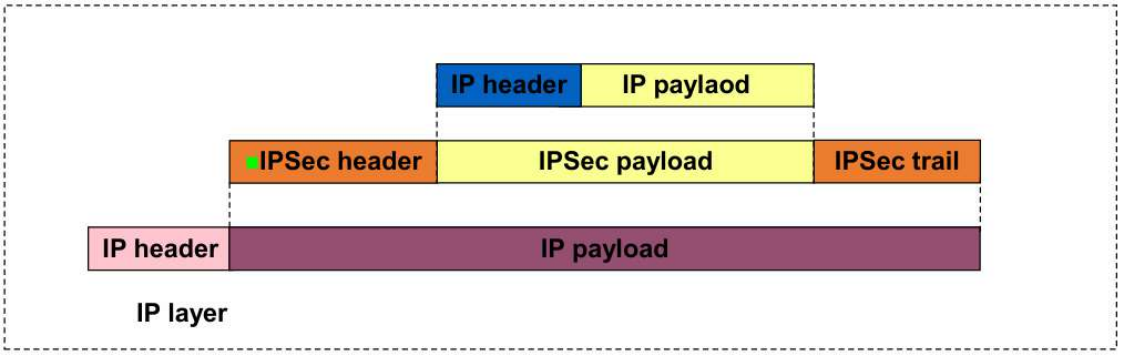

Figure 3.4. Tunnel mode

3.5 Protection mechanisms

This section surveys IPsec's two foundational mechanisms: Authentication Header (AH) and Encapsulating Security Payload (ESP). It outlines their security objectives, packet formats processing rules and clarifies how each interacts with transport and tunnel mode.

3.5.1 Authentication Header (AH)

The Authentication Header has progressed through three main specifications. The first version (RFC-1826), together with RFC-1828 and RFC-1852, provided data integrity and data origin

authentication, mandating support for keyed-MD5 and offering keyed-SHA-1 as an option. The second version (RFC-2402) retained these guarantees and introduced (partial) anti-replay protection, standardising truncated message authentication codes: HMAC-MD5-96 and HMAC-SHA-1-96. The current specification (RFC-4302 [45]) refines processing rules, header structure and clarifies algorithm agility and anti-replay handling.

AH header format (RFC-4302)

Figure 3.5 illustrates the AH placed between the IP header and the upper layer payload. Its fields are:

- **Next Header**: in the IP header, it will indicate the transport of AH, but within the AH, there will be the actual transport packet field;
- **Length**: 1 byte to describe the length of the packet;
- **Reserved**: bytes for future uses;
- **Security Parameters Index (SPI)**;
- **Sequence Number**: monotonically increasing counter supporting anti-replay protection;
- **Integrity Check Value (ICV)**: variable number of 4-byte words to provide authentication data.

Processing of a received AH protected packet

Upon receipt :

1. parse AH and extract the ICV supplied by the sender;
2. normalise the IP packet into a canonical form so that both peers MAC have the same bitstring (e.g. zeroing mutable header fields);
3. select the SA by using the SPI to locate the appropriate entry in the SAD, which determines the algorithm, key, and parameters to be used;
4. recompute the ICV over the normalised packet using the SA's algorithm and key;
5. compare ICVs, if the recomputed value matches the received one, the packet is considered *authentic* and *integral*, otherwise it is deemed fraudulent and/or as a counterfeit sender.

The AH provides assurance about the identity of the peer associated with a given SA. The binding between the SPI and the peer identity is established during SA negotiation via IKE, when the remote endpoint is authenticated. As a result, each packet benefits from implicit peer authentication: the MAC is verified using the key bound to that peer's SA.

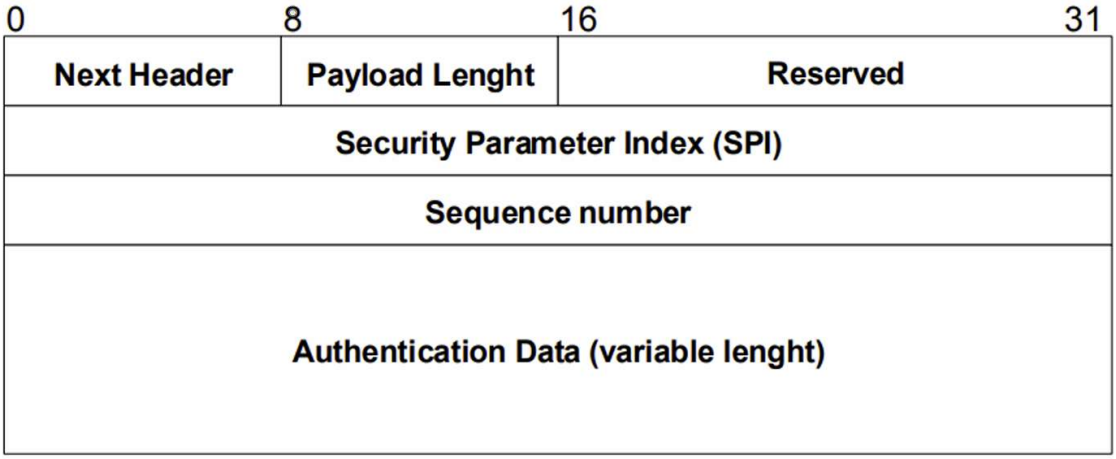

Figure 3.5. Authentication Header (AH) format

3.5.2 Encapsulating Security Payload (ESP)

When confidentiality is required, the ESP must be employed. In its first specification (RFC-1827), it provided confidentiality only, with the reference transform based on DES–CBC (RFC-1829), although other mechanisms were possible. A subsequent version introduced authentication capabilities, albeit not for the IP header, so its coverage does not coincide with that of AH. In practice, this design reduces packet size and saves one SA. Even though, both packet types can be utilised simultaneously without restriction and combined as necessary.

ESP packet format (RFC-4303 [9])

In the current specification, an ESP packet comprises an *ESP Header*, consisting of the 32-bit SPI and a 32-bit sequence number, followed by the protected data and, optionally, an integrity trailer. The protected data, *"Payload Data"*, carries the encrypted content: in **transport mode** this is the upper-layer payload, whereas in **tunnel mode** it is an entire inner IP packet. Where required by the transform, an initialisation vector or nonce is included within this protected portion. The encrypted block also contains any necessary padding together with a 1 octet *"Pad Length"* and a 1 octet *"Next Header"* that identifies the encapsulated protocol. When authentication is configured, an ICV follows as a trailer to provide data-origin authentication and integrity. By design, encryption covers the *Payload Data, Padding, Pad Length, and Next Header*, while (when enabled) the integrity computation authenticates the ESP Header and the encrypted portion according to the selected transform and the SA policy.

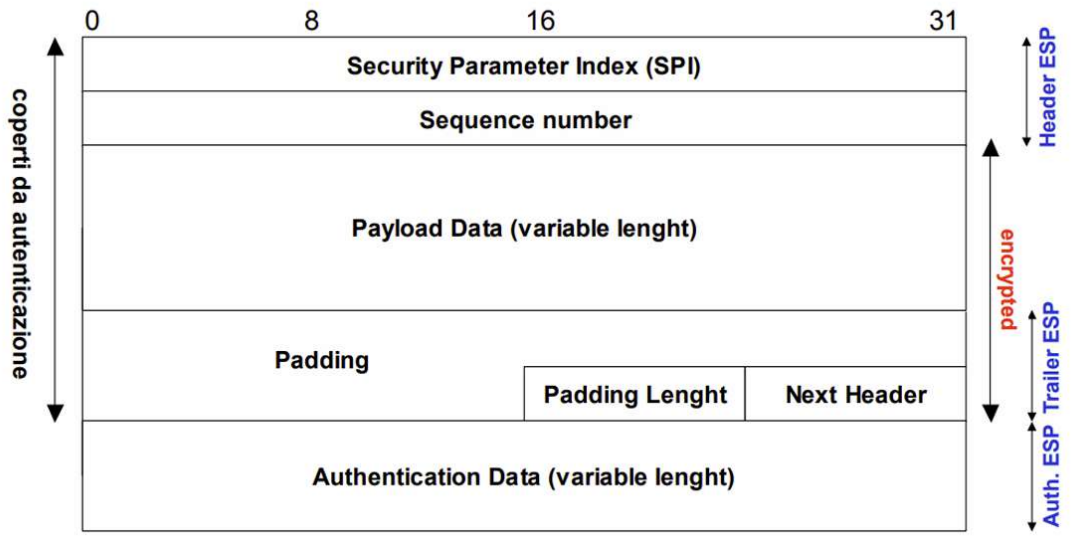

Figure 3.6. Encapsulating Security Payload (ESP) format

ESP supports both *transport* and *tunnel* mode.

In **transport mode**, ESP is inserted between the IP header and the payload. After the ESP header, all subsequent content is encrypted up to the terminating ESP trailer. In this arrangement, the payload is concealed, including information that might otherwise be used for Quality of Service (QoS), filtering, or intrusion detection. While the IP header remains in cleartext.

By contrast, in **tunnel mode**, a tunnel is first established and protection is then applied to the tunnel payload. The entire original packet, including the end-to-end header, is encrypted so that both the payload and the original header are hidden, at the cost of an increased packet size.

3.6 IPsec (partial) *replay* protection

IPsec provides an anti-replay service designed to protect against the unauthorised retransmission of previously captured packets. Each sender maintains a monotonically increasing *sequence number counter* within the relevant SA). Every outbound AH or ESP packet carries this sequence number, which the receiver uses to detect duplicates or replays.

Due to the fact that the underlying IP network is inherently unreliable, packets may be delayed, dropped, duplicated or delivered out of order.

To handle this ambiguity in an efficient manner, IPsec uses a fixed-length *sliding window* of width W that tracks recently accepted sequence numbers. Each position in the window corresponds to a single packet, and the slot is marked when that packet is accepted. When a new packet arrives, if its sequence number falls within the current window and the associated slot is still unset, the packet is treated as a valid out-of-order delivery and accepted. If, instead, the slot has already been marked, the packet is recognised as a duplicate and dropped.

The sliding window advances dynamically as higher sequence numbers are received. Whenever a packet arrives with a value greater than the current maximum, the window is shifted forward to include this new position. This mechanism offers a practical compromise, providing robust replay protection while tolerating a bounded amount of reordering, without requiring the receiver to maintain an unbounded history of all previously seen packets.

For connection-oriented traffic such as TCP, the occasional discard of a legitimate out-of-order packet is of limited concern, as retransmissions are handled by the transport layer. In contrast, for connectionless protocols such as UDP, accepting old packets is not permitted, as doing so could lead to the processing of stale or malicious data. Accordingly, the sender must initiate transmission with a fresh, valid sequence number to remain synchronised within the current SA.

3.7 Mode of use

The IPsec architecture can be deployed in several configurations depending on the desired level of protection, the position of the protected entities, and the operational constraints of the network. The following subsections describe the principal modes of use.

3.7.1 End-to-end security

In this configuration, the IPsec module is enabled directly on the communicating end hosts. The peers establish a *transport mode* SA to form a secure virtual channel between them. This ensures that packets are cryptographically protected at the source host's network layer before transmission, providing confidentiality, integrity, and authentication across untrusted segments such as local or wide area networks.

The main advantage of this approach lies in its independence from the underlying infrastructure: even if intermediate gateways or network segments are untrusted, the traffic remains protected. Consequently, the only feasible attack becomes a Denial of Service. The drawback, however, is the need to enable and manage IPsec on all communicating endpoints. Although modern operating systems natively support IPsec, certain constrained devices such as, mobile platforms or embedded systems, may lack this capability or sufficient computational power to process cryptographic workloads efficiently.

Security management also becomes more complex at scale. When many hosts across a large enterprise network require protection, a coordinated configuration management solution is needed to distribute and maintain SAs in a consistent manner. Furthermore, when ESP encryption is in use, network-level monitoring tools such as Intrusion Detection Systems (IDSs) cannot inspect the protected payloads unless they are deployed directly on the endpoints.

3.7.2 Basic VPN (site-to-site)

In the basic virtual private network (VPN) configuration, the IPsec modules are deployed on the security gateways that interconnect and protect the internal networks. The gateways establish a *tunnel mode* SA to encapsulate and secure all traffic between the two sites. This approach assumes that the internal networks are trusted, and therefore protection is required primarily across the untrusted wide area network.

Although this model simplifies deployment, only the gateways require configuration rather than every endpoint. Confining authentication to the intermediate devices rather than the ultimate communicating peers. Moreover, since encryption and encapsulation are performed at the gateways, traffic remains inspectable within the internal network but not across the external link.

From a performance standpoint, gateways must perform all encapsulation and cryptographic operations for transit traffic, which may require high-end processors or dedicated accelerators such as Hardware Security Modules (HSMs). Nevertheless, the administrative burden is significantly lower, as configuration and key management are centralised on a relatively small number of gateway appliances.

3.7.3 End-to-end security with basic VPN

This hybrid configuration implements the principle of *defence in depth*, combining both end-to-end and site-to-site protection. In this architecture, IPsec is active on the communicating endpoints as well as on the connecting gateways. Two distinct layers of defence are thus established, allowing for flexible security policies.

This mode of use preserves observability inside the local network while enforcing confidentiality on the external path. However, it entails the administrative burden of maintaining configurations on both the endpoints and the gateways.

3.7.4 Secure gateway

The secure gateway model is typically employed to support mobile users who need secure access to an internal enterprise network while travelling or working remotely. In this setup, an IPsec client on the user's device establishes a *tunnel mode* SA with the enterprise gateway. All traffic between the mobile node and internal servers is thereby protected, and the gateway can perform both authentication and authorisation functions to enforce access control policies.

3.7.5 Secure remote access

This configuration extends the secure gateway model by establishing two layers of protection: a *tunnel mode* SA between the mobile user and the gateway, and a separate *transport mode* SA between the mobile user and the final destination host within the network. Typically, the tunnel mode SA is used for user authentication and initial authorisation, granting access to internal resources, while the transport mode SA provides end-to-end data protection. This layered configuration allows organisations to tailor security levels based on the sensitivity of specific applications or services while maintaining robust access control at the network perimeter.

3.8 IPsec version 3

In version 3, ESP has become mandatory, while AH remains optional. As a result, some implementations of IPsec no longer include AH support, thereby providing integrity and authentication solely for the payload rather than for the entire IP packet header. This approach reflects the growing preference for using ESP in both transport and tunnel modes due to its flexibility and wider compatibility with network devices. Furthermore, IPsec v3 extends its functionality to support single-source multicast communications, improving scalability for group-oriented secure transmission.

Given that IPsec is widely deployed in high-throughput settings such as site-to-site VPNs, version 3 tackles the practical risk of sequence-number exhaustion. To prevent sequence number overflow in long-lived sessions, the protocol adopts the Extended Sequence Number (ESN) mechanism with a 64-bit counter: only the lower 32 bits are carried in each packet, while the full value is maintained locally. Use of ESN is enabled by default when operating with IKEv2 [46].

Another significant enhancement is the adoption of AEAD, which consolidates confidentiality and integrity/authentication into a single cryptographic primitive.

With respect to cryptographic algorithms, IPsec v3 supports a wide range of primitives to ensure both backward compatibility and forward security.

For integrity and authentication, supported algorithms include HMAC-SHA2-256-128 (mandatory), HMAC-SHA2-512-256 (recommended), and AES-XCBC-MAC-96 (optional, particularly for IoT profiles), with HMAC-SHA1-96 retained only for legacy interoperability. The NULL authentication option is acceptable *only* when an AEAD cipher is used for ESP.

For confidentiality, AES-GCM-16 is mandatory and ChaCha20-Poly1305 is recommended, while AES-CCM-8 is recommended for IoT deployments. The block cipher AES-CBC remains mandatory for interoperability, whereas 3DES-CBC is *SHOULD NOT* and NULL encryption remains permitted to support "authentication-only" ESP configurations.

In authenticated encryption mode, the recommended AEAD algorithms are AES-GCM-16, AES-CCM-8 (notably for IoT), and ChaCha20-Poly1305, which offer both robustness and efficiency. Furthermore, longer hash digests such as HMAC-SHA2-256-128 and HMAC-SHA2-512-256 can be used to achieve enhanced resistance against collision and preimage attacks [47].

3.9 Key management

The key management portion of IPsec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both integrity and confidentiality.

The IPsec architecture document mandates support for two types of key management:

- **manual**: a system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments;
- **automated**: an automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

In the automated in-band key distribution scenario, it is referred to as the IKE protocol, which performs the following functions: *negotiation of security parameters*, *authentication*, *key establishment* and *key management* post-establishment.

3.9.1 Internet Key Exchange (IKEv1)

In this subsection, a concise overview of the first version of the Internet Key Exchange is provided, outlining only the essential concepts required to appreciate the significant enhancements introduced in its successor, IKEv2.

As defined in RFC-2409 [48], IKEv1 unified two earlier protocols: Internet Security Association and Key Management Protocol (ISAKMP), specified in RFC-2408 [49], which provided the framework for negotiating, establishing, updating and deleting SAs, and the Oakley Key Determination Protocol [50], which provided the authenticated key exchange methods for deriving shared secrets.

The IKEv1 operates in two distinct phases. In *phase 1*, the peers establish an SA to protect subsequent ISAKMP exchanges, thereby creating a secure, authenticated channel for later IKE traffic. This initial phase can be executed either in *Main Mode* or in *Aggressive Mode*.

The **Main Mode** involves six messages and affords identity protection, since authentication data are exchanged only after encryption is in place.

While **Aggressive Mode**, by contrast, completes the exchange in three messages, lowering latency at the cost of exposing peer identities during the handshake. Upon completion of phase 1, both sides hold keys derived from a DH exchange and random nonces, which protect against replay and supply keying material for subsequent negotiations.

Phase 2, also known as *Quick Mode*, uses the protection of the phase 1 ISAKMP SA to negotiate one or more IPsec SAs. These SAs contain the cryptographic transforms and traffic selectors required for the ESP or AH protocols, enabling the secure transmission of payload packets.

The **Quick Mode** typically involves three messages, bringing the **total number of datagrams exchanged to nine** for the establishment of a single IPsec SA.

3.9.2 Internet Key Exchange (IKEv2)

As originally specified in RFC-4306 and subsequently refined through later updates culminating in its current definition in RFC-7296, IKEv2 streamlines its predecessor by establishing a single IPsec SA in just four UDP datagrams (fig. 3.7). The protocol follows a strict request-response pattern in which each response also serves as an explicit acknowledgement. Retransmission responsibility lies solely with the initiator, a design that markedly improves robustness under loss and congestion compared with IKEv1, where both peers might otherwise retransmit concurrently.

The initial IKE_SA_INIT round negotiates the cryptographic transforms for the IKE SA ($SA1_i/SA1_r$), performs a shared DH secret derivation via KE_i/KE_r, exchanges nonces N_i/N_r, all within a single round trip.

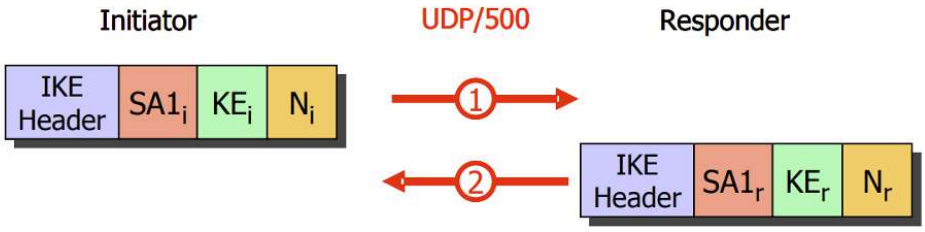

Figure 3.7. IKE_SA_INIT

The subsequent IKE_AUTH exchange authenticates the peers ($AUTH_i/AUTH_r$) using pre-shared keys, RSA signatures or Extensible Authentication Protocol (EAP), and simultaneously installs the first Child SA by conveying traffic selectors. The (TS_i/TS_r) and the transforms for the IPsec connection ($SA2_i/SA2_r$). Each party supplies its identity (ID_i/ID_r) and may attach a certificate ($CERT_i/CERT_r$). The initiator can also request that the responder assume a specific identity value if multiple identities are available.

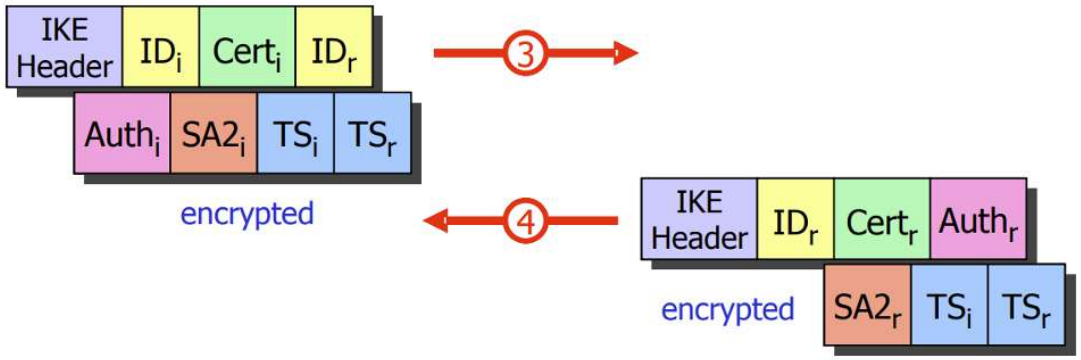

Figure 3.8. IKE_AUTH

Further Child SAs are established using the CREATE_CHILD_SA request/response pair (fig. 3.9). This exchange conveys the new transform proposals (SA_i/SA_r), fresh nonces (N_i/N_r), optional DH material when perfect forward secrecy is required and the corresponding traffic selectors (TS_i/TS_r). The same mechanism also enables periodic rekeying of either a Child SA or the parent IKE SA via an appropriate notification payload.

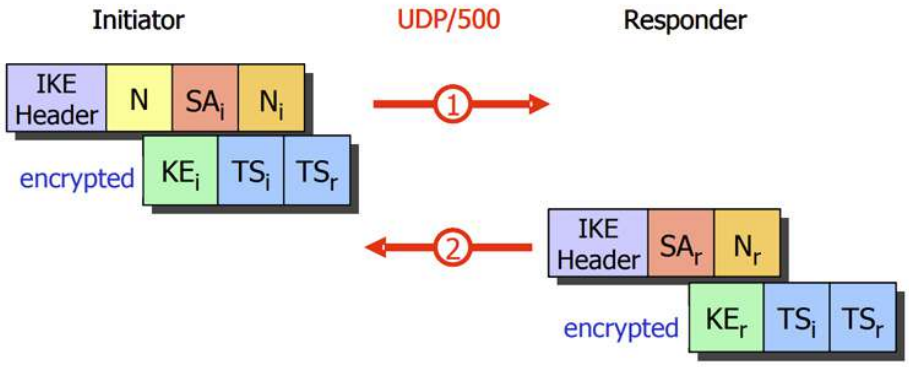

Figure 3.9. CREATE_CHILD_SA

At any time, either peer may send an `INFORMATIONAL` exchange, which is always acknowledged. As shown in fig. 3.10, such messages can carry notification (`N`), delete (`D`), or configuration payloads (`CP`). Empty `INFORMATIONAL` exchanges are commonly used to implement Dead Peer Detection (DPD).

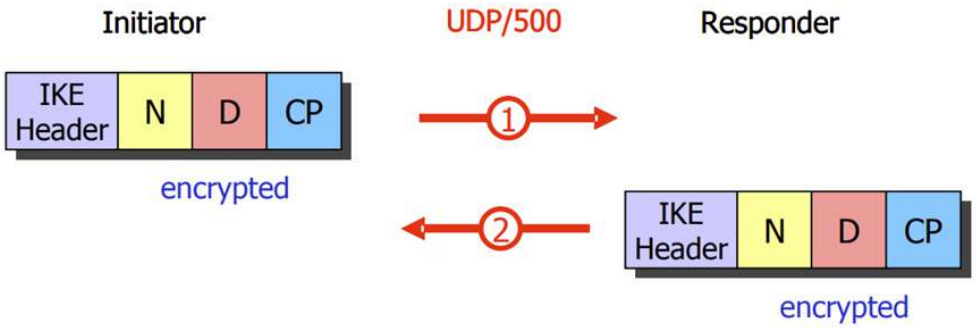

Figure 3.10. INFORMATIONAL MESSAGES

3.9.3 Intermediate Exchange and Additional Key Exchanges

As specified in RFC-9242, the *Intermediate Exchange* was introduced as a substantive enhancement to IKEv2, intended to remedy operational inefficiencies and security limitations inherent in the traditional two-exchange workflow. In the baseline design of RFC-7296, the `IKE_SA_INIT` and `IKE_AUTH` exchanges are tightly coupled, forcing substantial cryptographic payloads to be transmitted early in the handshake. This frequently results in fragmentation, especially over User Datagram Protocol (UDP) when multiple certificates, large keys, or complex authentication data are present, thereby elevating loss rates and jeopardising a successful negotiation.

The Intermediate Exchange (IE) defines an optional, standardised facility for inserting additional message pairs between `IKE_SA_INIT` and `IKE_AUTH`. This intermediate stage can be used to gather pre-authentication data, negotiate advanced cryptographic options, or conduct external authentication procedures (e.g. EAP), without prematurely committing to authentication. In doing so, it improves robustness in the face of packet loss and affords greater flexibility for large-scale deployments, including mobile scenarios and environments with constrained devices. However, it also introduces security considerations: any intermediary message must preserve cryptographic

binding to the IKE_SA_INIT exchange to avoid man-in-the-middle and replay attacks. To mitigate these risks, RFC-9242 mandates that all intermediate messages remain cryptographically protected and authenticated within the negotiated IKE_SA context.

From an operational standpoint, the IE addresses the problem of UDP fragmentation by allowing the division of large payloads into smaller, manageable segments transmitted in separate messages, making use of the existing IKE fragmentation mechanism, avoiding Internet Protocol (IP) fragmentation thus reducing retransmissions and negotiation failures. Furthermore, it enables new use cases, such as staged authentication or exchange of post-quantum public keys, by deferring the full authentication until all cryptographic material is safely exchanged. Nonetheless, implementations must consider potential DoS vectors introduced by additional exchanges, ensuring validation and rate-limiting mechanisms are applied.

Based on these concepts, RFC-9370 introduces *Additional Key Exchange* within IKEv2. Where the traditional protocol performs a single Diffie-Hellman exchange during `IKE_SA_INIT` (with an optional exchange in `CREATE_CHILD_SA` for Perfect Forward Secrecy (PFS) during the rekey phase), RFC-9370 extends the model to permit multiple, concurrent key exchanges both in `IKE_SA_INIT` and in subsequent `CREATE_CHILD_SA` exchanges. In order to strengthen cryptographic agility and accommodate hybrid post-quantum deployments, combining classical and post-quantum key material, ensuring that session keys remain secure even if one primitive is later broken.

From an operational perspective, `Additional Key Exchange (ADDKE)` transform types are carried within the `IKE_SA_INIT` SA payload, each one identifying an additional KE method through its specific `ADDKE` transform identifier (e.g. `ADDKE2`, `ADDKE3`, and so on). These transforms announce the extra key exchanges that the peers will subsequently perform during `IKE_INTERMEDIATE` exchanges.

During the `IKE_AUTH` phase, the contributions of all negotiated `ADDKE` transforms are incorporated into the shared-secret derivation, resulting in a hybrid Key Derivation Function (KDF). The same principle applies to CREATE_CHILD_SA exchanges, where additional ADDKEs may be included if required.

The specification introduces two complementary safeguards for managing multiple key exchanges. First, *ordering and dependency* are enforced by stipulating that each $\texttt{ADDKE}_i$ must be fully completed and validated before processing $\texttt{ADDKE}_{i+1}$. In practice, this yields a strict, deterministic sequence: any reordering, omission or duplication of `KE`/`ADDKE` payloads causes the exchange to fail immediately. Second, *cryptographic binding* is achieved by feeding the shared secret from each key exchange into the key schedule in the same order, using the negotiated Pseudo Random Function (PRF).

As originally defined in RFC-7296, the peers first derive a base set of keys—namely SK_d, $SK_{ai/ar}$ and $SK_{ei/er}$—from the initial DH exchange in `IKE_SA_INIT`. They then perform one or more `IKE_INTERMEDIATE` exchanges, each protected using the previously established $SK_{e[i/r]}$ and $SK_{a[i/r]}$ keys.

After each IE, the `SKEYSEED` value is refreshed by incorporating the newly derived shared secret $SK(n)$ from the corresponding ADDKE, together with the nonces N_i and N_r, as follows:

$$SKEYSEED^{(n)} = \mathrm{PRF}\big({SK_d}^{(n-1)},\ SK(n) \,\|\, N_i \,\|\, N_r\big).$$

The updated $SKEYSEED^{(n)}$ is then expanded via the PRF to obtain the refreshed keying material:

$$\{{SK_d}^{(n)},\ {SK_{ai}}^{(n)},\ {SK_{ar}}^{(n)},\ {SK_{ei}}^{(n)},\ {SK_{er}}^{(n)},\ {SK_{pi}}^{(n)},\ {SK_{pr}}^{(n)}\}$$

$$=$$

$$\mathrm{PRF}^{+}\big(SKEYSEED^{(n)},\ N_i \,\|\, N_r \,\|\, \mathrm{SPI}_i \,\|\, \mathrm{SPI}_r\big).$$

After each update, both peers compute the intermediate authentication values ${IntAuth_{i/r}}^{(n)}$ using the corresponding ${SK_{p[i/r]}}^{(n)}$ keys [11]. These intermediate values are later used during the `IKE_AUTH` exchange to produce the final authentication data (`InitiatorSignedOctets` and `ResponderSignedOctets`), thereby binding the authentication step to all preceding key exchanges and providing robustness against downgrade or truncation attacks.

Fragmentation is handled in following manner. Since multiple key exchanges naturally increase message sizes, RFC-9370 relies on the fragmentation framework of RFC-7383 [51] to segment and reassemble large payloads without compromising integrity. Implementers must nonetheless ensure that fragmentation behaviour does not inadvertently leak information about message structure or group selection, which could in turn be exploited through side-channel analysis.

For IEs, the main concern is that partially authenticated states could be abused to inject or replay intermediate messages. Within RFC-9370, the central challenge is to bind multiple key exchanges securely and to coordinate their inclusion in the key-derivation process. This introduces additional operational complexity: peers must negotiate supported groups, accommodate asymmetric capabilities and absorb the computational cost of parallel key exchanges without unduly harming performance or introducing timing side channels.

Finally, RFC-9370 endorses the *Childless IKE_AUTH* mode, as defined in RFC-6023 [52], which allows completion of the `IKE_SA` setup without creating a corresponding `CHILD_SA` in the same phase. This is particularly useful in scenarios where key management and policy negotiation must be finalised before the data plane is brought up, such as deferred VPN activation, zero-trust authentication workflows or post-quantum rekeying strategies.

Operationally, this mechanism permits the initiator and responder to agree on establishing an IKE_SA alone, confirmed by the `CHILDLESS_IKEv2_SUPPORTED` notification in the `IKE_SA_INIT` response. Subsequent `CREATE_CHILD_SA` exchanges can then be used to derive traffic keys, optionally incorporating further post-quantum key exchanges via `IKE_FOLLOWUP_KE` messages. In this model, peers may omit the initial Child_SA entirely, ensuring that all later CHILD SAs benefit from quantum-resistant key material.

From a security perspective, establishing a childless IKE_SA also mitigates the risk of unauthenticated DoS attacks. Since no Child_SA or data-plane channel is created prior to completing peer authentication, attackers cannot easily exploit unauthenticated requests to consume cryptographic or networking resources. This effectively narrows the attack surface during the initial exchange phase, ensuring that heavy cryptographic operations, especially those involving post-quantum primitives, are only performed once authenticity has been verified.

Chapter 4

StrongSwan

Having delineated the overarching architecture of IPsec, it is useful to view a "secure channel" not as a monolithic artefact but as a policy-driven construction: a set of negotiated algorithms, parameters and traffic selectors bound into SAs and enforced by the system's security policy & association databases. In practice, the semantics of the channel are fixed by policy (what to protect and how), while the mechanics are realised by IKE state machines that instantiate and renew the requisite SAs. This separation of concerns is fundamental to engineering cryptographic change: policy expresses intent. The key management layer executes it.

strongSwan is a widely deployed open-source implementation of the IPsec architecture with a particular focus on the IKE key management protocol. In contemporary deployments, strongSwan is typically used to negotiate SAs and enforce policy for protected traffic flows, while the **operating system kernel** performs the actual ESP/AH processing once the negotiated parameters are installed. This division of responsibilities allows strongSwan to provide rich control-plane features without handling user-plane packets directly. In particular, IKEv2 exchanges are transported over UDP/500 (or UDP/4500 when NAT traversal is in use), and are used to create and rekey IKE and Child SAs that, in turn, control ESP/AH processing in the kernel [53].

4.0.1 Daemon and modular architecture

At the heart of strongSwan is the *charon* daemon, implemented largely in the reusable `libcharon` library so that several front-ends (`charon`, `charon-systemd`, `charon-svc`, `charon-cmd`, NetworkManager's `charon-nm`, Android) can share the same IKEv2 state machine.

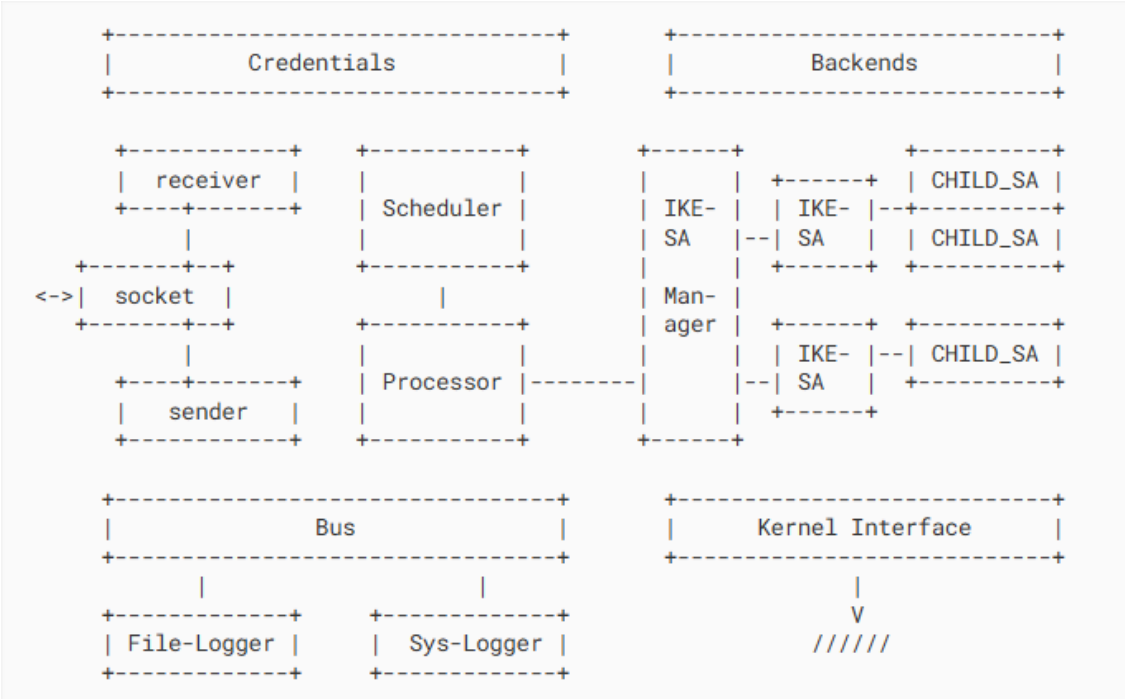

Figure 4.1. Authentication Header (AH) format

The daemon exposes a strictly modular plugin architecture: at start-up, plugins are loaded to extend core functionality (credential back-ends, EAP methods, PKI helpers, logging/transcript sinks, control interfaces such as *VICI*, SQL configuration back-ends and more). Internally, `charon` uses a job processor (*thread pool*), a scheduler for timed events (e.g. rekey), an *IKE_SA manager* to serialise access to IKE state, and a kernel interface to install SAs, policies, routes and virtual addresses. A message *bus* provides pub/sub signalling to interested listeners, enabling rich logging and external orchestration.

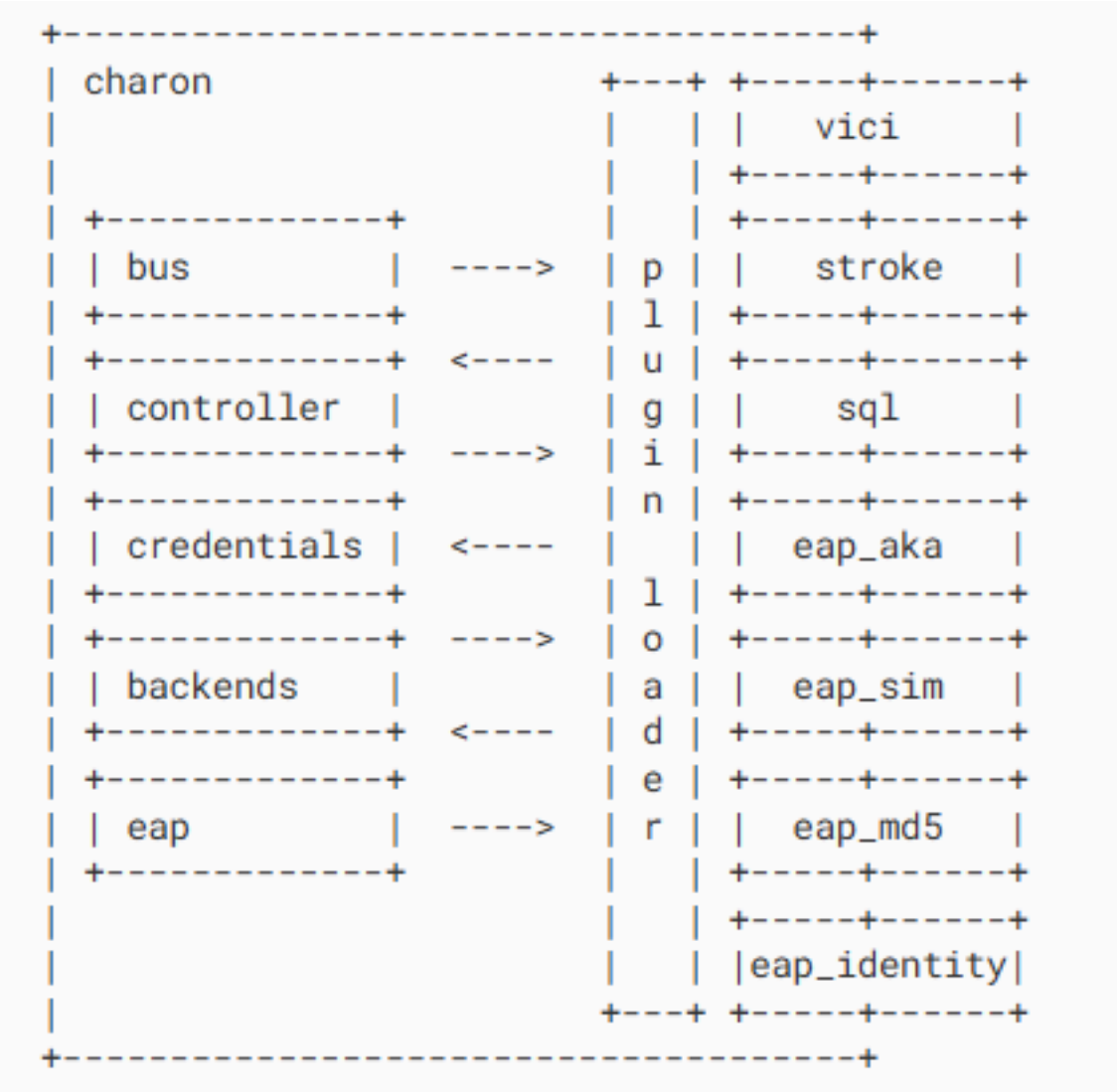

Figure 4.2. Authentication Header (AH) format

`libcharon`

Most of the IKEv2 logic resides in `libcharon`. Parsing and proposal negotiation are primarily implemented under `src/libcharon/sa` and `src/libcharon/encoding`. Key components include:

- `sa/ike_sa.c`, `sa/ike_sa.h`: define `ike_sa_t` and handle the lifecycle of IKE_SAs, including IKE_SA_INIT and IKE_AUTH;
- `sa/ike_sa_manager.c`: coordinates IKE_SA management (e.g. half-open checks, message reordering) and dispatches messages to the appropriate IKE_SA;
- `sa/child_sa.c`, `sa/child_sa.h`: implement creation and reconfiguration of CHILD_SAs (CHILD_SA_INIT), including ESP proposal negotiation;
- `encoding/payloads/*.c`: parse and generate IKE payloads, notably `SA` payloads (with `Proposal` and `Transform` sub-payloads). While payload types are declared in `payloads/payload.h`;
- `plugins`: `src/libcharon/plugins/` hosts existing plugins (e.g. `whitelist`, `duplicheck`) illustrating integration with negotiation. Ultimately, `src/libcharon/sa/credential.c` manages identities and credentials used during IKE_AUTH, useful for inspecting remote IDs.

Plugin interfaces

`charon` plugins implement the `plugin_t` interface and are loaded at start-up. A plugin can interact with the daemon via:

- *Bus subscriptions and listeners* (`listener_t`): plugins register with the internal event bus (`charon->bus->add_listener(listener)`) and expose callbacks that fire on IKE lifecycle events, for example:
  - `listener_t::ike_updown`: notification when an IKE SA is established or torn down;
  - `listener_t::child_updown`: notification when a CHILD SA is created or closed;
  - `listener_t::message`: hook for raw IKE messages sent or received;
  - `listener_t::ike_key`: key derivation events (e.g. `SK_ai`, `SK_ar`, `SK_ei`, `SK_er`);
  - `listener_t::authorize`: post-authorisation hook to allow/deny based on identities or policy.

  These hooks afford full visibility and targeted intervention throughout the exchange (e.g. `updown` for SA installation and `duplicheck` to handle duplicate sessions). A plugin can filter by connection context and negotiated parameters before deciding on enforcement.
- *controllers and back-ends*: `controller_t` enables programmatic start/stop of IKE_SA (e.g. scheduling jobs similar to `charon down`). While `backend_t` supplies configuration from external sources.

4.0.2 Out-of-band control: VICI and `swanctl`

Administrative control may be decoupled from the daemon's internal state and performed out of band via the *Versatile IKE Control Interface*. VICI is an RPC-style interface provided by the `vici` plugin in `charon`. It is designed for inter-process communication and exposes reliable request/response operations plus asynchronous event notifications over a local transport: by default, a UNIX domain socket. As the protocol itself offers neither wire-level security nor authentication, deployments are expected to use the UNIX domain socket with restrictive file permissions rather than a network socket.

Thus, external tools can configure, control and observe the IKE daemon without embedding strongSwan components or requiring manual intervention. Moreover, the `vici` plugin publishes event notifications to signal material changes, such as connection state transitions or report certificate expiration, by enabling real-time monitoring and prompt adaptation in dynamic IPsec deployments.

Contemporary systems interface with `charon` via `swanctl`, a command-line client that speaks VICI. The `swanctl` tool is central to manage IPsec-based VPNs, providing exemplifications to configure, control and monitor the daemon on demand. It provides numerous subcommands to direct how the IKE daemon manages secure connections, inspect current connection states, retrieve detailed logs and obtain information about overall daemon status. A set of `--load-`prefixed commands reads runtime configuration, such as connections, secrets and IP address pools, from the `swanctl.conf` file, which serves as the central point for defining secure channels. Operational tasks, including initiating or terminating IKE/CHILD SAs, are driven entirely via VICI calls, avoiding direct manipulation of kernel state and supporting automated provisioning and lifecycle management.

4.0.3 Configuration files and directory layout

strongSwan adopts a layered configuration model. Global daemon and plugin options are expressed in `strongswan.conf` and modular snippets under `/etc/strongswan.d/`. Per-connection policy and credentials are defined in `/etc/swanctl/swanctl.conf`. The daemon reads these artefacts at start-up and, when required, on demand via VICI. Inclusion and inheritance mechanisms are used throughout to keep large configurations maintainable.

strongswan.conf and /etc/strongswan.d/

The strongswan.conf file specifies global settings for charon and its plugins using a hierarchical key–value configuration syntax.

Individual configuration snippets are placed under /etc/strongswan.d/ and can be brought into strongswan.conf with directives such as:

```
include /etc/strongswan.d/charon/*.conf
```

Included files are merged recursively into the active configuration scope: if a section with the same name already exists, the imported content extends it. When keys collide at the same level, the values from the included file take precedence. This approach enables administrators to activate or deactivate plugins and adjust default settings without modifying the codebase. References are resolved at runtime, and fully qualified section names allow configuration values to be inherited from nested subsections.

/etc/swanctl: directory and file layout

The /etc/swanctl directory hosts the primary configuration file, swanctl.conf, together with several subdirectories that store file-based credentials and private keys consumed by charon when operating IKEv2. With the adoption of swanctl, swanctl.conf supersedes the legacy ipsec.conf for runtime policy configuration. It uses the same hierarchical syntax as strongswan .conf: unspecified options fall back to sensible defaults, and configurations can be loaded or reloaded dynamically via VICI or the swanctl command-line tool.

swanctl.conf

Operational policy is expressed in /etc/swanctl/swanctl.conf, which is organised into four top-level sections:

- connections: negotiation policy for IKE and CHILD SAs (proposals, lifetimes, traffic selectors);
- authorities: metadata and attributes for certification authorities;
- secrets: authentication material (e.g. PSK, private keys, EAP credentials);
- pools: named address pools and associated virtual-IP settings.

The connections section is the most substantial. Each connection is a uniquely named subsection describing the IKE_SA on which subsequent CHILD SAs depend. Moreover, within a connection, child names must be unique in that connection's scope. To complete an IKE_SA definition and prescribe the associated CHILD SAs, three further subsections are used:

local parameters for the local authentication round (multiple rounds may be declared by suffixing distinct local blocks);

remote parameters for the peer's authentication (multiple rounds may likewise be declared via suffixed remote blocks);

children one or more uniquely named child definitions, each specifying ESP proposals and selectors for a single CHILD SA.

The remaining sections are structured as follows:

- authorities: each authority is a subsection with a unique name and options such as CA certificates or validation services.

- `secrets`: subsections are keyed by secret type, with suffixed names uniquely identifying each entry and holding the pertinent key–value pairs (e.g. `id`, `secret`);
- `pools`: uniquely named pools that connections may reference to assign virtual addresses and related attributes.

Listing 4.1. Illustrative `/etc/swanctl/swanctl.conf`

```
connections {
  gw-wan-in-bankA {
    local_addrs = 203.0.113.2
    remote_addrs = 198.51.100.10
    version = 2
    encap = yes
    reauth_time = 0
    rekey_time = 120s

    local { auth = pubkey; certs = gateway-ml-dsa.pem; id = pep-gateway; }
    remote { auth = pubkey; id = banka; cacerts = pq-root-ca-ml-dsa.pem; }

    proposals = aes256gcm16-prfsha512-ecp521

    children {
      inbound-legacy-pay-L3 {
        local_ts = 10.200.0.0/24
        remote_ts = 198.51.100.10/32
        esp_proposals = aes256gcm16-ecp521
        rekey_time = 90s
        start_action = none
        updown = /usr/local/sbin/updown-verifier.sh
      }
    }
  }
}

secrets {
  private-gateway-ml-dsa { file = etc/swanctl/private/gateway-ml-dsa-key.pem }
}

authorities {
  pq-root-ca-ml-dsa { cacert = pq-root-ca-ml-dsa.pem }
}
```

Chapter 5

Open Policy Agent

This section provides an in-depth analysis of OPA, covering its design philosophy, policy language, operational model and configuration, management practices and representative use cases.

The OPA is an open-source, general-purpose policy engine that unifies policy enforcement across the stack. Graduated project within the Cloud Native Computing Foundation (CNCF) landscape, OPA provides a high-level declarative language for expressing *policy as code*, together with straightforward APIs that allow to offload policy decision-making from software. It may be utilised to enforce rules across *microservices*, *Kubernetes*, *CI/CD pipelines*, *API gateways* and more [54].

5.1 Design

OPA separates policy decisions from their enforcement. Instead of hard-coding rules into applications, a service that requires a decision submits a query to the policy engine together with structured input (e.g. JSON). The engine, which is agnostic to the input schema and can consume arbitrary structured data, evaluates this input against the configured policies and any accompanying datasets to return a decision.

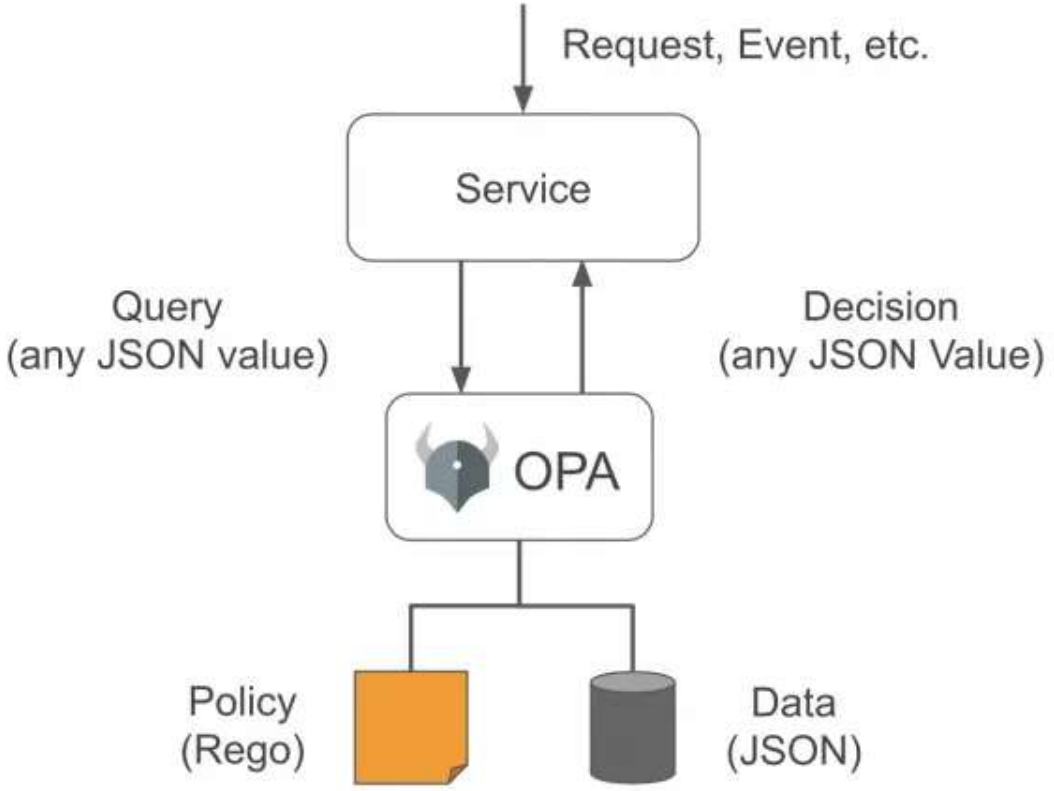

Figure 5.1. OPA philosphy

Given that OPA and its policy language, *Rego*, are domain-agnostic, policies can capture a broad spectrum of invariants, such as:

- which users may access particular resources;
- to which subnets egress traffic is permitted;
- on which clusters a workload must be deployed;
- from which registries binaries may be downloaded;
- with which OS capabilities a container may execute;
- at what times of day the system may be accessed.

Outcomes are not be limited to simple allow/deny results. Just as inputs may be arbitrarily structured, policies can yield rich, structured responses, enabling applications to act on descriptive guidance rather than a binary response.

5.1.1 Philosophy

A *policy* is a set of rules that governs the behaviour of a software service. Policies may prescribe rate limits, enumerate trusted servers, specify the clusters to which an application should be deployed, define permitted network routes or constrain the accounts from which a user may withdraw funds.

Authorisation vs. authentication

Authorisation is a particular class of policy that determines which people or machines may perform which actions on which resources. It is often confused with *authentication*, which concerns how entities prove their identity. While policy (and authorisation in particular) frequently relies on authentication outputs (e.g. username, attributes, groups, claims), decisions typically depend on a broader context than identity alone. Indeed, many policy decisions are independent of users altogether and instead capture system-wide invariants (e.g. "all binaries must originate from a trusted source").

Why decouple policy?

All organisations maintain policies. They are essential to long-term success because they encode critical knowledge about legal compliance, technical constraints and the avoidance of repeated mistakes. In the simplest case, policies may be applied manually, based on written rules or tacit conventions embedded in organisational culture. They may also be enforced programmatically in application logic or specified statically at deployment time. In many systems, policy is still hard-coded into the very services it governs. OPA enables policy to be externalised, so that policy owners can read, write, analyse, version, distribute and manage it independently of the application. OPA also provides a unified toolset to decouple policy from *any* software service and to express context-aware rules over *any* relevant data. In short, OPA helps to separate any policy, using any context, from any software system.

Software services should allow policies to be defined *declaratively*, updated at any time without recompilation or redeployment and enforced automatically, particularly when decisions must be made faster than is humanly possible. Decoupling policy supports these aims at scale: it improves adaptability to changing business requirements, enhances the discovery of violations and conflicts, increases the consistency of compliance, and reduces the risk of human error. Decoupled policies are also more resilient to external change factors the original developers may not have anticipated.

The costs of building in-house

In the absence of OPA, policy management must be constructed from first principles: design a policy language (with clear syntax and semantics), implement and optimise an evaluation engine, and then test, document and maintain the full stack to guarantee correctness and a satisfactory developer experience. In addition, security, tooling and day-to-day operations require careful design and sustained attention.

This constitutes a significant, ongoing investment that OPA is intended to reduce. OPA offers a comprehensive policy engine that externalises decision-making from application code. Functioning as a *concierge* for services, it answers fine-grained, context-aware queries on users' behalf, providing the primitives needed to deliver consistent control and visibility over policy across disparate systems.

5.1.2 The document model

OPA policies, written in `Rego`, operate over hierarchical structured data. This information is often described as a *document*, a set of attributes, contextual input, or simply "JSON". As mentioned, OPA is domain-agnostic: policies may consume *arbitrary* structured data and produce decisions that are likewise arbitrarily structured (e.g. booleans, strings, objects, lists or nested combinations thereof).

Base and virtual documents

Data originating outside OPA, is loaded into the engine via push or pull interfaces, either synchronously or asynchronously with respect to evaluation. Such externally supplied data is called

base documents. Policies frequently depend on these base documents, but rules can also build on one another. The values computed by rules (i.e. policy decisions) are called *virtual documents*. "Virtual" indicates that the value is derived during evaluation rather than being externally loaded.

Base and virtual documents share the same underlying value domain (numbers, strings, lists and maps) and are addressed uniformly via dot and bracket notation, so that policy authors can rely on a single, consistent modelling and referencing style. Both categories of document are made available under the global `data` namespace.

Although JSON and YAML are frequently used as input formats, OPA itself is not bound to any specific external encoding. Internally, it stores information in its own native representation of objects and arrays.

Namespacing and placement

Because base documents are supplied by external systems, their position under `data` is determined by the component responsible for loading them. By contrast, the placement of virtual documents is governed by the policy itself via the language's `package` directive.

Synchronous vs. asynchronous loading and access paths Base documents may be:

- *asynchronously* pushed or pulled so that `data` is refreshed when the world changes (e.g. periodically or on events such as database notifications). These documents are read via the `data` namespace and are cached in memory for efficient evaluation;
- *synchronously pushed* as part of a policy query. Such per-request context is exposed under the global `input` variable to avoid name clashes with `data`;
- *synchronously pulled* during evaluation using built-in functions (e.g. `http.send`). Return values can be bound to local variables and surfaced via virtual documents.

Moreover, HTTP requests such as `GET /v1/data` or `GET /v1/data/foo/bar` are translated internally into `Rego` queries that mirror the request path (e.g. `data` or `data.foo.bar`).

5.2 Policy Language

In OPA, policies are written in `Rego`, a declarative language with Datalog roots, extended to traverse nested structures (e.g. JSON). A `Rego` query states properties over the data managed by OPA and returns a decision. Deviations are detected by enumerating elements that diverge from the intended system state. The language is readable, unambiguous and benefits from optimisations in OPA's evaluator.

5.2.1 Rego at a glance

Rules define *virtual documents*. The simplest rule binds a scalar. Composite values are equally natural.

Listing 5.1. Scalars and composites

```
package example
pi := 3.14159
rect := {"width": 2, "height": 4}
same := rect == {"width": 2, "height": 4}
```

Guarded rules and truth A rule body is a conjunction: it holds when all expressions are true.

Listing 5.2. Guarded rule

```
package example
ok if {
  x := 42
  y := 41
  x > y
}
```

Referencing nested data Dot/bracket references traverse structures.

Listing 5.3. Existential query over a collection

```
package sites
sites := [{"name": "prod"}, {"name": "dev"}]

prod_exists if {
  some s in sites
  s.name == "prod"
}
```

5.2.2 Values and collections

Scalars (strings, numbers, booleans, `null`) and **composites** (arrays, objects, sets) are first-class. Arrays preserve order and duplicates, objects map arbitrary keys to values, sets are unordered, unique, and serialise to arrays when emitted as JSON.

5.2.3 Variables and references

Variables may appear in rule heads and bodies and act as both inputs and outputs. Head variables must be bound by a non-negated equality in the body. References can use variables (or "_" as throwaway iterators) to range over collections.

5.2.4 Comprehensions

Array, object and set comprehensions build collections from sub-queries, closing over outer variables:

Listing 5.4. Array comprehension (join)

```
package comp
import data.example.apps
import data.example.sites

app_to_hostnames := {app.name: hostnames |
    app := apps[_]
    hostnames := [hostname |
                  name := app.servers[_]
                  s := sites[_].servers[_]
                  s.name == name
```

```
                    hostname := s.hostname]
}
```

5.2.5 Rule styles

Rules may define **sets** or **objects** (partial definitions are unionised) or give **complete** definitions (single value).

Listing 5.5. Sets, objects, complete vs. incremental

```
package forms
import data.example.sites, data.example.apps

hostnames contains h if { h := sites[_].servers[_].hostname }

apps_by_hostname[hostname] := app if {
    some i
    server := sites[_].servers[_]
    hostname := server.hostname
    apps[i].servers[_] == server.name
    app := apps[i].name
}

pi := 3.14159 # complete definition
```

5.2.6 Functions

User-defined functions mirror built-ins and return exactly one value.

Listing 5.6. User function

```
package fun
trim_split(s) := parts if { parts := split(trim(s, " "), ".") }
```

Functions may be defined incrementally (multiple clauses) but *not* overloaded by arity.

5.2.7 Negation and universal quantification

Negation asserts non-existence and is safe only when variables are bound elsewhere in the rule. Universal conditions are expressed via `every` or via "existential + negation".

5.2.8 Built-ins and error handling

`Rego` offers a rich suite of built-ins (arithmetic, aggregation, string and set operations, time, I/O, cryptography). By default, runtime errors evaluate to *undefined* and do not halt execution.

5.3 Control and management

OPA exposes management APIs for unified, logically centralised policy operations. These interfaces allow to build a control plane that distributes policy artefacts and collects operational telemetry (e.g. decision logs), while individual OPAs enforce policy locally for low latency and high availability.

5.3.1 Deployment models

Agent (sidecar) model OPA runs alongside an application and is invoked over HTTP. Policies and data reside in OPA's in-memory store. The service offloads decisions to the colocated agent, minimising network hops and failure domains.

Distributed enforcement In a distributed pattern, a separate OPA instance is deployed alongside each service instance (or per pod/node), so that decisions are taken locally. This approach provides consistently low latency and improves resilience by decentralising the enforcement of policy.

5.3.2 Management APIs

An OPA agent can be configured to contact management endpoints that provide unified control and observability:

- **bundles**: distribution and update of policies and data;
- **decision logs**: streaming of decision telemetry for audit and analytics;
- **status**: reports on agent health and the state of bundles and plugins;
- **discovery**: dynamic retrieval of the agent's own configuration.

Exposing these APIs enables centralised governance across fleets of OPAs, while preserving local, in-memory evaluation at the edge. Note, however, that OPA does not include a control-plane service. Integration with third-party control plane is required.

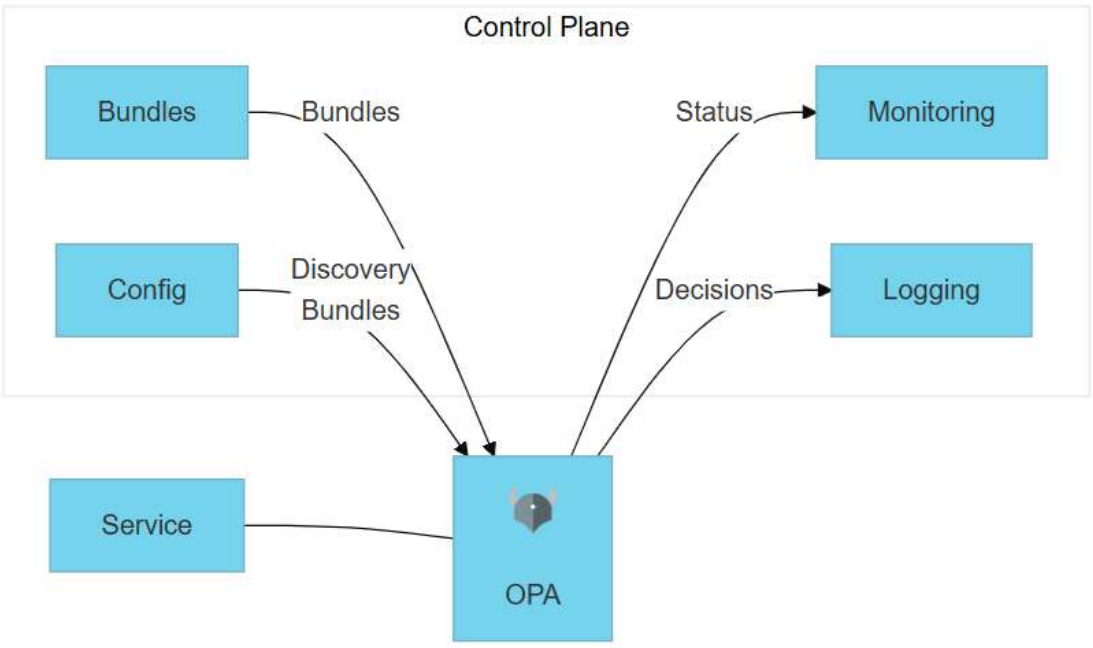

Figure 5.2. OPA management

5.4 Deployment

OPA serves as a general-purpose *PDP* that answers authorisation and compliance queries on behalf of applications, i.e. the *PEP*. In production, an OPA instance should respond with low

latency, refresh policies as they evolve, ingest live and external data necessary for evaluation, and produce auditable decision logs. As a design principle, it is recommended to locate OPA as near as possible to the PEP that calls it: the closer the PDP to the enforcement path, the lower the end-to-end latency and the higher the resilience.

That said, centralising OPA behind a network endpoint can be appropriate where state is very large, where compute is ephemeral (e.g. serverless), or where batch or CI workloads benefit from shared caches and simplified management. In short: co-location yields speed and fault tolerance at the cost of per-instance resources. Centralisation economises resources but introduces a network hop and requires careful high-availability engineering. The "right" model is therefore workload-dependent rather than absolute. Moreover, various deployment modes for OPA are available: Kubernetes, Docker, AWS, Google Cloud and Azure.

5.4.1 Deploying OPA with Docker

Docker provides a straightforward route to running OPA across environments. Official images are published at `openpolicyagent/opa`. By default the image starts the interactive `run` REPL, for deployments you will run OPA as an HTTP server.

Starting the server

Launch the agent with the `--server` flag and expose its listener. When running in a container, bind to all interfaces to permit connections from outside the container.

Listing 5.7. OPA as an HTTP server in Docker

```
docker run -p 8181:8181 openpolicyagent/opa
run --server --addr=0.0.0.0:8181 --log-level=info
```

Operational flags

The server accepts `--addr` (listener, default `localhost:8181`), `--log-level` (`debug|info|error`, default `info`) and `--log-format` (`json|json-pretty|text`, default `json`). Prefer structured JSON in production and reserve `debug` for development as it can emit request/response bodies.

Supplying policies and data

The container image is deliberately minimal, no policy or data are embedded. Is required to mount the required policy and data from the host and direct OPA to those mount points (e.g. `/policy`, `/data`). Evaluation may then be performed ad hoc, or the server may be started with these paths preloaded.

Listing 5.8. Evaluate a mounted policy

```
docker run -v "$PWD":/example openpolicyagent/opa
eval -d /example 'data.example.greeting'
```

```
package example

greeting := msg if {
info := \ac{OPA}.runtime()
host := info.env["HOSTNAME"] # set by Docker
msg := sprintf("hello from container %q!", [host])
}
```

To serve those policies over HTTP, it is needed to pass the directory to `run`:

```
docker run -p 8181:8181 -v "$PWD":/policies openpolicyagent/opa
run --server --addr=0.0.0.0:8181 /policies
```

Image tags

It is recommended to use explicit version tags (e.g. `1.0.0`) for reproducibility. The `X.Y.Z-dev` tracks development builds for a release line, while `edge` follows the main branch and is unsuitable for production.

Chapter 6

Post-Quantum IPsec Gateway: design

In view of the credible prospect of CRQCs, it is prudent to assume that, in the near to medium term, a sufficiently powerful quantum computer could compromise the most widely deployed public-key schemes *classical* (through the Shor algorithm) and materially weaken certain symmetric constructions (through Grover algorithm).

Let

$$Q := \text{time until a quantum computer capable of breaking current schemes exists,}$$

$$S := \text{time for which the protected data must remain confidential,}$$

$$U := \text{time to migrate to, validate, and deploy quantum-resistant cryptography.}$$

Mosca's rule states that if

$$U + S > Q$$

a substantial strategic problem emerges.

Since the design, standardisation, testing, integration, and *mass deployment* of new cryptographic mechanisms is inherently protracted, preparations should start *now*. This imperative is particularly pronounced for legacy systems with long upgrade cycles, or even without the option for updates due to computational limitations, as deferral increases the likelihood that the migration window U will exceed the remaining time $Q - S$.

The aim of this design is not merely to sketch a post-quantum IPsec approach, work already initiated by RFC-9242 and extended by RFC-9370, but to specify a concrete implementation that enables legacy services, which cannot yet perform a post-quantum transition, to interoperate with external post-quantum enabled services via a gateway acting as a *translator*. The architecture is deliberately *crypto-agile* and modular, providing fine grained, policy-driven control of inter-service security levels. It facilitates rapid reconfiguration in response to vulnerabilities in **newly standardised algorithms** and explicit management of **trust anchors** and **authorities** in line with known security and performance properties.

In effect, adjustments to the environment's security posture can be realised by updating a small policy module, without altering hard-coded gateway logic or restarting the system, even when the level of trust in a particular algorithm or authority must be revised at run time.

6.1 Architectures evaluation

A review of the literature and available commercial offerings for implementing post-quantum VPNs reveals a heterogeneous landscape: approaches target different layers of the stack, threat models and architectural patterns, reflecting varied deployment priorities and constraints.

6.1.1 A Minimalist Approach to Hybrid Key Exchange

Minimally Invasive Key Augmentation (MIKA) realises hybrid key exchange *without* altering existing protocol state machines. Rather than mix post-quantum and classical primitives within a single handshake, MIKA launches multiple, standard negotiations in parallel (e.g. IKEv2 with ECDH alongside IKEv2 with "opaque" extensions or TLS employing a post-quantum KEM). Each run proceeds independently to completion. Their shared secrets are then fused into a single session key via a KDF capable of absorbing auxiliary inputs. A lightweight controller orchestrates these concurrent runs, assigns a common session identifier, and hands the resulting secondary secrets to the primary data-channel protocol for binding into the final key. This decoupling confers strong crypto-agility (algorithms and even whole protocols can be exchanged without invasive changes), keeps the core implementation small, and exploits parallelism so that the end-to-end overhead is close to the sum of the constituent negotiations. Moreover, measurements on strongSwan indicate modest additional cost for two-protocol hybrids and good scaling as further secondaries are executed in parallel [55].

However, while MIKA is operationally efficient on *endpoints*, it is inappropriate to a gateway that translates between legacy and post-quantum enabled peers under fine-grained policy control:

- **asymmetric capabilities**: the minimalist design of MIKA assumes that both communicating peers can initiate one or more independent key exchange protocols in parallel and then locally combine the resulting secrets. In a gateway scenario, the device has to *terminate* a legacy IKEv2 exchange on one side and *originate* a post-quantum exchange on the other. There is no common controller on the endpoints that can run parallel negotiations. Consequently a translator must perform an in path transformation of the key material rather than deferring combination to the endpoints, which undermines the simplicity that makes MIKA attractive on end hosts.
- **per-flow policy binding and auditing**: a policy enabled gateway needs to enforce security requirements for each flow (choose algorithms, enforce minimum levels, deny downgrades, record decisions) and to expose these decisions for audit. In MIKA the "secondary" key exchanges occur out-of-band under the control of a lightweight controller on the endpoint, and there is no natural hook for a central policy engine to inspect or veto those runs. Attaching fine-grained policies, logging them and updating them at run time would therefore require invasive modifications to the controller and to the data plane, defeating the minimalist goal;
- **policy design**: the described abstraction does not straightforwardly map stable security levels into concrete operational decisions, determining *a priori* the resultant assurance when keying material is combined from parallel, heterogeneous protocol exchanges is non trivial;
- **operational complexity on the gateway**: the proposed solution gains efficiency by starting multiple key exchanges in parallel, but a gateway that bridges between different technologies would have to manage those extra control channels for every tunnel (additional TLS/IKE exchanges, session identifiers, retransmissions) while also dealing with NATs and firewalls.

6.2 IKE-less IPsec

Recent work on Software-defined Network (SDN)-driven IPsec management reframes key establishment as a controller service rather than a device capability. [56] distinguish an "IKE case", where endpoints (NSFs) run IKEv2 with locally managed SPD and SAD, from an "IKE-less case", where Network Security Functions (NSFs) expose only the IPsec datapath while the controller generates keys and SPIs, chooses algorithms, performs rekey before hard expiry, handles NAT traversal, and installs SPD and SAD entries proactively or reactively. The aim is to avoid unscalable manual IKE provisioning and to simplify constrained devices, accepting controller-side latency and scalability challenges as the natural cost of centralisation. The model has been standardised in RFC-9061 [57] via Interface to Network Security Function (I2NSF) YANG modules

covering IKE (when present) and the IKE-less mode, defining secure controller to NSF configuration and state for PAD,SPD, SAD and IKEv2 over NETCONF/RESTCONF, and analysing operational aspects as well as the trade-offs between the two cases. A recent work [58], targeting IoT and security within SDN, reiterates these points arguing that IKE-less simplifies low-end nodes and enables uniform policy rollout, provided the controller is hardened, by means of Trusted Execution Enviroment (TEE) and HSM, for SA and policy generation and carefully engineered for timing-sensitive rekeys and scale.

In the context of *quantum-safe state migration*, the controller-centric architecture offers a different opportunity in this field. In fact, introducing post-quantum KE into the controller's key management workflow enables uniform SA derivation and policy enforcement across heterogeneous dataplanes, without necessarily first requiring upgrades to every legacy IKE stack.

However, a *post-quantum translator gateway*, interposed between legacy IKEv2 peers and post-quantum ready counterparts, must also *enforce negotiation policy*, e.g. blocking non quantum safe transforms, pinning identities, constraining CHILD_SA creation and IKE rekey semantics, preventing downgrades. Indeed, the IKE-less datapath faces an intrinsic limitation: it never processes IKE messages, so it cannot natively inspect or veto proposal or authentication phases, and must rely on a controller that pushes SPD and SAD consistent with desired outcomes while reacting to notifications. The RFC-9061 explicitly contrasts this with the "IKE case", where endpoints authenticate peers and derive session keys locally, reducing reliance on controller timing and correctness. Consequently, for safe policy *interdiction* at negotiation boundaries, the translator needs either an IKE edge on the legacy side (with SPD control) paired with IKE-less distribution on the post-quantum side, or it must accept fragile coupling between external IKE middleboxing and controller-driven state, with attendant risks of latency, induced races and state drift.

One might consider Quantum Key Distribution (QKD) to "make IPsec post-quantum ready". This is one of Telefónica's research projects, which builds on the aforementioned SDN framework with a centralised controller and employs QKD to ensure a transition to a quantum-safe posture. However, while QKD establishes a shared key over a physical quantum channel, typically optical fibre, where eavesdropping perturbs quantum states and is, in principle, detectable, it still requires a classical *authenticated* side channel, reintroducing computational assumptions (or pre-shared authenticators) that post-quantum cryptography already addresses. Moreover, practical QKD faces distance and throughput limits (order 10–100 km and 0.1–1 Mbps). It also important to note that it will provide hop-by-hop rather than true end-to-end security, fitting better the site-to-site deployments (e.g. between data centres) rather than heterogeneous, wide-area overlays.

6.3 Industry solutions

Among industry offerings for building a post-quantum gateway, Palo Alto Networks documents two manufactured configurations on PAN-OS that are positioned as standards-based but delivered through a proprietary platform and management workflow.

The first, *"Post-Quantum IKEv2 VPNs with RFC-8784 PPKs"*, mixes out-of-band pre-shared key material (PPKs) with the classical Diffie-Hellman exchange to harden IKEv2 against HNDL threats while requiring no immediate change to the new digital signature algorithms on peers. It is analogous to Telefónica's programme: whereas Palo Alto's solution operates within IKEv2, Telefónica's adopts an IKE-less, centralised SDN controller framework with optional QKD for keying, providing a pragmatic mitigation of HNDL risks without fixing up endpoint stacks.

In contrast, the second workflow, *"Post-Quantum IKEv2 VPNs with RFC-9242 and RFC-9370 Hybrid Keys"*, enables multiple additional KEM rounds (up to seven Additional Key Exchanges per [12]) to form a hybrid shared secret that combines classical and post-quantum KEMs, for crypto-agility and resilience during the transition. However, Palo Alto's documentation states that, at present, authentication in these exchanges remains classical (PSK or traditional digital signatures) and therefore post-quantum signatures are still a future requirement. Moreover, it is important to note that adding multiple KEM rounds before the initiator is authenticated, as allowed during the IKE initial establishment, increases resource consumption and raises the risk of DoS during the extended IKE_INTERMEDIATE phase. In fact, large payloads can also introduce

fragmentation and latency, risks that a translator gateway must weigh carefully when enforcing negotiation policies.

6.4 The proposed solution

The next section sets out the proposed design, which employs a Docker-based deployment in which distinct containers cooperate to realise a policy-driven, post-quantum IPsec gateway.

6.4.1 Policy Enforcement Point

To realise a gateway that supports IPsec and works as a *translator*, terminating tunnels on both sides (legacy for the internal network and post-quantum capable for peers on the external side) and forwarding only policy approved flows between them, we deploy a Docker container running `strongSwan` v6.0.`beta`6. A stable 6.0.x release would be suitable for production. However, the chosen experimental build enables broader evaluation of multiple post-quantum KEMs and signature schemes during the transition, whereas the stable line concentrates only on the NIST's standardised key encapsulation algorithm, with exclusive emphasis on mitigation of HNDL threat. In addition, this experimental version permits activation of post-quantum features by just enabling the `oqs` plugin at configuration time, which integrates `liboqs` to provide quantum-safe algorithms and includes core support for `IKE_INTERMEDIATE` exchanges to realise Additional Key Exchanges.

The gateway is connected to three Docker networks: a legacy segment, an external segment, and an internal control-plane network dedicated to communication with the PDP. The traffic translation between these domains is achieved by enabling `ip.forwarding=1` and by configuring kernel `XFRM` state via `strongSwan`, in accordance with the *child* connection selectors.

The establishment of a CHILD SA installs the requisite SAD/SPD entries and `XFRM` policies for each tunnel. Forwarding follows a *double encryption* pattern: packets are encrypted under the first SA, decrypted at the gateway, routed internally, and then re-encrypted under the second SA before egress automation provided by kernel `XFRM` hooks within the routing datapath. Peers need only configure static routes so that traffic destined for internal legacy services is forwarded via the gateway's address.

To apply fine-grained control over external services attempting to reach internal legacy systems, and conversely, to ensure that outbound internal traffic is adequately protected, it is essential to delineate the phases of IKE negotiation and the subsequent creation of `CHILD SAs`. As already mentioned in chapter 3, IKEv2 principally comprises `IKE_SA_INIT` and `IKE_AUTH`. During `IKE_AUTH`, peers also exchange the SA and traffic selector proposals for the first child. While the `charon` daemon offers no native hook to halt processing during `IKE_SA_INIT`, it does permit gating *after* authentication and *before* installation of the IKE SA via the `ext-auth` plugin. This facility momentarily pauses `IKE_AUTH` prior to any SA being installed, invokes an external script, and awaits an `allow/deny` decision (and profile) from the PDP, thereby providing strong pre-install security control.

However, `charon` lacks a native mechanism to expose child negotiation parameters until the `CHILD_SA` is actually up. Thus, following a successful `IKE_AUTH`, the first child would otherwise be created automatically, precluding policy checks at that boundary. Nonetheless, a method could be devised to retrieve these parameters, yet doing so would incur additional delay before issuing a denial, thereby heightening the risk of DoS. Aside from this, conducting multiple ADDKE rounds during the initial unauthenticated KE phase again increases the risk of DoS, due to the heavy unauthenticated post-quantum workload. To address both issues, the gateway mandates *childless* IKE (RFC-6023 [52], with post-quantum generalisations in RFC-9370). In this mode, peers authenticate and complete the IKE SA *without* creating a child. The additional key exchanges (`IKE_FOLLOWUP_KE`) and the first `CHILD_SA` are deferred until *after* authentication, reducing exposure to unauthenticated computation and enabling strict policy gating before any child is installed.

Listing 6.1. IKE SA establishment (childless mode)

```
HDR(IKE_SA_INIT), SAi1(.. ADDKE*...), --->
 KEi(Curve25519), Ni, N(IKEV2_FRAG_SUPPORTED),
 N(INTERMEDIATE_EXCHANGE_SUPPORTED)
    Proposal #1
    Transform ECR (ID = ENCR_AES_GCM_16,
                   256-bit key)
    Transform PRF (ID = PRF_HMAC_SHA2_512)
    Transform KE (ID = Curve25519)
    Transform ADDKE1 (ID = PQ_KEM_1)
    Transform ADDKE1 (ID = PQ_KEM_2)
    Transform ADDKE1 (ID = NONE)
    Transform ADDKE2 (ID = PQ_KEM_3)
    Transform ADDKE2 (ID = PQ_KEM_4)
    Transform ADDKE2 (ID = NONE)
    Transform ADDKE3 (ID = PQ_KEM_5)
    Transform ADDKE3 (ID = PQ_KEM_6)
    Transform ADDKE3 (ID = NONE)
                  <--- HDR(IKE_SA_INIT), SAr1(.. ADDKE*...),
                       KEr(Curve25519), Nr, N(IKEV2_FRAG_SUPPORTED),
                       N(INTERMEDIATE_EXCHANGE_SUPPORTED)
                       Proposal #1
                         Transform ECR (ID = ENCR_AES_GCM_16,
                                        256-bit key)
                         Transform PRF (ID = PRF_HMAC_SHA2_512)
                         Transform KE (ID = Curve25519)
                         Transform ADDKE1 (ID = PQ_KEM_2)
                         Transform ADDKE2 (ID = NONE)
                         Transform ADDKE3 (ID = PQ_KEM_5)
 HDR(IKE_INTERMEDIATE), SK {KEi(1)(PQ_KEM_2)} -->
                  <--- HDR(IKE_INTERMEDIATE), SK {KEr(1)(PQ_KEM_2)}
 HDR(IKE_INTERMEDIATE), SK {KEi(2)(PQ_KEM_5)} -->
                  <--- HDR(IKE_INTERMEDIATE), SK {KEr(2)(PQ_KEM_5)}
 HDR(IKE_AUTH), SK{ IDi, AUTH } --->
                     <--- HDR(IKE_AUTH), SK{ IDr, AUTH }

          /* No SAi2/SAr2 or TSi/TSr are exchanged in childless mode */
```

Once authentication completes, `ext-auth` is invoked and calls an external script that assembles a structured JSON request for the PDP. This payload captures the role, peer address and identity, the negotiated IKE suite (including PRF, DH and any agreed ADDKE), together with certificate derived metadata (subject, issuer, signature and public-key algorithms). The PDP evaluates this context to determine whether the minimum policy level required for the target service has been met for both the IKE key exchange and the certificate's signing algorithms. Through an internal mapping, the policy engine also identifies which legacy service, or subnet, the external peer is permitted to reach (and vice versa).

When the PDP returns `allow`, it provides the minimum IKE level achieved, the certificate level achieved for the session and a complete child SA configuration describing traffic selectors, rekey time and ESP proposals (including ADDKE choices) in detail. The `ext-auth` script consumes this child template, converts it into `swanctl` format via an internal mapper, verifies consistency against the local `swanctl.conf` and then persists {`ike_unique_id`, `child_sa_config`} to the `ike-<unique-id>.json` file. In light of a positive decision, the script exits successfully, allowing the IKE negotiation to complete in *childless* mode. If the PDP returns `deny`, it supplies a reason, such as the required KE level or certificate algorithm thresholds, which is delivered to the peer via a vendor specific `Notify` alongside the standard `AUTH_FAILED`.

A background controller listens for `ike-up` events. Upon notification, it loads the corresponding state file, reads `ike-<unique-id>.json`, and installs the authorised child using VICI's *initiate*

child SA command. An ex-post verifier, then re-checks that the installed selectors and ESP parameters exactly match the approved child configuration received from OPA, any discrepancy triggers a controlled teardown of the child. The same gating pattern applies to rekeys: during make-before-break, the daemon negotiates a new IKE SA in parallel, submits both the incumbent and candidate suites to OPA, and promotes the new SA only if it satisfies policy minima. Otherwise, the existing SA remains active until a compliant rekey succeeds. Since rekeys do not reauthenticate peers, they can be used to *raise* the post-quantum strength of the IKE SA safely after authentication, avoiding the earlier risk of unauthenticated heavy exchanges.

6.4.2 Policy Decision Point

All decisions concerning IKE establishment, the enforcement of child parameters, and the minimum levels accepted during rekey are subjected to rigorous validation by OPA, which serves as the central PDP in our environment. To present these policy checks clearly, we first outline the `Rego` files employed at each validation stage, explaining how each artefact contributes to the overall verdict and how responsibilities are partitioned across modules to maximise code modularity and enable rapid changes.

IKE establishment gate

This module governs whether the gateway will accept an incoming or outgoing IKE_SA before any children are created. It specifies a set of security levels (`KE-L1` - `KE-L4`) that constrain the PRF, Diffie-Hellman group and, the post-quantum Key Encapsulation Mechanism. At this point, only a *single* KEM (`ke1`) is permitted, specifically the NIST standardised ML-KEM at the required security strength, while the auxiliary KEM slots `ke2` and `ke3` are disabled. By design, this phase forbids additional ADDKEs (up to seven are allowed by RFC-9370 during the initial IKE exchange), as previously said, to limit unauthenticated post-quantum computation and reduce the DoS attack surface, while multi-KEM suites are deferred to the child stage. The module maps peers to minimum `KE` levels (e.g. Bank A requires `KE-L3`) and resolves peer identities through a separated peer mapping `Rego` file. The main IKE policy file, enforces address based access lists, verifies that the connection is running in *childless* mode, ensures that the negotiated suite meets or exceeds the minimum level, and consults the specific module to check certificates metadata. Moreover, real-time Cyber Threat Intelligence (CTI) feeds are incorporated, allowing the gate to block traffic from known malicious. If all conditions pass, the module returns an *allow* decision together with the complete child SA template (selected again from a specific module dedicated to child SA parameters definition). Otherwise, it returns a securely framed explanation of the failure and, where appropriate, indicates the minimum level required for that specific IKE connection.

Certificate validation To augment IKE establishment checks, a certificate validator file is invoked to verify that the peer certificate, together to the previously negotiated parameters during IKE_SA_INIT, attests to a fully post-quantum channel and satisfies the level mandated for the specific external-to-internal service pairing. This file defines OIDs for the newly standardised ML-DSA and for composite hybrid signatures [30]. It maps signature and public key algorithms to a hierarchy of signature levels (`SIG-L1`, `SIG-L2`, ...) and specifies per-peer minimum requirements. For example, Bank A's certificates must have signature and public key strength at least `SIG-L3-SUF`, and must be issued by one of the permitted CAs. The module checks that the certificate has a valid composite or "pure" post-quantum signature and public-key, that the signature algorithm is at least as strong as the public-key algorithm, that the subject matches the expected peer, and that the issuer is trusted. It returns the certificate's public key level and a boolean `cert_allow` which is consulted by the IKE gate. By decoupling signature policy from key exchange policy, the design permits signature requirements to be tuned independently of the KE levels.

Service classification and peer mapping This helper module is the authoritative source for classifying internal subnets and external partners, and for deriving service-to-peer authorisation.

Rather than listing individual host IPs, it adopts a *subnet-oriented* model: any host within a designated range (e.g. `10.200.0.0/24` for legacy services) automatically inherits the relevant security profile, cryptographic capabilities, and permitted partner set for that class. External partners are described by subnet (or single IP), service type (e.g. payments, ERP, HR), and minimum cryptographic requirements (`min_ke_level`, `min_sig_level`, `min_pubkey_level`). Each entry also enumerates admissible CA issuers so that certificate checks can consistently validate subject names and trust anchors. For example, `banka` maps to `198.51.100.10/32`, is classified as a payments service, and must satisfy `KE-L3` for key exchange and `SIG-L2` for signatures.

Due to consolidating service and partner definitions in a single location, any changes to subnet ranges, partner permissions or minimum security levels are automatically reflected in dependent modules without requiring modifications to the enforcement logic. In practice, when network addressing changes or a particular pairing must be disabled, it is sufficient to update `service_classes`, thereby preserving maintainability, auditability and operational agility as threats and organisational policies evolve.

Child templates and levels Once an IKE_SA is authenticated, a childless handshake defers child SA creation until after policy evaluation. This file defines child security levels (`CHILD-L1` - `CHILD-L4`) and, for each level, enumerates ESP proposal strings that combine AEAD ciphers, DH groups, and a *full* multi-KEM suite. The policy is structured so that the cryptographic profile for a child, including the ADDKEs, is specified in a dedicated module, whereas responder/initiator templates that map peers and KE levels to concrete traffic selectors, rekey intervals, and an `updown` hook for ex-post verification are defined in a separate module within the same file. This separation keeps security parameters distinct from selector and lifecycle settings, maximises modularity, and preserves crypto-agility, to ensure that, updating a profile requires changes only to the relevant section. Helper functions fetch the appropriate template for a given role, peer and KE level and expose it as a structured configuration.

Threat intelligence data for augmented decisions The threat intelligence inputs are provided as a signed CTI bundle by a dedicated container. This service periodically retrieves sources (in this case lists of malicious IPs), packages them into an OPA bundle with a deterministic ETag, signs the bundle and exposes it via a token protected endpoint. At evaluation time, the CTI dataset is accessed via `data.cti.threats`. The IKE establishment policy consults the predicate `ip_is_malicious` to block connections originating from these addresses. Due to the fact that CTI content is decoupled from policy logic, new feeds can be integrated by updating the dataset rather than rewriting rules, preserving modularity and allowing operators to select their own trusted intelligence sources.

Ex-post child validation

After a child SA has been installed by the gateway, this module validates that the actual parameters match the template authorised by the IKE establishment policy. It checks that the child name corresponds to the expected profile, that the negotiated traffic selectors equal the expected selectors, and that the cryptographic algorithms satisfy the requirements of the declared child level. It also uses the same CTI feed to block child creation if the peer address becomes malicious. Reasons for denial are recorded internally indicating, "profile_mismatch" and/or "crypto_mismatch", providing clear telemetry to the decision logger.

IKE SA rekey gate

IKE rekeys do not require peers to reauthenticate, they are effected via `CREATE_CHILD_SA` without exchanging Traffic Selectors (TSs). Because authentication is not repeated, additional KEM rounds can be introduced at this stage without incurring the risk of heavy, unauthenticated fragmented messages. Accordingly, this module permits secondary KEMs (`ke2`, `ke3`) during rekey, provided the new suite meets the service's minimum requirements and does not *downgrade* the

existing IKE_SA. It imposes an ordering over PRFs, DH groups and KEMs, detects regressions (e.g. from ML-KEM-1024 to ML-KEM-512), and returns either `promote_new_delete_old` or `terminate_new_keep_old`. CTI checks are enforced at this stage as well. The rekey gate applies exclusively to the IKE_SA, rekeying the IKE SA legitimately renegotiates a fresh transform set, whereas (per RFC-7296) a Child_SA rekey is intended to be an *equivalent* replacement and therefore *SHOULD NOT* alter its TSs or algorithms.

Algorithm choices and NIST guidance

The policy's cryptographic settings reflect the state of the NIST post-quantum cryptography standardisation process.

IKE cryptographic suites The IKE establishment policy, restricts the unauthenticated handshake to a *single* KEM, namely the NIST standardised ML-KEM, in order to minimise pre-authentication computational load and bandwidth while relying on the only KEM formally standardised to date. This choice reflects NIST selection of ML-KEM as the first PQC KEM, from a cost/performance perspective, ML-KEM provides a favourable balance between encapsulation speed and ciphertext size at this stage of the exchange, at the same time mitigating the HNDL threat: delaying multi-KEM operations until after authentication reduces the potential for attackers to exhaust resources with heavy unauthenticated exchanges.

The configured KE levels are *conjunctive profiles* that align with NIST's security strength categories: a level is satisfied only when *all* prescribed primitives (AEAD, PRF, DH group(s), and `ke1` KEM) are jointly present. The design does *not* sum independent scores for each primitive. Instead, it mandates a singular, cohesive tuple that encapsulates the desired strength. This approach mirrors NIST's guidance that security strength is a property of the overall construction, benefits from algorithmic diversity and is not directly inferred from any one parameter such as key length alone [23]. The proposed policy adopts a *hybrid* stance, integrating classical and post-quantum components, to maintain a cautious approach as the PQC ecosystem evolves. By pairing a post-quantum KEM with a classical counterpart, the derived keys retain robustness even if one primitive is later weakened. Practically, the post-quantum KEM level is paired with an appropriate classical suite so that each acts as a backstop for the other, hedging against future cryptanalytic advances and reflecting the IKEv2 framework for multi-KEM (hybrid) exchanges. For example, ML-KEM-512 targets the 128-bit class (comparable to AES-128), hence its pairing with `aes128gcm16` at `KE-L1`, by contrast, although both `KE-L2` and `KE-L3` employ `mlkem768`, `KE-L3` hardens the classical tuple, upgrading to `aes256gcm16`, `sha512`, and mandating `ecp521`, thereby increasing cryptographic margin at the expense of greater computational cost, except for the PRF change, that is effectively cost-neutral on 64-bit platforms since `SHA-384` is a truncated variant of `SHA-512`.

Table 6.1. Configured IKE levels (KE–L1 to KE–L4): AEAD, PRF, DH groups and KEM slot

Level	AEAD	PRF	DH groups	`ke1` (ADDKE)
KE-L1	{aes128gcm16}	{sha256}	{ecp256, modp2048}	{mlkem512}
KE-L2	{aes192gcm16}	{sha384}	{ecp256, ecp384, x25519}	{mlkem768}
KE-L3	{aes256gcm16}	{sha512}	{ecp384, ecp521}	{mlkem768}
KE-L4	{aes256gcm16}	{sha512}	{ecp521}	{mlkem1024}

Certificate algorithms In aligning certificate validation with contemporary guidance, we adopt *security strength* as the organising principle rather than treating any single parameter as determinative. The idea was to establish a common scale for comparing primitives and suites, which we use to tier post-quantum signatures by the standardised ML-DSA parameter sets (`mldsa44`, `mldsa65`, `mldsa87`) and to map composites accordingly. In practice, ML-DSA L2/L3/L5 serve as reference anchors, composite schemes then combine ML-DSA with established RSA/ECDSA/EdDSA variants to achieve a target strength while accommodating interoperability and risk mitigation during transition. This design choice mirrors current LAMPS work on

composite signatures, where multiple algorithms are bound into a single atomic key and signature so that verification succeeds only if *all* components validate, thereby retaining security so long as at least one constituent remains robust.

The policy stratifies certificate requirements into three macro levels, each keyed to an ML-DSA parameter set. Within each strength tier, we further differentiate *composite* signatures by the assurance contributed by their classical component. In line with the LAMPS draft [30], selections using stronger classical primitives (e.g. ECDSA on higher-strength curves or RSA with larger moduli and modern padding schemes) populate "enhanced" sub-classes, while choices such as Ed25519 can offer additional assurance properties, notably *Strong Unforgeability under Chosen Message Attack (SUF-CMA)*. As suggested by the draft, the profiles are designed to be prioritised by application domain, for example: `id-MLDSA65-ECDSA-P256-SHA512` as a balanced default, `id-MLDSA65-RSA3072-PSS-SHA512` where RSA is required, `id-MLDSA44-ECDSA-P256-SHA256` or `id-MLDSA44-Ed25519-SHA512` for performance and bandwidth sensitive deployments, `id-MLDSA87-ECDSA-P384-SHA512` where PQC Level 5 is mandated, and `id-MLDSA65-Ed25519-SHA512` when the signature primitive must provide SUF-CMA assurance. This *profiling* approach explicitly counters the combinatorial explosion of composite options by encouraging domain specific, interoperable subsets rather than attempting to support the entire option space. Although `FALCON` offers compact keys and high throughput, its secure implementation remains intricate: floating-point arithmetic complicate constant-time design, and studies have demonstrated timing and power analysis leakages on common platforms. Consequently, we do not include `FALCON` among admissible options until a mature, demonstrably constant-time profile is broadly available, only when the planned *FN-DSA* standard FIPS 206 is finalised and implementation guidance stabilises, this decision can be revisited.

At the policy layer, service specific modules interpret security levels in context (enforcing stricter subclasses for payment flows or critical infrastructure), while an independent trust-anchor check verifies that issuers belong to the designated Certification Authority (CA) set. This separation of responsibilities permits signature policy to evolve independently of key exchange policy and streamlines operational changes (e.g. introducing a new composite scheme or deprecating a classical fallback) without requiring modifications to the enforcement code.

To prevent parsing inconsistencies across different stacks (such as strongSwan, OpenSSL and related toolchains), algorithms are matched using their *Object Identifiers (OIDs)* rather than human-readable names. The current implementation relies on the *provisional* OIDs defined in the aforementioned draft. However, these experimental identifiers must be replaced promptly once IANA assigns the final standard values, in order to avoid ambiguity during certificate validation.

Is important to note that, the effective security of a certificate chain is bounded by its weakest primitive. If a certificate is signed with an algorithm whose strength is lower than that of the subject's public-key algorithm, an attacker may forge the certificate (or an issuing link) despite the subject key itself remaining hard to break. In these cases, NIST explicitly cautions against *mixed-strength* suites and recommends selecting algorithms so that the *overall* protection meets the intended strength target. The IETF's algorithm profiles for PKI operations follow the same principle, requiring algorithms no weaker than those of the objects being protected. In post-quantum deployments the asymmetry is more acute: pairing a post-quantum secure public key with a classically secure only signature reintroduces an existential forgery path for quantum adversaries. Enforcing signature $\geq$ public-key thus preserves the soundness of the chain under both classical and quantum threat models.

Child templates The policy is intentionally split into two layers. First, the *security level catalogue* enumerates the admissible `ESP` proposal strings per level (`CHILD-L1-CHILD-L4`). Each proposal fixes the AEAD cipher, the (optional) DH group for *PFS*, and a *full* multi-KEM suite (`ke1`, `ke2`, `ke3`) used via `IKE_FOLLOWUP_KE`. Second, two *binding* modules (`responder_templates`, `initiator_templates`) map a concrete peer (e.g. `banka`, `partnerB` and `opsC` and the achieved `KE-L*` to a named child profile. These bindings only carry *deployment specifics* - `name`, `local_ts`, `remote_ts`, `reqid`, `start_action`, and the `updown` verifier, while the cryptographic proposals are referenced by level from the catalogue. This partitioning maximises modularity (cryptography and traffic scoping evolve independently) and enables rapid changes without touching enforcement code.

Table 6.2. Configured composite and pure ML-DSA signature suites with assigned levels

Suite	Assigned level
Pure ML-DSA	
mldsa44	SIG-L1
mldsa65	SIG-L2
mldsa87	SIG-L3
Composites	
mldsa44-rsa2048-pss-sha256	SIG-L1
mldsa44-rsa2048-pkcs15-sha256	SIG-L1
mldsa44-ecdsa-p256-sha256	SIG-L1
mldsa44-ed25519-sha512	SIG-L1-SUF
mldsa65-rsa3072-pss-sha512	SIG-L2
mldsa65-rsa3072-pkcs15-sha512	SIG-L2
mldsa65-rsa4096-pss-sha512	SIG-L2
mldsa65-rsa4096-pkcs15-sha512	SIG-L2
mldsa65-ecdsa-p256-sha512	SIG-L2+
mldsa65-ecdsa-p384-sha512	SIG-L2+
mldsa65-ecdsa-brainpoolp256r1-sha512	SIG-L2+
mldsa65-ed25519-sha512	SIG-L2-SUF
mldsa87-ecdsa-p384-sha512	SIG-L3
mldsa87-ecdsa-brainpoolp384r1-sha512	SIG-L3
mldsa87-ecdsa-p521-sha512	SIG-L3
mldsa87-rsa3072-pss-sha512	SIG-L3
mldsa87-rsa4096-pss-sha512	SIG-L3
mldsa87-ed448-shake256	SIG-L3-SUF

As introduced in section 6.4.1, child profiles are emitted in the same schema and naming style as `/etc/swanctl/swanctl.conf`. The gateway's `ext-auth` handler (upon *allow*) verifies that the *exact* child profile returned by OPA exists locally (same profile name, traffic selectors, and `ESP` suite). Only on an exact match is `vici initiate --child <name>` issued against the specific `IKE_SA`. While the `updown` hook performs an ex-post check to confirm that the installed selectors and cryptographic transforms are those mandated by policy, any discrepancy triggers a controlled teardown.

Within each child template, `local_ts` denotes the selector on the gateway side of that child, `remote_ts` denotes the peer side. Thus, for inbound service access (external to legacy), selectors constrain flows from the external peer to the designated internal host/subnet, for outbound cases, they constrain the inverse direction. In both cases, the establishment of a `CHILD_SA` installs the corresponding SAD/SPD entries and kernel `XFRM` policies.

The RFC-9370 allows multiple additional key exchanges and, in principle, peers could perform several (e.g. up to seven `ADDKE` transform slots). In this design, children are capped at *three* combined KEMs(`ke1`, `ke2`, `ke3`) to balance assurance and algorithmic diversity against operational cost: each added KEM grows message size and increases fragmentation and retransmission pressure, therefore it results fundamental to keep fragment counts modest and avoid advertising overly long algorithm lists.

The child-level catalogue pairs the standardised lattice-based ML-KEM as `ke1` with two additional KEMs (`ke2`, `ke3`) drawn from *distinct mathematical families* to avoid common mode failure. Our child-stage policy couples the standardised lattice family (ML-KEM at `ke1`) with a second, non-lattice KEM at `ke2`/`ke3` to avoid common-mode failure. The NIST's fourth round report formalises this diversification goal and records the decision to standardise *HQC* as the non-lattice companion to ML-KEM [43]. Moreover, in the comparative discussion, BIKE is acknowledged as having "the most competitive performance among the non-lattice-based KEMs" while HQC generally achieves *faster key generation and decapsulation*, at the cost of *larger public keys and ciphertexts*. The relative advantage can flip under adverse network conditions where BIKE's smaller

bandwidth footprint helps. This trade-off is reiterated indicating that HQC, requires only a fraction of BIKE's kilocycles for key generation and decapsulation at comparable security levels, but with larger on-the-wire sizes.

From an IPsec perspective, these characteristics translate into concrete operational pros and cons. The **HQC** algorithm offers lower CPU cost during frequent rekeys on busy dataplanes and benefits control-plane latency where compute dominates. However, as said, its drawback is greater message size, which increases fragmentation risk during `IKE_FOLLOWUP_KE` child establishment. **BIKE**, conversely, reduces bandwidth and fragmentation pressure thanks to more compact keys and ciphertexts, thus can be attractive on lossy or high-RTT paths. The costs are higher keygen/decap time and the need to manage *decryption failure rate* and related robustness issues that remain an area of scrutiny, even though the specification and implementations have adopted countermeasures (e.g. constant-time samplers and improved decoders). Our catalogue therefore treats *HQC* as the default non-lattice complement (performance-oriented rekeys, mature analysis) and *BIKE* as an optional alternative where bandwidth and fragmentation constraints dominate and in the case where its decryption failure rate posture is deemed acceptable.

On the other hand, **Classic McEliece** is deliberately excluded for our gateway's child proposals. While cryptanalytically conservative, its public keys are extremely large (hundreds of kilobytes to over a megabyte at higher categories). In an IKEv2 setting this burdens policy artefacts and control-plane signalling, provoking heavy fragmentation and larger state, even though ciphertexts are very small. Ultimately, it is essential to note that the policy remains modular, enabling authorised KEMs to be exchanged as standards and operational evidence progress, or to let users to incorporate their trusted algorithms without modifying the validator's basic logic.

Within a given child level the proposals are intentionally *not* identical: for example, `aes128gcm16-ecp256-ke1_mlkem512-ke2_none-ke3_HQC1` and `aes128gcm16-modp2048-ke1_mlkem512-ke2_BIKE1-ke3_HQC1` are both `CHILD-L1`. The former needs fewer ADDKEs because `ecp256` is stronger than `modp2048`, as stated in the RFC-9142 "security strength" tables [59], the latter compensates by employing two post-quantum KEMs to achieve comparable hybrid assurance. This pattern recurs across levels: where the classical group offers a smaller margin, the proposal uses more post-quantum exchanges. In all cases, however, the number of ADDKEs is still bounded.

Table 6.3. Child security levels and admitted `ESP` proposals

Child level	**Admitted** `esp_proposals`

`CHILD-L1`	`aes128gcm16-ecp256-ke1_mlkem512-ke2_none-ke3_HQC1`
	`aes128gcm16-ecp256-ke1_mlkem512-ke2_BIKE1-ke3_none`
	`aes128gcm16-modp2048-ke1_mlkem512-ke2_BIKE1-ke3_HQC1`

`CHILD-L2`	`aes192gcm16-ecp384-ke1_mlkem768-ke2_none-ke3_HQC3`
	`aes192gcm16-ecp384-ke1_mlkem768-ke2_BIKE3-ke3_none`
	`aes192gcm16-x25519-ke1_mlkem768-ke2_BIKE3-ke3_HQC3`

`CHILD-L3`	`aes256gcm16-ecp521-ke1_mlkem1024-ke2_none-ke3_HQC5`
	`aes256gcm16-ecp521-ke1_mlkem1024-ke2_BIKE5-ke3_none`
	`aes256gcm16-ecp384-ke1_mlkem1024-ke2_BIKE5-ke3_HQC5`

`CHILD-L4`	`aes256gcm16-ecp521-ke1_mlkem1024-ke2_BIKE5-ke3_HQC5`

6.4.3 Modularity and maintenance

Dependencies are imported *explicitly* to maintain clear separation of concerns: modifications to the classification, security levels and authorisation logic for internal subnets and external partners are localised within `service_classes`. Each policy module queries this file through well-defined helper functions, ensuring that changes to subnet definitions or partner authorisation rules propagate automatically without requiring modifications to the other modules. This architectural pattern ensures that policy evolution remains tractable, scalable and that cross-cutting concerns are managed centrally. The separation of concerns improves readability, simplifies unit testing, and enables rapid reaction to evolving standards or new CTI indicators. In addition, algorithm

catalogues and strength tiers are *centralised* as data tables, aligning the gateway with future NIST guidance typically requires editing those tables rather than altering decision logic, preserving policy reviewability. Operationally, OPA runs as its own container beside companion services (CTI feeder, decision logger, metrics stack) on an internal bridge network. This sidecar pattern is the recommended way to decouple policy evaluation from applications and to keep decisions low-latency and highly available. In particular, containerisation provides portability and process isolation. All the services in our design are situated within an isolated "bridge" network. A Docker *bridge* network is a software switch: containers attached to the *same* bridge can talk to each other, while traffic is isolated from containers on other bridges. By default, the bridge driver installs host firewall and NAT rules so that containers on *different* bridge networks cannot communicate unless a port is explicitly *published*.

In the proposed deployment model, OPA clearly separates the *decision* path from ancillary *management* activities (such as bundle distribution, status reporting and decision logging), enabling scalable distribution and observability across multiple components. This arrangement preserves centralised governance while still performing evaluations locally in memory. Policies and data can be updated *in place*, either by modifying Rego source files or by releasing signed bundles. More specifically, OPA retrieves, validates and activates new bundles *without requiring a restart*, and emits decision events that can be consumed for auditing and analytics. In combination with container-mounted volumes, this permits targeted changes to shared Rego tables (e.g. algorithm catalogues or strength tiers) or the roll-out of signed bundles, without necessitating gateway rebuilds or redeployments.

6.4.4 Decision logger

All policy decisions and data queries are forwarded to an independent *decision-logger* service. Every entry is timestamped in *UTC* using ISO 8601 to guarantee unambiguous time correlation across systems. The logger performs three functions: **normalisation** (canonical keys, types and ordering), **redaction** of sensitive fields (tokens, secrets), and **derivation** of user-friendly attributes (e.g. allow/deny outcome, negotiated level, deny reason, service labels). For auditability and triage it maintains *separate daily files*: a full decision stream (`decisions-YYYY-MM-DD.log`), a compact audit ledger (`audit-YYYY-MM-DD.log`, pipe delimited for quick scanning), and an errors only channel (`errors-YYYY-MM-DD.log`). Due to the fact that, OPA forwards *all* decisions, the logger can emit Prometheus compatible counters that OPA does not provide natively (OPA native metrics focuses on CPU, heap and timings). Specifically, the `/metrics` endpoint exposes totals and rates for IKE and Child events by level, rekey outcomes (success,fail or downgrade) and unique endpoints.

6.4.5 Metrics and dashboards

Prometheus scrapes the decision-logger's `/metrics` alongside OPA runtime endpoints, storing time-series for query and alerting [60]. Grafana [61] renders two complementary views: an *operations* dashboard (CPU, heap, evaluation latency) from OPA's own metrics and a *security posture* dashboard from decision-logger metrics (allow/deny volumes, IKE/Child level distributions, rekey success and downgrade attempts, CTI feed health such as bundle revision and uptime). Alert rules can flag, unusually high `deny` rates in a time window, a sustained drop to low security levels or recurrent rekey downgrades. The result is an auditable, centrally enforced pipeline with clear separation of concerns: OPA evaluates, the decision logger explains and quantifies, while Prometheus and Grafana deliver real-time transparency for post-quantum era risk, rapid change and compliance.

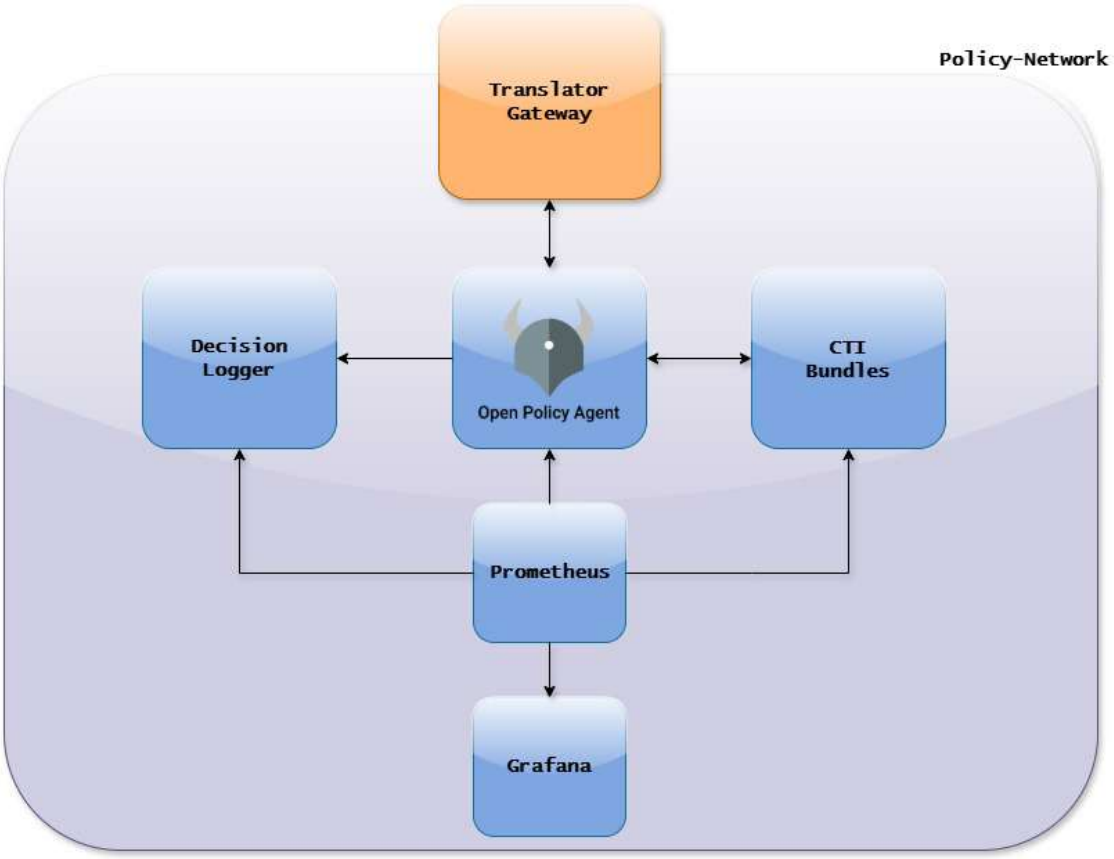

Figure 6.1. The Crypto-Agile Post-Quantum Gateway architecture

Chapter 7

Post-Quantum IPsec Gateway: implementation

In this chapter we set out the implementation steps and the modifications applied to make the design concrete. The proof-of-concept is delivered as a *containerised* environment orchestrated via Docker Compose. Each logical component: gateway, PDP, CTI feeder, decision logger, Prometheus and Grafana, runs in its own container and is attached to one or more virtual bridges. This approach mirrors an enterprise edge and provides clean separation between a public domain, a legacy LAN and a private policy plane. Within this sandbox, the gateway mediates between classical IKEv2 peers on the inside and post-quantum capable peers on the outside, while an Open Policy Agent (OPA) container acts as the PDP and exposes management APIs for bundle distribution and decision logging. Prometheus and Grafana collect and visualise operational metrics, and a CTI service produces signed threat intelligence bundles that are ingested by OPA. By encapsulating each component in its own Docker image and connecting them via named bridges, we achieve reproducibility, isolation and the ability to apply security hardening, by dropping all capabilities and mounting read only volumes, consistently across the stack.

More precisely, we discuss:

- the strongSwan changes, including daemon and file level modifications and the `ext-auth` plugin script;
- the implementation of PDP policies and the associated validation logic;
- the internal policy network for policy augmentation and the build of the monitoring services.

7.1 Network topology

The deployment mirrors an enterprise edge with a three-segment topology that cleanly separates concerns: an external partner domain, a legacy LAN, and a private policy plane. The gateway attaches to *three* Docker bridges, a`wan-net` (public `203.0.113.2`), a `lan-net` (internal `10.200.0.2`) and the `opa-network` (control). An *edge router*, realised as a lightweight Alpine container, links the partner segment `external-net` (`198.51.100.0/28`, router `198.51.100.2`) to the gateway's WAN via `203.0.113.3`, emulating routed external connectivity. It simply routes traffic between the partner subnet and the WAN network. This minimalist router suffices to emulate an enterprise border without conflating routing and policy functions. Peers install static routes towards the gateway and the legacy LAN via `198.51.100.2`. The gateway exposes a single public address and routes the partner block via the edge router. Internal hosts, situated on a `/24` segment derived from the `/16` legacy address space, access the gateway at `10.200.0.2`, with transit enforced by IPsec decapsulation/encapsulation and kernel forwarding. In the proposed demonstration, the CHILD_SA traffic selectors are advertised as `10.200.0.0/24` $\rightarrow$ X`/32` (e.g. `198.51.100.10/32`), thereby permitting any host in the legacy `/24` to reach the designated external peer X through

the gateway. Finally, OPA, the decision-logger, the CTI feeder, Prometheus and Grafana reside on the private `opa-network`. The gateway consults this control enclave for SAs establishment, rekeys and ex-post verification only.

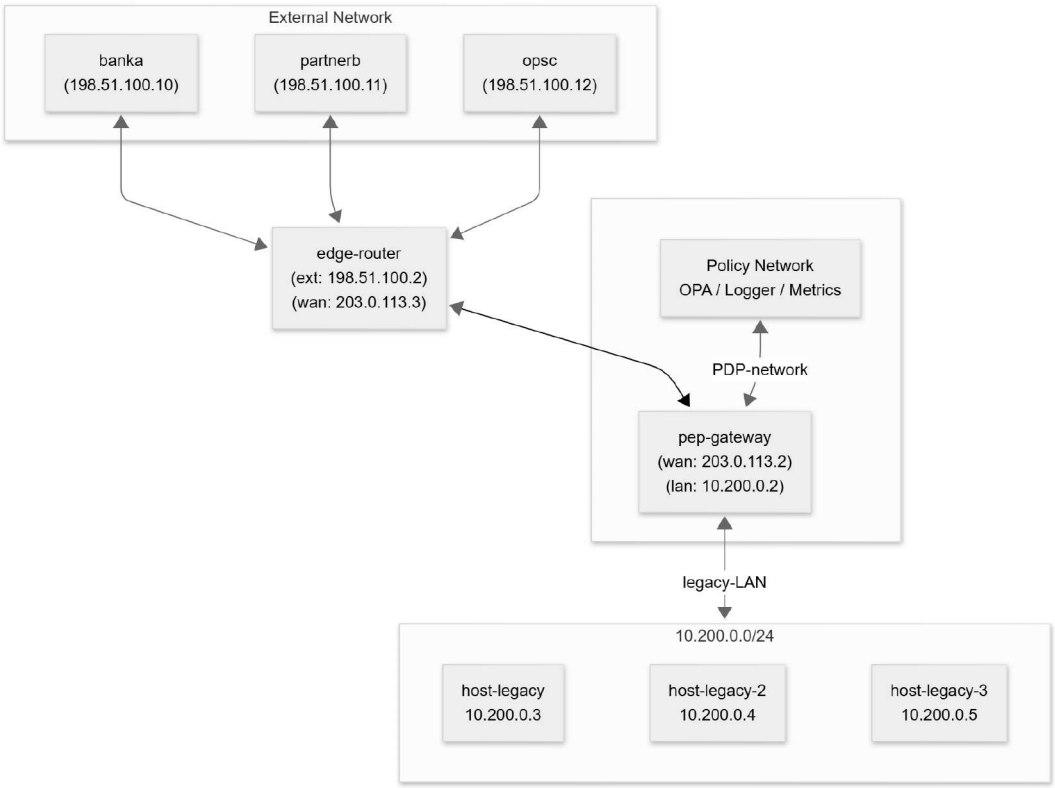

Figure 7.1. Network topology

7.2 StrongSwan modifications

The gateway container is based on `strongSwan 6.0.beta6`, an experimental build that ships the `oqs` plugin. The image is assembled in a multi-stage `Dockerfile`: a builder stage uses Ubuntu 22.04 to compile `liboqs` (release 0.10.0) with both static and shared libraries, followed optionally by the OQS provider for OpenSSL [62]. It then clones strongSwan at the `6.0.0beta6` tag, applies a custom patch (`ext-auth-runtime-cert.patch`) and configures the build with `--enable-openssl`, `--enable-oqs` and `--enable-ext-auth`. The patch extends the `ext-auth` plugin to export the peer certificate via `AUTH_RULE_SUBJECT_CERT`, add a vendor-specific notification (`NOTIFY_REQUIRED_LEVEL`) and capture hints from the external script. The final runtime stage installs the compiled strongSwan binaries, `liboqs`, OpenSSL configuration for the OQS provider, and Python 3. It copies custom orchestration scripts, config templates and a start-up script into the container. The beta image enables extensive evaluation of multiple post-quantum KEMs and signature schemes alongside the classical stack. Enabling the `oqs` plugin simply requires loading it in `strongswan.conf`. Beyond plugin activation, the container modifies the kernel runtime to permit forwarding (`net.ipv4.ip_forward=1`) and to disable source-route and redirect handling. The strongSwan configuration files (`swanctl.conf`, `strongswan.conf` and plugin snippets in `strongswan.d`) are mounted read-only from the host, rendering the IPsec stack declarative and eliminating configuration drift.

PKI and certificate management Shared root Certification Authoritys, one for legacy and one for external peers, are created outside the container environment and then imported into each strongSwan instance. On the gateway, both a legacy RSA certificate (used for the classical tunnel) and a post-quantum certificate (e.g. based on ML-DSA) are issued by the respective roots and

placed under `swanctl/x509`, while the associated private keys are stored in `swanctl/private`. Partner containers are provisioned in an analogous way, with certificates tailored to their specific role. Since the patch makes the peer certificate available to the `ext-auth` script, the script can apply policy checks on the signature and public-key algorithms at runtime without having to trust the certificate chain implicitly. The underlying cryptographic validation (e.g. signature verification and expiry checks) remains the responsibility of strongSwan.

The gateway is connected to three separate networks: a legacy LAN (`10.200.0.0/16`) via interface `eth1`, a public WAN segment (`203.0.113.0/29`) via `eth2`, and a dedicated policy bridge (`opa-network`) used exclusively for control-plane traffic between OPA, CTI and monitoring services. Static routes inside the container ensure that traffic targeting `198.51.100.0/28` (external partners) is forwarded to the edge router at `203.0.113.3`, and that legacy hosts direct outbound traffic via the gateway. Moreover, the container entrypoint script configures these routes and then executes `/opt/gateway/bin/startup.sh`, which launches `charon` along with the auxiliary daemons described below.

The implementation comprises several scripts beyond the stock strongSwan daemons:

- `opa-auth-check.py`: the ext-auth hook invoked by `charon` to gate IKE establishment.
- `vici-child-manager.py`: an event-driven controller that installs authorised CHILD SAs and enforces rekey/downgrade rules.
- `vici-suite-publisher.py`: a background worker that extracts negotiated IKE suites via VICI or from `charon.log` and caches them for the ext-auth script.
- `vici-tunnel-provisioner.py`: a monitor that observes legacy traffic patterns and, upon authorisation from OPA, provisions outbound tunnels to post-quantum safe partners.
- `crypto_mapper.py`: a utility library that maps algorithm names between OPA's policy vocabulary and strongSwan's internal identifiers and validates that child configurations match the local `swanctl.conf`.

7.2.1 `ext-auth` plugin script: `opa-auth-check.py`

The `ext-auth` plugin is patched to hand control to `opa-auth-check.py` during the `IKE_AUTH` exchange. Execution occurs after authentication but before any IPsec SA is installed. In addition to the standard environment variables provided by strongSwan, the patch adds facilities to capture the peer's certificate and the complete negotiated suite.

Dynamic certificate parsing The patched listener uses the `AUTH_RULE_SUBJECT_CERT` hook to obtain the remote end-entity certificate from the authentication rule and writes it to a temporary file. It then pushes an extra environment variables into the hook, the `PLUTO_PEER_CERT_PEM` (PEM encoded certificate). This enables the Python script to parse subject and issuer fields and to extract signature and public-key OID algorithms at run time without embedding certificate parsing logic inside strongSwan or depending on preloaded certificates within configuration files.

Suite extraction and normalisation Since the negotiated AEAD, PRF, DH group and ADDKEs are not directly visible to `ext-auth`, an auxiliary daemon (`vici-suite-publisher.py`) subscribes to IKE events and records the selected suite in a cache under `/run/opa-ike/`. The `ext-auth` script then reads this cache together with relevant environment variables and normalises algorithm identifiers using `crypto_mapper.py` (e.g. mapping `prf_hmac_sha2_512` to `sha512` and `ecp256` to `p_256`). Finally, it constructs a JSON payload containing the negotiated suite, peer identity, certificate metadata and connection name, and submits it to the PDP at `/v1/data/ike/establishment/decision`.

OPA decision and child configuration The PDP evaluates whether the proposed IKE SA meets the minimum levels for the service and, if so, returns a child template defining the traffic selectors, rekey time and a multi-KEM ESP suite. The script converts this template into `swanctl.conf` syntax and validates it against the locally installed configuration to ensure that the gateway will install exactly the authorised child. It persists the decision and template in `/run/opa-ike/ike-`⟨*id*⟩`.json`. If OPA denies the request because the key exchange or certificate algorithms are insufficient, the script triggers an informative failure.

Vendor-specific notifier for minimum level The patch also introduces a new vendor-specific notification type `NOTIFY_REQUIRED_LEVEL` (`0xA001`). When OPA returns `deny`, the script prints a line of the form `OPA_HINT unique_id=... required_ke=...` on `stdout`. The patched listener parses this hint, stores the required level in an internal structure and marks a pending notification. When `charon` constructs the `IKE_AUTH` response in case of a deny, the listener attaches the `NOTIFY_REQUIRED_LEVEL` payload with the required level to the outgoing message. Thus the peer receives both the standard `AUTH_FAILED` and the implemented vendor-specific notify indicating the minimum acceptable KE/signature level. This is conveyed without modifying the IKEv2 protocol itself. On receipt of such a notification from a peer, the listener stores the peer's requirement and logs it, enabling future negotiations to satisfy the advertised level.

7.2.2 Child manager: `vici-child-manager.py`

The child manager is implemented in Python and as an event driven service that maintains tight coupling between strongSwan and the PDP throughout the SA lifecycle. The daemon subscribes to `charon` events via VICI and reacts to three conditions: new IKE SAs (`ike-up`), child SA creation (`child-up`) and IKE rekeys (`ike-up` with rekey flag). Its core responsibilities are to install only the authorised child SA for each IKE, to enforce the one child per IKE invariant, to gate rekeys via OPA, and to perform `ex-post` verification of installed children via an `updown` hook. It uses non-blocking VICI event subscriptions and periodic polling to avoid deadlocks, ensuring that `rekey` and `updown` events do not interleave incorrectly.

When a new IKE SA is reported, the manager retrieves the corresponding state file (`ike-<`*id*`>.json`) generated by `opa-auth-check.py`. This file contains the OPA verdict and, if access is granted, the full CHILD SA configuration (`child_sa_config`). The manager then instantiates the authorised child via VICI, invoking `swanctl --initiate --child <child_name>`. Once the tunnel is established, it installs the approved CHILD SA and removes any unsolicited children that may have been created by the peer. All operations are recorded as structured JSON logs, and internal caches are updated to avoid duplicate installations.

For the rekey gate, the daemon employs two complementary mechanisms to govern rekeys. It subscribes to IKE rekey events and, whenever a new IKE SA is negotiated with the same remote endpoint and connection name, the manager extracts the negotiated cryptographic suites for both the old and new IKE associations, encapsulates the transformation sets into a JSON object, and submits it to `/v1/data/ike/rekey/decision`. The PDP responds with a decision containing an `allow` flag and an `action` (e.g. `promote_new_delete_old`, `terminate_new_keep_old`). If promotion is approved, the manager marks the new IKE SA as "adopted", copies all CHILD state files from the old to the new association, deletes the old IKE SA, and logs the result. Otherwise, it terminates the new IKE SA.

To guard against state drift and misconfiguration, every CHILD installation triggers an `updown` hook (`updown-verifier.py`). This script queries the effective CHILD parameters via VICI (including traffic selectors, AEAD algorithm, DH group and KEM slots) and submits them to `/v1/data/ike/child_create/decision`. Then, OPA then verifies that the installed CHILD SA conforms to the authorised template, if any field diverges, it returns `deny`. Upon such a denial, the manager tears down the CHILD SA and records the policy violation.

7.2.3 Tunnel provisioner

Dynamic outbound provisioning is implemented in `vici-tunnel-provisioner.py`. This process scans the installed child SAs to detect patterns indicating that a legacy host (`10.200.0.X`) is trying to communicate with an external partner (`198.51.100.X/32`). When a new pattern is found, the provisioner constructs a JSON payload with `legacy_host_ip` and `external_partner_ip` and posts it to OPA via `/v1/data/ike/tunnel_provisioning/decision`. If OPA returns `allow` with the name of an outbound post-quantum connection required to contact the external peer via the external interface of the gateway (e.g. `gw-wan-out-bankA`), the script initiates the corresponding childless IKE connection via `swanctl --initiate` and records the provisioning state, subsequently, all the previously specified IKE gating flow applies. This dynamic mapping allows flows from new legacy hosts or new partners to be provisioned without manual configuration. The provisioner runs in a loop, rescans theSA list at configurable intervals, and continues until terminated.

7.2.4 Legacy subnet implementation

The gateway's LAN interface (`10.200.0.2/16`) terminates a legacy subnet, within which a `/24` segment (`10.200.0.0/24`) is populated with representative hosts for the testbed. The traffic selectors in the CHILD SA templates are expressed at subnet granularity (e.g. `local_ts=10.200.0.0/24`), allowing any host in the legacy slice to reach a designated external peer via the gateway. These selectors are enforced through kernel `XFRM` state so that outbound packets from the legacy subnet are encrypted towards the gateway, decapsulated, forwarded internally and re-encapsulated towards the external peer, and similarly processed in the reverse direction. The `swanctl` configuration assigns distinct `reqid` values to inbound and outbound legacy children, keeping state separated for each internal network segment. Moreover, static routes on the legacy hosts direct traffic destined for the external subnet through the gateway, ensuring that all flows traverse the policy-enforced IPsec tunnel.

7.2.5 External post-quantum safe peers

External partners are modelled as individual containers on an `external-net` bridge (`198.51.100.0/28`) and connect to the gateway through an edge router. Each partner has its own strongSwan instance configured with post-quantum ready IKE proposals using the standardised ML-KEM and optional BIKE/HQC KEMs for Child SAs. The strongSwan patch is also included, on the external peer side, to elucidate the additional failure `Notify` in the event of a denial of IKE establishment, caused by a low security level. This patch enables recording of the received notifier payload containing the required KE or `SIG` level for that connection. Static routes on the partners point `10.200.0.0/24` and the WAN network towards the edge router, directing traffic to the gateway. The gateway's WAN interface advertises a single public IP (`203.0.113.2`) and uses the edge router (`203.0.113.3`) as the next hop for the external subnet.

7.3 The PDP implementation

The PDP is deployed on a dedicated Docker bridge network (`opa-network`), such that only containers attached to this bridge are able to communicate with one another. By default, the Docker bridge driver installs `iptables` rules on the host to isolate traffic between bridges, ensuring that containers on different bridges remain separated and can interact only via explicitly exposed ports. In this specific setup, `opa-network` acts as a private control plane: external peers cannot reach the gateway on this network, and policy evaluation traffic remains fully segregated from the data plane.

7.3.1 Policies structure

The PDP container is assembled through a deliberately lightweight multi-stage build. The first stage simply extracts the static upstream OPA binary (`openpolicyagent/opa:0.70.0-static`), ensuring that only the executable required for policy evaluation and bundle verification is retained, without introducing compilers, build tools or auxiliary packages. The second stage constructs the runtime image on top of a minimal `alpine:3.19` base. Alpine is chosen for its small footprint and reduced attack surface, providing only the essential runtime environment. A limited set of dependencies (`ca-certificates` and `tzdata`) is added, and all package caches are removed to keep the image compact. The static OPA binary, unified configuration file, and policy directory are copied into the container and assigned to an unprivileged user (`UID 1000, GID 1000`), under which the PDP exclusively operates. At runtime, all capabilities are dropped via `cap_drop:{ALL}` and `no-new-privileges:true` is enforced, with the sole exception of a few ephemeral paths explicitly designated as writable. These writable areas are isolated via `tmpfs` and carry restrictive flags, ensuring that no persistent or executable content can be introduced at runtime, and preventing any form of escalation or lateral movement even in the presence of a compromise. The final image exposes only the `8181/tcp` API endpoint and starts OPA in server mode using the unified configuration, resulting in a hardened and fully reproducible policy engine.

Policies are modularised in several `Rego` modules.

7.3.2 `service_classes.rego`

The `service_classes.rego` file acts as the central classification and routing oracle for the PDP. It defines the authoritative mapping between internal legacy subnets, external partners, service types and minimum cryptographic requirements. Rather than enumerating individual hosts, the policy relies on subnet-level classification, ensuring scalability and consistency across the gateway's decision pipeline.

Internal subnet and partner classification

Legacy networks are expressed as subnet classes (e.g. `10.200.0.0/24`), each associated with a security profile and the list of authorised external partners such as `bankA`, `partnerB` or `opsC`. External entities are likewise defined as partner classes indexed by their prefixes (e.g. `198.51.100.10/32`), with each class specifying the *service type* (payments, ERP, operational control) and the minimum acceptable security levels for key exchange, signatures and public keys. By centralising classification logic in a single module, policy updates such as extending subnet ranges, modifying trust relationships, or raising minimum cryptographic thresholds require changes in only one place, while the rest of the authorisation pipeline adapts automatically. This design ensures scalability, auditability and rapid policy evolution while preserving strict correctness guarantees.

Listing 7.1. Single control panel by service_classes.rego

```
internal_service_classes := {
    "legacy_internal": {
        "subnet": "10.200.0.0/24",
        "security_profile": "legacy",
        "crypto_capabilities": ["RSA-2048", "ECDH-P256", "ECDSA-P256"],
        "description": "Internal legacy hosts using traditional cryptography",
        "allowed_external_partners": ["banka", "partnerB", "opsC"]
    },

    ...
}

external_partner_classes := {
```

```
    "banka": {
        "subnets": ["198.51.100.10/32"],
        "service_type": "payments",
        "min_ke_level": "KE-L3",
        "min_sig_level": "SIG-L2",
        "min_pubkey_level": "SIG-L2",
        "allowed_issuers": {
            "C=CH, O=Cyber, CN=Cyber Root CA",
            "O=PQ-Gateway Lab, CN=PQ-Gateway IKE CA PQ",
            "O=PQ-Gateway Lab, CN=PQ-Gateway Root CA PQ"
        },
        "description": "Banking partner requiring highest security"
    },
    "partnerB": {
                    ...
```

Moreover, to support dynamic tunnel instantiation in che of an outbound connection, the module embeds explicit mappings between IKE connection names and the internal subnet that each connection serves.

Listing 7.2. Connection-to-subnet mappings

```
# Outbound connections initiated by the gateway
outbound_connection_subnet_mapping := {
    "gw-wan-out-bankA": "10.200.0.0/24",
    "gw-wan-out-partnerB": "10.200.0.0/24",
    "gw-wan-out-opsC": "10.200.0.0/24",
}

# Inbound authentication connections (responder mode)
inbound_connection_subnet_mapping := {
    "gateway-legacy-auth": "10.200.0.0/24"
}
```

These mappings enable *just-in-time tunnel provisioning*: when the tunnel provisioner observes a legacy host contacting an external partner, it queries this module to identify the correct internal subnet and trigger the creation of the appropriate post-quantum ready outbound tunnel.

7.3.3 `ike_establishment.rego`

This policy gate evaluates incoming IKE_SA requests. It checks that the negotiated AEAD, PRF, DH group and ADDKE combination meets the minimum required level for the service, that the peer's certificate is issued by a trusted CA, that respect the signature level in order to establish a full post-quantum safe tunnel, and that the subject matches the expected peer, to avoid spoofing. If all checks pass, it returns `allow=true` and the child configuration template to enforce. Otherwise, it returns `allow=false` along with a denial explanation, particularly indicating that a denial due to insufficient security during the IKE setup specifies the requisite KE and SIG levels.

A skeleton of the decision logic is:

Listing 7.3. Skeleton of ike_establishment.rego

```
package ike.establishment

# Import of the others Rego modules
import rego.v1
import data.ike.service_classes
```

```
import data.ike.child_templates
import data.ike.certificates
import data.cti.threats

...

# Example suite -> level mapping (simplified)
ke_suites := {
  "aes128gcm16-sha256-ecp256-mlkem512-none-none": "KE-L1",
  "aes192gcm16-sha384-x25519-mlkem768-none-none": "KE-L2",
  "aes256gcm16-sha512-ecp384-mlkem768-none-none": "KE-L3",
  "aes256gcm16-sha512-ecp521-mlkem1024-none-none": "KE-L4",
}

...

# Derive achieved KE level from suite; fall back to KE-L0 if unknown
ke_achieved := level {
  suite := build_suite_string(input.ike)
  level := ke_suites[suite]
} else := "KE-L0"

...

# Map IKE connection name to the internal subnet being protected
internal_subnet := subnet {
  subnet := service_classes.get_subnet_for_connection(input.connection_name)
}

# External IP: peer address classified as partner
external_ip := ip {
  ip := input.peer.addr
  service_classes.classify_external_ip(ip)
}

# Required KE level for this (internal subnet, external partner) pair
req_ke_level := level {
  level := service_classes.get_required_ke_level(internal_subnet, external_ip)
}

# Service type used later to select child template and metrics labels
service_type := service_classes.get_service_type(external_ip) {
  external_ip
}

# Partner must be allowed for the resolved internal service class
peer_ok {
  internal_subnet
  external_ip
  partner_name := service_classes.classify_external_ip(external_ip)
  some class_name, class_def in service_classes.internal_service_classes
  class_def.subnet == internal_subnet
  partner_name in class_def.allowed_external_partners
}

...
```

```
# CTI integration: block malicious or TOR endpoints early
cti_blocked {
  external_ip
  ip_is_malicious(external_ip)
} else {
  external_ip
  ip_is_tor(external_ip)
}

ip_is_malicious(ip) {
  ip in threats.malicious_ips
} else {
  some subnet in threats.malicious_ips
  net.cidr_contains(subnet, ip)
}

ip_is_tor(ip) {
  ip in threats.tor_exit_nodes
}

...

# Certificate validation via dedicated module
cert_ok {
  input.certificate
  certificates.cert_allow
}

cert_level := certificates.cert_level {
  input.certificate
} else := "none"

...

# Crypto validation: achieved suite level vs required policy level
crypto_ok {
  req_ke_level
  ke_level_order[ke_achieved] >= ke_level_order[req_ke_level]
}

...

# Child SA template resolution (wildcard TS for legacy subnet)
child_template := child_templates.get_child_config_wildcard(
  input.role,
  internal_subnet,
  external_ip,
  req_ke_level,
) {
  internal_subnet
  external_ip
}

...

# Decision object returned to the gateway
decision := {
```

```
  "allow": allow,
  "reason": reason,
  "ke_level": ke_achieved,
  "ke_required": req_ke_level,
  "cert_level": cert_level,
  "service_type": service_type,
  "child_sa_config": child_template,
}
```

7.3.4 child_templates.rego

Once the policy engine confirms that the negotiated key exchange and signature levels satisfy the minimum requirements for the connection, this module specifies the corresponding ESP proposals, covering the AEAD cipher, the DH group, and the full set of ADDKE parameters, associated with each security tier. It then binds individual service classes to their required levels so that a complete, level appropriate child SA template can be constructed. The resulting structure mirrors the layout used in the `swanctl.conf` configuration, enabling a precise, one-to-one comparison during validation. This ensures that the gateway installs exclusively the authorised and policy-compliant child SA, preventing any deviation from the intended cryptographic or traffic selector profile.

Listing 7.4. child_templates.rego snippets

```
...

child_security_levels := {
    "CHILD-L1": {
        "esp_proposals": [
            "aes128gcm16-ecp256-ke1_mlkem512-ke2_none-ke3_hqc1",
            "aes128gcm16-ecp256-ke1_mlkem512-ke2_bike1-ke3_none",
            "aes128gcm16-modp2048-ke1_mlkem512-ke2_bike1-ke3_hqc1"
        ],
    },
    ...

_templates := {
    "banka": {
        "KE-L3": {
            "name": "inbound-legacy-pay-L3",
            "local_ts": "10.200.0.0/24",
            "remote_ts": "198.51.100.10/32",
            "child_level": "CHILD-L3",
            "esp_proposals": child_security_levels["CHILD-L3"].esp_proposals,
            "rekey_time": "90s",
            "reqid": 101,
            "start_action": "none",
            "updown": "/usr/local/sbin/updown-verifier.sh",
            "description": "Banka inbound payments tunnel (L3 security)"
        },
        ...
```

The separation between *level definitions* and *mapping* enables to update the proposals or add a new algorithm family requiring the edit of only that specific part of the file.

7.3.5 Auxiliary validation modules

The remaining policy modules, `certificate_validation.rego`, `child_create.rego` and `rekey_ike_sa.rego`, provide the complementary validation stages required to complete the PDP's enforcement pipeline. Their conceptual role and decision flow have already been outlined in section 6.4.2, and their implementation follows the same `Rego` based principles illustrated in the earlier modules.

The `certificate_validation.rego` module applies end-entity checks to the metadata exported by the gateway's patched `ext-auth` hook, verifying that the peer identity, trust anchor and cryptographic algorithms satisfy the requirements of the selected security level. The `child_create` module performs an ex-post validation of installed CHILD SAs by comparing the effective TSs and ESP parameters with the internal templates authorised by OPA. Any discrepancy yields a negative decision and triggers removal of the affected SA by the child manager. Finally, `rekey_ike_sa` steers make-before-break rekeys by comparing the relative strength of the old and new IKE suites and returning either a promotion or termination directive.

Although they address distinct validation stages, these modules adopt the same structured and modular approach as the core policies above, enabling uniform evaluation semantics and straightforward future extensions.

7.3.6 Policy network services

All containers in the `opa-network` adopt a least-privilege posture. They run as non-root users, mount configuration and certificates as read-only volumes, set `no-new-privileges:true`, drop all capabilities except those needed (e.g. `CHOWN` for OPA to change ownership of policy directories), and use `tmpfs` for writable paths with `noexec,nosuid` flags. All these measures assist in mitigating hazards associated with lateral movement and privilege escalation.

CTI bundle service

The CTI unified service is implemented in Python using `FastAPI`. It runs on the `opa-network`, fetches threat feeds from "AbuseIPDB" on a schedule (configurable via `CTI_FETCH_INTERVAL`), and aggregates them into a unified dataset. Each update produces a JSON bundle containing arrays of malicious IPs, a monotonic `version` number, and an `expires` timestamp. The service signs the bundle with an EdDSA private key stored in a Docker secret. The corresponding public key is distributed to OPA via `opa-config-unified.yaml`. To support caching, the service returns an `ETag` header that encodes the bundle version, then OPA includes `If-None-Match` on each fetch, and the service responds with `304` if the dataset has not changed. The service exposes a `/bundles/cti` endpoint protected by a bearer token. More in details, the OPA's configuration includes:

```
services:
  cti-bundle-service:
    url: http://cti-unified-service:8090/bundles/cti
    credentials:
      bearer_token: ${CTI_BUNDLE_TOKEN}
bundles:
  - name: cti
    service: cti-bundle-service
    polling: true
    verified_signature: true
    signing_key: |
      -----BEGIN PUBLIC KEY-----
      ...
      -----END PUBLIC KEY-----
```

Bundles are periodically pulled by the policy engine and stored under `data.cti.threats`, where the Rego policies call helper functions to check whether an IP is malicious.

Decision Logger

The decision logger is a `Node.js` service that receives all decisions from OPA via the `decision_logs` sink. It uses the `http` module to listen on port **8080** and writes three separate ISO 8601 timestamped logs on a per-day basis: `decisions-YYYY-MM-DD.log` (full request and result), `audit-YYYY-MM-DD.log` (compact summary with outcome, peer IP, IKE level, child level, rekey downgrade result and deny reason) and `errors-YYYY-MM-DD.log`. Sensitive fields are redacted, and derived attributes (e.g. allow/deny flag, service and peer tags) are added. The logger exposes a `/metrics` endpoint using the `prom-client` library, which exports counters such as `opa_decisions_total`, `opa_decisions_allow`, `ike_rekeys_downgrade`, etc. Moreover, batching is configured via environment variables: decisions are streamed immediately, while summaries can be grouped and flushed on a timer.

Metrics and dashboards

Prometheus runs in its own container and scrapes three endpoints: OPA's runtime metrics (`http://opa-server:8282/metrics`), the decision logger (`/metrics`) and the CTI service one, every 5 seconds. The configuration file `prometheus.yml` defines separate jobs for each and stores time-series in a local volume. While Grafana runs as another container, pre-provisioned with a datasource pointing at Prometheus and dashboards for *operations* (fig. 7.2: OPA heap usage, evaluation durations, bundle refresh times) and *security* (fig. 7.3: volume of allow/deny decisions, IKE/Child levels negotiated, rekey outcomes, CTI freshness). Alert rules are defined in Grafana's alerting UI. For instance, trigger a warning if `opa_decisions_deny` exceeds a threshold in a rolling window, or if the CTI bundle is older than six hours.

Figure 7.2. OPA system monitoring dashboard

Figure 7.3. OPA security monitoring dashboard

Chapter 8

Test

This chapter contains the description of several tests conducted on the implemented solution.

The first part outlines the capabilities provided by the implemented system, followed by a performance assessment examining execution time and the volume of packets and bytes exchanged during operation. Subsequently, a practical demonstration is presented to illustrate the DoS exposure inherent in enabling multiple unauthenticated multi-ADDKE exchanges during the initial IKE SA negotiation. A comparative analysis is then performed between a conventional legacy IKE configuration, where the CHILD SA is instantiated directly following the `IKE_AUTH` exchange, and the legacy childless procedure, which serves as the reference baseline against which our post-quantum establishment method is evaluated. The evaluation also measures the overhead contributed by OPA, taking into account the combined time spent on *input parsing*, *query compilation*, *query evaluation* and additional processing within each PDP decision cycle.

Finally, a comprehensive analysis of the entire session-establishment workflow is provided. For every phase, we report the elapsed time, the OPA evaluation latency, the delay between the PDP decision and the subsequent CHILD SA installation, the child creation duration itself, and the number of packets, fragments and bytes exchanged. The additional cost of the ex-post CHILD validation, after which the secure tunnel is considered fully operational, is also included. This examination is repeated for all defined post-quantum security levels, enabling a detailed comparison of their respective performance characteristics.

8.1 Testbed

The experiments were conducted on an **HP Pavilion Gaming 15-`dk0000` Laptop PC** with the following specifications:

- **CPU**: Intel Core i5-9300H @ 2.40 GHz;
- **GPU**: NVIDIA GeForce GTX 1650 4 GB GDDR6 Dedicated VRAM;
- **RAM**: 8 GB DDR4;
- **Storage**: 256 GB SSD, 500 GB HDD;
- **OS**: Ubuntu 22.04 LTS, 64-bit.

8.2 Functional tests

Different tests have been executed to check the correctness of the features introduced.

In sequence, the section covers the establishment of the IKE SA, followed by the scenario in which the initial setup is rejected because the negotiated security level is insufficient, thereby

activating the additional vendor-specific `Notifier`. Subsequently, the parameters of the installed child are further examined with ex-post validation to ensure that the installed child corresponds exactly to the configuration mandated by OPA for the relevant security level. Finally, the rekey downgrade validation mechanism is analysed.

8.2.1 IKE SA establishment

The first experiment illustrates the correct establishment flow of an IKE SA. The procedure begins with the `IKE_INIT` exchange, followed by an `IKE_INTERMEDIATE` round, which in this scenario is fragmented into two parts due to the sizeable ML-KEM − 1024 (Kyber_L5) key exchange payloads. Subsequently, the peers proceed with the `IKE_AUTH` exchanges, which themselves appear in multiple fragments in both directions. This fragmentation is primarily caused by the certificate transmission, which represents the dominant contributor to message size in this phase. In this testbed, the end-entity certificate relies on ML-DSA − 65 (Dilithium_3) for its public key and issuer's signature algorithms.

```
[IKE] initiating IKE_SA banka-responder[4] to 203.0.113.2
[ENC] generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG) N(REDIR_SUP) N(IKE_INT_SUP) V ]
[NET] sending packet: from 198.51.100.10[500] to 203.0.113.2[500] (384 bytes)
[NET] received packet: from 203.0.113.2[500] to 198.51.100.10[500] (461 bytes)
[ENC] parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(FRAG_SUP) N(HASH_ALG) N(CHDLESS_SUP) N(IKE_INT_SUP) N(MULT_AUTH) V ]
[IKE] received strongSwan vendor ID
[CFG] selected proposal: IKE:AES_GCM_16_256/PRF_HMAC_SHA2_512/ECP_521/KE1_KYBER_L5
[IKE] remote host is behind NAT
[IKE] received cert request for "CN=PQ-Gateway Root CA"
[IKE] received cert request for "O=PQ-Gateway Lab, CN=PQ-Gateway IKE CA PQ"
[IKE] received cert request for "O=PQ-Gateway Lab, CN=PQ-Gateway Root CA PQ"
[IKE] received cert request for "CN=Post-Quantum CA, OU=strongSwan, O=Linux strongSwan"
[ENC] generating IKE_INTERMEDIATE request 1 [ KE ]
[ENC] splitting IKE message (1633 bytes) into 2 fragments
[ENC] generating IKE_INTERMEDIATE request 1 [ EF(1/2) ]
[ENC] generating IKE_INTERMEDIATE request 1 [ EF(2/2) ]
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (450 bytes)
[NET] received packet: from 203.0.113.2[4500] to 198.51.100.10[4500] (1248 bytes)
[ENC] parsed IKE_INTERMEDIATE response 1 [ EF(1/2) ]
[ENC] received fragment #1 of 2, waiting for complete IKE message
[NET] received packet: from 203.0.113.2[4500] to 198.51.100.10[4500] (450 bytes)
[ENC] parsed IKE_INTERMEDIATE response 1 [ EF(2/2) ]
[ENC] received fragment #2 of 2, reassembled fragmented IKE message (1633 bytes)
[ENC] parsed IKE_INTERMEDIATE response 1 [ KE ]
[IKE] sending cert request for "O=PQ-Gateway Lab, CN=PQ-Gateway Root CA PQ"
[IKE] authentication of 'banka' (myself) with DILITHIUM_3 successful
[IKE] sending end entity cert "CN=banka"
[ENC] generating IKE_AUTH request 2 [ IDi CERT N(INIT_CONTACT) CERTREQ IDr AUTH N(MOBIKE_SUP) N(NO_ADD_ADDR) N(MULT_AUTH) N(EAP_ONLY) N(MSG_ID_SYN_SUP) ]
[ENC] splitting IKE message (9056 bytes) into 8 fragments
[ENC] generating IKE_AUTH request 2 [ EF(1/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(2/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(3/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(4/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(5/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(6/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(7/8) ]
[ENC] generating IKE_AUTH request 2 [ EF(8/8) ]
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (751 bytes)
[IKE] retransmit 1 of request with message ID 2
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
[NET] sending packet: from 198.51.100.10[4500] to 203.0.113.2[4500] (1248 bytes)
```

Figure 8.1. Initial IKE SA exchanges

After authentication concludes, but before the IKE SA is fully installed, the `ext-auth` plugin intercepts the workflow. At this point, the state machine is deliberately paused while the plugin's script extracts all relevant metadata, including the peer identity, source address, negotiated cryptographic suite and certificate attributes. These are transmitted to the PDP (OPA) for policy evaluation.

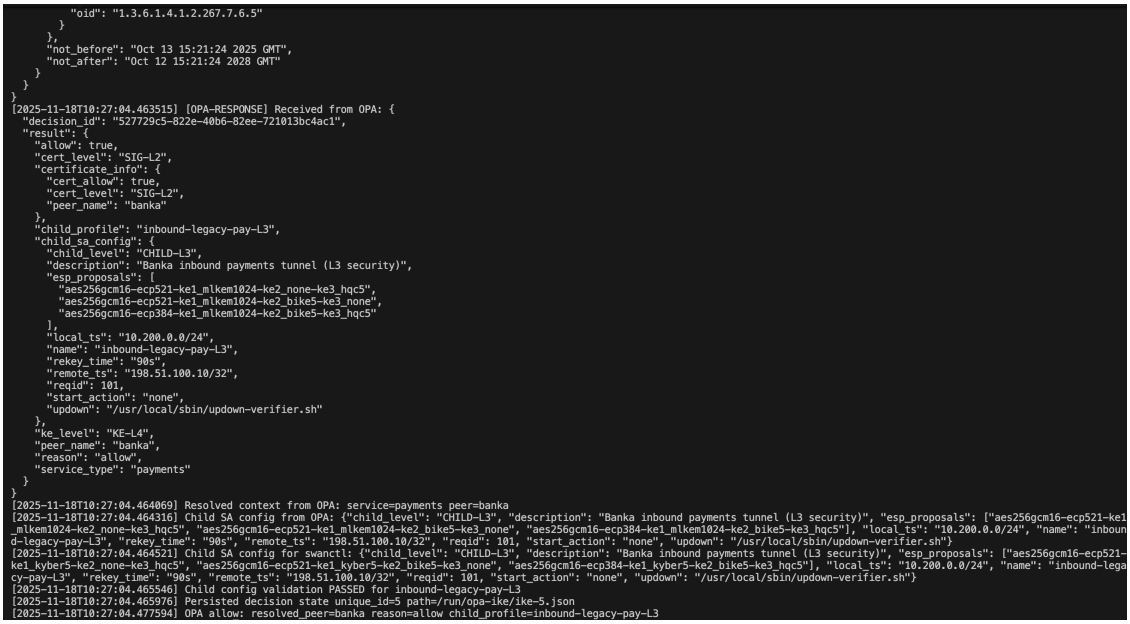

Figure 8.2. IKE SA metadata collection via `ext-auth`

Based on this input, the PDP returns a decision that includes both the approved security level and the complete child SA configuration template to be enforced. Before finalising the installation of the IKE SA, the returned template is validated against the locally defined `swanctl` configuration to ensure that the child to be installed corresponds exactly to the policy mandated one. Once this final consistency check succeeds, the `ext-auth` plugin issues an authorisation success verdict and the IKE SA is installed.

```
[ENC] received fragment #13 of 13, reassembled fragmented IKE message (14593 bytes)
[ENC] parsed IKE_AUTH response 2 [ IDr CERT CERT AUTH N(MOBIKE_SUP) N(ADD_4_ADDR) N(ADD_4_ADDR) ]
[IKE] received end entity cert "CN=pep-gateway"
[IKE] received issuer cert "O=PQ-Gateway Lab, CN=PQ-Gateway IKE CA PQ"
[CFG]   using trusted certificate "CN=pep-gateway"
[CFG]   using trusted intermediate ca certificate "O=PQ-Gateway Lab, CN=PQ-Gateway IKE CA PQ"
[CFG]   using trusted ca certificate "O=PQ-Gateway Lab, CN=PQ-Gateway Root CA PQ"
[CFG]   reached self-signed root ca with a path length of 1
[IKE] authentication of 'pep-gateway' with DILITHIUM_3 successful
[IKE] peer supports MOBIKE
[IKE] IKE_SA banka-responder[4] established between 198.51.100.10[banka]...203.0.113.2[pep-gateway]
[IKE] scheduling rekeying in 112s
[IKE] maximum IKE_SA lifetime 124s
initiate completed successfully
```

Figure 8.3. IKE SA successfully installed

8.2.2 IKE SA establishment denial due to insufficient security level

When the PDP rejects an incoming IKE SA request because the negotiated parameters do not satisfy the minimum security requirements, the gateway halts the establishment process. In this case, the PDP response includes both the reason for the denial and the minimum KE and signature levels expected for that specific peer. Consequently, the responder returns a failure during the `IKE_AUTH` exchange: besides the standard `AUTH_FAILED` notification, the gateway also emits a vendor-specific notification whose payload explicitly encodes the required KE and signature levels.

```
[NET] sending packet: from 198.51.100.11[4500] to 203.0.113.2[4500] (760 bytes)
[NET] received packet: from 203.0.113.2[4500] to 198.51.100.11[4500] (90 bytes)
[ENC] parsed IKE_AUTH response 2 [ N(AUTH_FAILED) N(BEET_MODE) ]
[IKE] received AUTHENTICATION_FAILED notify error
initiate failed: establishing IKE_SA 'partnerb-responder' failed
```

Figure 8.4. IKE SA establishment denied owing to an insufficient security level

The remote peer, which also includes the corresponding parsing patch, is capable of recognising, decoding, and logging this additional notification. As illustrated in fig. 8.5, the peer processes the vendor-specific payload and records the required cryptographic levels, thereby enabling diagnostic feedback and future renegotiations that comply with the advertised security policy.

```
Nov 18 10:52:14 08[ENC] <partnerb-responder|2> parsing NOTIFY payload, 33 bytes left
Nov 18 10:52:14 08[ENC] <partnerb-responder|2> parsing payload from => 33 bytes @ 0xffff7001ac28
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>    0: 29 00 00 08 00 00 00 18 00 00 00 19 00 00 A0 01  ).............. 
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>   16: 4B 45 2D 4C 32 3B 63 65 72 74 3D 53 49 47 2D 4C  KE-L2;cert=SIG-L
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>   32: 32                                              2
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>   parsing rule 0 U_INT_8
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>    => 41
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>   parsing rule 1 FLAG
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>    => 0
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>   parsing rule 2 RESERVED_BIT
Nov 18 10:52:14 08[ENC] <partnerb-responder|2>    => 0
```

Figure 8.5. Peer-side parsing of vendor-specific notification indicating required security levels

8.2.3 Child SA installation

As already noted, in the final phase of the exchange, once the PDP returns the authorised child template, the gateway verifies that this configuration exactly matches the corresponding entry in `swanctl.conf`. If the template is consistent with the locally declared policy, it is persisted under `/run/opa-ike/ike-<id>.json`, thereby recording both the decision and the full set of parameters required for subsequent enforcement.

Figure 8.6. Persisted OPA decision and child template

From this point, the `vici-child-manager` retrieves the approved child profile name from the stored decision file and triggers its installation. The ensuing CHILD SA negotiation follows the standard strongSwan flow. However, in this instance the level-4 child requires multiple fragments, reflecting the computational weight and message size associated with the HQC-5 component of the multi-KEM ESP proposal.

Figure 8.7. Authorised CHILD SA being initiated via `VICI`

Once installed, the CHILD SA is formally associated with the pre-existing corresponding IKE SA, completing the childless workflow in which the child is created only after the IKE SA has been authorised and validated.

```
gw-wan-in-bankA: #20, ESTABLISHED, IKEv2, 9fc1f57fea2d6b7a_i* fe8f329fb154c208_r
  local  'pep-gateway' @ 203.0.113.2[4500]
  remote 'banka' @ 198.51.100.10[4500]
  AES_GCM_16-256/PRF_HMAC_SHA2_512/ECP_521/KE1_KYBER_L5
  established 18s ago, rekeying in 100s
  inbound-legacy-pay-L3: #29, reqid 101, INSTALLED, TUNNEL-in-UDP, ESP:AES_GCM_16-256/ECP_521/KE1_KYBER_L5/KE3_HQC_L5
    installed 50s ago, rekeying in 34s, expires in 51s
    in  c597d61f,      0 bytes,     0 packets
    out ca0cb60c,      0 bytes,     0 packets
    local  10.200.0.3/32
    remote 198.51.100.10/32
```

Figure 8.8. Final CHILD SA bound to the IKE SA after installation

8.2.4 Child ex-post validation

After the CHILD SA has been successfully installed, the `updown` verification hook performs an ex-post consistency check, consulting again the PDP. Using the parameters obtained via VICI, it inspects the effectively negotiated ESP suite and traffic selectors, ensuring that they precisely correspond to the configuration authorised by the PDP for that specific connection. Any deviation, whether a weaker cryptographic proposal, an unexpected selector, or an omitted KEM component, would trigger a controlled teardown of the tunnel to prevent the establishment of a non-compliant security association.

Once the verification step has completed, the result is passed to the *decision logger*, which records both the authoritative OPA decision and the parameters of the CHILD SA. This yields an auditable, time-stamped trace of the enforcement path, capturing the effective security level and the outcome of the ex-post validation.

```
025-11-18T10:59:45.457989Z updown[16] DEBUG Resolved service=payments (source=state.service) peer_addr=198.51.100.10 (source=peer.resolved_ip)
                           child_profile=inbound-legacy-pay-L3
025-11-18T10:59:45.458253Z updown[16] DEBUG Child payload addke (converted to OPA format): {'ke1': 'mlkem1024', 'ke2': 'none', 'ke3': 'none'}
025-11-18T10:59:45.458667Z updown[16] DEBUG Prepared child.create payload: {"input": {"child": {"addke": {"ke1": "mlkem1024", "ke2": "none", "ke3": "none"},
                           "esp": {"aead": "aes256gcm16", "dh": "ecp521"}, "name": "inbound-legacy-pay-L3", "ts": {"local": ["10.200.0.0/24"],
                           "remote": ["198.51.100.10/32"]}}, "ike_unique_id": "16", "level": "CHILD-L3", "peer_addr": "198.51.100.10", "phase": "child_up",
                           "role": "responder", "service": "payments"}}
025-11-18T10:59:45.475164Z updown[16] DEBUG OPA child.create response: {"decision_id": "0ed7191b-0858-454c-89ca-fdb77a96a349",
                           "result": {"allow": true, "level": "CHILD-L3", "reason": "allow"}}
025-11-18T10:59:45.493164Z updown[16] INFO OPA child.create allow for inbound-legacy-pay-L3: reason=allow
```

Figure 8.9. Ex-post validation performed by the `updown` hook

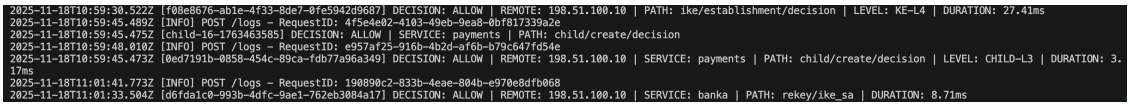

Figure 8.10. Decision Logger entry for validated CHILD SA

Selector mismatch enforcement

To demonstrate that the `ex-post` validation mechanism reliably blocks any CHILD SA whose parameters diverge from the policy authorised by the PDP, a controlled tampering experiment was carried out using the `tamper-updown.sh` script. The scenario emulates an adversary manipulating state on the enforcement gateway, deliberately corrupting the cached metadata consumed by the `updown` hook so that the traffic selectors stored in the cache no longer correspond to those negotiated on the wire.

In order, the script first restarts the `gateway` and `banka` containers, reloads all `swanctl` configuration and establishes a clean IKE/CHILD context (`gw-wan-out-bankA/to-banka-L3`). It then suspends the auxiliary services that manage metadata (`vici-suite-publisher`, `child_suite_cache`, `vici-child-manager`), ensuring that the cached state remains fixed while it is being manipulated. The child-suite file under `/run/opa-ike/child-suites/ike-*-to-banka-L3.json` is subsequently edited, replacing the authorised `ts.local` with the broader prefix `10.200.0.0/24`, even though the on-wire negotiation continues to request `10.200.0.3/32`. A fail-close flag (`/run/opa-ike/fail_on_mismatch`) is then created so that any detected discrepancy forces an immediate teardown. A fresh IKE/CHILD negotiation is triggered, causing `updown` to validate against the tampered metadata. Once the experiment concludes, the flag is removed and the previously suspended helper processes are resumed.

As illustrated in fig. 8.11, the log output shows the hook successfully identifying the discrepancy: the cached and negotiated selectors (`meta_local/meta_remote` versus `config_local/config_remote`) differ, prompting an immediate policy-driven teardown.

Every subsequent installation attempt under the corrupted state is therefore consistently rejected, confirming that the enforcement layer prevents the activation of any CHILD SA whose selectors deviate from the PDP approved configuration.

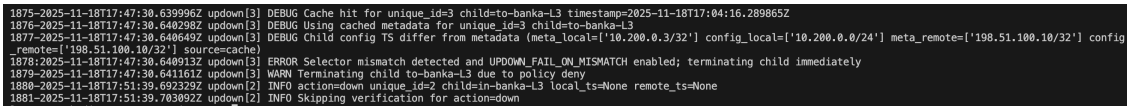

Figure 8.11. The `updown` hook detecting a selector mismatch between OPA metadata and the negotiated traffic selectors, and aborting the CHILD SA installation

8.2.5 Rekey validation gate

A final validation stage is executed during the lifetime of an established IKE SA in order to prevent any form of cryptographic downgrade when rekeying occurs. When strongSwan signals that an IKE rekey is imminent, the `vici-child-manager` intercepts the event and leverages the native *make-before-break* mechanism to retrieve both the "old" and the "new" cryptographic suites associated with the rekey attempt. The suites, are packaged and submitted to OPA through the dedicated rekey policy endpoint.

```
2025-11-18 12:14:42,626 - INFO - IKE REKEY detected for gw-wan-out-bankA: uniqueid 9 -> 10
2025-11-18 12:14:42,626 - INFO - Preparing OPA rekey validation for gw-wan-out-bankA
2025-11-18 12:14:42,626 - INFO -   Old suite: {'aead': 'aes256gcm16', 'prf': 'sha512', 'dh': 'ecp521',
                          'addke': {'ke1': 'mlkem1024', 'ke2': 'none', 'ke3': 'none'}}
2025-11-18 12:14:42,626 - INFO -   New suite: {'aead': 'aes256gcm16', 'prf': 'sha512', 'dh': 'ecp521',
                          'addke': {'ke1': 'mlkem1024', 'ke2': 'none', 'ke3': 'none'}}
2025-11-18 12:14:42,627 - INFO - Calling OPA rekey policy: http://opa-server:8181/v1/data/rekey/ike_sa
2025-11-18 12:14:42,627 - INFO - Payload: {
  "input": {
    "service": "banka",
    "connection_name": "gw-wan-out-bankA",
    "peer_addr": "198.51.100.10",
    "old": {
      "aead": "aes256gcm16",
      "prf": "sha512",
      "dh": "ecp521",
      "addke": {
        "ke1": "mlkem1024",
        "ke2": "none",
        "ke3": "none"
      }
    },
    "new": {
      "aead": "aes256gcm16",
      "prf": "sha512",
      "dh": "ecp521",
      "addke": {
        "ke1": "mlkem1024",
        "ke2": "none",
        "ke3": "none"
      }
    }
  }
}
2025-11-18 12:14:42,644 - INFO - OPA rekey decision: allow=True, reason=allow, action=promote_new_delete_old
2025-11-18 12:14:42,645 - INFO - IKE rekey ALLOWED by OPA for gw-wan-out-bankA
```

Figure 8.12. Rekey request intercepted and evaluated against PDP policy

The PDP evaluates whether the proposed replacement satisfies the minimum security requirements for the associated service. If the new suite preserves or improves the previously achieved level, the `vici-child-manager` promotes the fresh IKE SA and subsequently tears down the old instance. Conversely, if the rekey attempt results in a weaker configuration, the new IKE SA is rejected and the existing one is retained, thus preventing any inadvertent or malicious downgrade.

```
2025-11-18T12:14:44.125Z [INFO] POST /logs - RequestID: 73c82307-f79b-4794-b4c5-29a74e91406a
2025-11-18T12:14:42.638Z [701bef00-7fc3-4f81-b57c-fe03793111a5] DECISION: ALLOW | REMOTE: 198.51.100.10 | SERVICE: banka | PATH: rekey/ike_sa | DURATION:
 3.29ms
```

Figure 8.13. Audit entry showing the validated rekey decision and resulting action

8.3 Performance tests

In this section, we present the performance evaluation of the proposed solution and contrast it with a classical baseline composed of a standard IKE exchange followed by immediate CHILD SA creation. The analysis examines the latency introduced throughout the entire establishment workflow, the additional computational effort required for post-quantum primitives, and the resulting increase in packet count, fragmentation and byte overhead.

Furthermore, the evaluation isolates the delay attributable to PDP consultation at each stage of the negotiation process, from the initial policy query to the moment at which a functional protected channel becomes available. A detailed breakdown of the policy evaluation pipeline is also provided, highlighting the contribution of each component, parsing, compilation, rule matching and final decision synthesis, to the overall OPA processing time observed during the establishment procedure.

8.3.1 Unauthenticated multi-KEM overhead in IKE establishment

This experiment quantifies the cost of adding multiple KEMs during the unauthenticated IKE SA setup, highlighting the DoS exposure that motivates the use of a single KEM in the IKE establishment and the use of the childless design adopted in this work. Two runs are compared: a baseline configuration in which the gateway negotiates a single post-quantum KE (ML-KEM-1024) and a multi-KEM configuration that adds BIKE-5 and HQC-5 in ADDKE #2 and ADDKE #3, respectively. Even though only three out of the seven additional KEM slots envisaged by RFC-9370 are exercised, the overall exchange time more than doubles (from roughly 100 ms to 217 ms), and the total UDP volume and packet count increase by factors of approximately 2.16 and 2.11, as illustrated in figs. 8.14 to 8.16. The bulk of the extra cost stems from the HQC exchange, which alone accounts for over 40 % of the total duration in the multi-KEM case.

The measurements are obtained by capturing the full IKE traffic on the `gateway` container using the helper script:

```
./scripts/capture-and-analyze-ike.sh \
  pep-gateway gw-wan-out-bankA 198.51.100.2
```

The script first terminates any existing IKE SA, then starts `tcpdump` inside the gateway container to record packets on UDP ports 500/4500. The resulting PCAP is copied to the host and analysed by `analyze-pcap-timing.py`, which invokes `tshark` to extract timestamps, message types and lengths, groups packets into phases (`IKE_SA_INIT`, `IKE_INTERMEDIATE` #1–#3, `IKE_AUTH`), and computes, for each phase, the duration, number of packets and bytes sent/received at the UDP level.

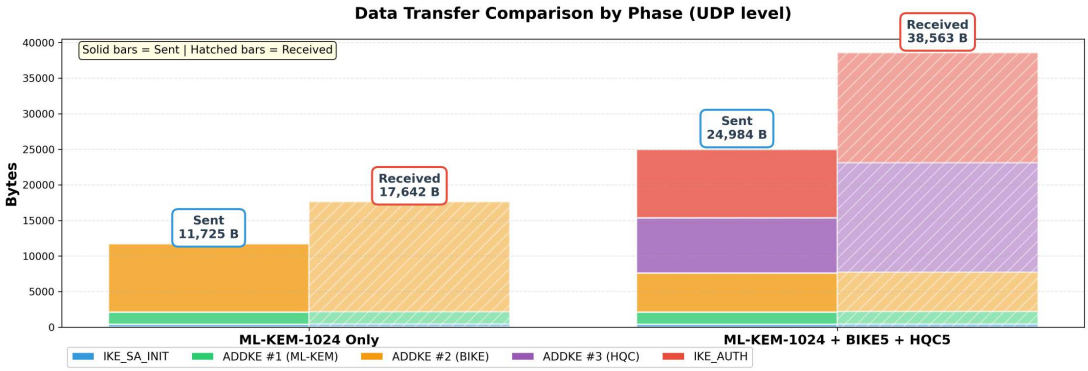

Figure 8.14. Comparison of transmitted and received data volumes for single-KEM and multi-KEM IKE establishment

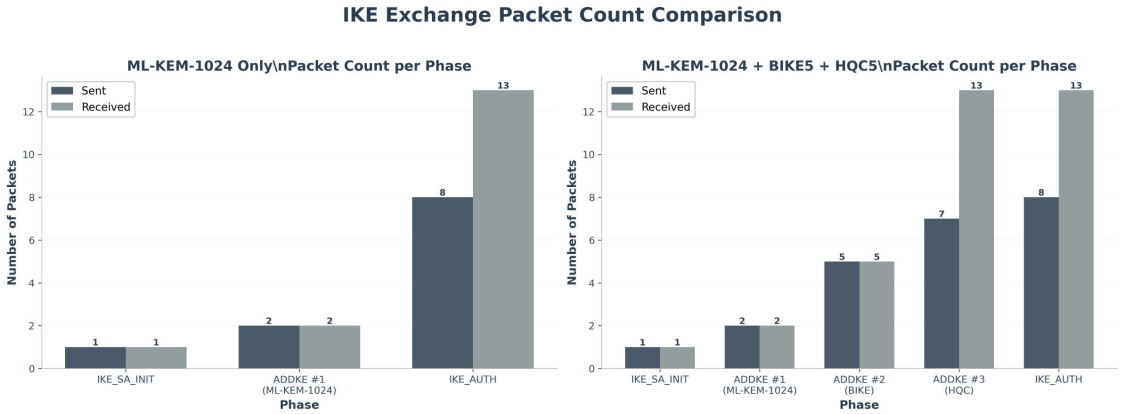

Figure 8.15. Packet count distribution across IKE phases for ML-KEM only versus multi-KEM setups

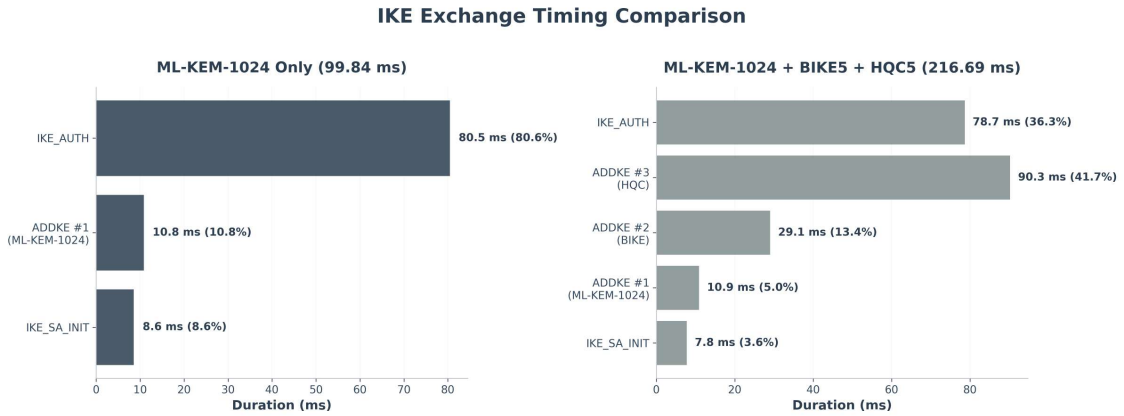

Figure 8.16. Phase-level timing comparison between ML-KEM only and multi-KEM IKE negotiations

8.3.2 Legacy IKE establishment: normal versus childless mode

A second experiment contrasts a conventional legacy IKE setup, in which a CHILD SA is created immediately during `IKE_AUTH`, with a childless initial exchange where only the IKE SA is established. As specified by RFC-7296, in the normal case the first CHILD SA is derived directly from the key material negotiated in `IKE_SA_INIT`. The `IKE_AUTH` message carries, in addition to the authentication payloads, also the ESP proposal and traffic selectors for that child. While additional effective independent key exchanges are used solely for subsequent `CREATE_CHILD_SA` operations to provide PFS.

The experiment is automated by the `simple-ike-comparison.sh` script, which alternates between the normal and childless configurations of the `gateway` and `host-legacy` containers. For each mode the script restarts the containers, reloads the `swanctl` configuration, captures the full handshake on the gateway with `tcpdump`, and derive the total establishment time and the number of bytes observed on the wire. The two JSON reports are finally compared, which computes the incremental overhead introduced by negotiating a CHILD SA within `IKE_AUTH`.

The results summarised in fig. 8.17 indicate that embedding a CHILD SA directly into the IKE handshake leads to a clear increase in both data volume and processing time. From a bandwidth standpoint, the overall handshake grows by roughly 7 %, driven by the additional payloads carried in the final IKE_AUTH response. In particular, this response enlarges by about 80 B, accounting for the ESP `SA` payload, the `TSi` and `TSr` payloads, and a small amount of padding and alignment overhead.

On the initiator side, the impact is more pronounced (around 476 B), as the peer must send the full CHILD SA proposal, the associated traffic selectors and several notify payloads. In our measurements, this request is fragmented at the IP layer into multiple packets, increasing both the number of packets on the wire and the per-packet processing cost for each endpoint.

From a latency perspective, the impact is even more pronounced. The end-to-end establishment time increases by more than a factor of three, from roughly 157 ms in the childless configuration to around 539 ms when the CHILD SA is negotiated in-band. This additional delay is largely due to the extra processing and fragmentation required to convey the CHILD proposal within IKE_AUTH. These results reinforce the hardening choices in our design, where the gateway deliberately establishes the initial IKE SA in childless mode and only subsequently, once the PDP has authorised the connection, creates a new CHILD SA using dedicated post-quantum key exchanges. This approach cleanly decouples tunnel key material from the initial IKE negotiation, keeps the IKE proposal compact (restricted to a single standardised KEM), and shifts the additional post-quantum overhead into an explicitly authorised and tightly controlled CHILD establishment phase, ensuring that even the first CHILD SA is characterised by a full, quantum-resistant key exchange.

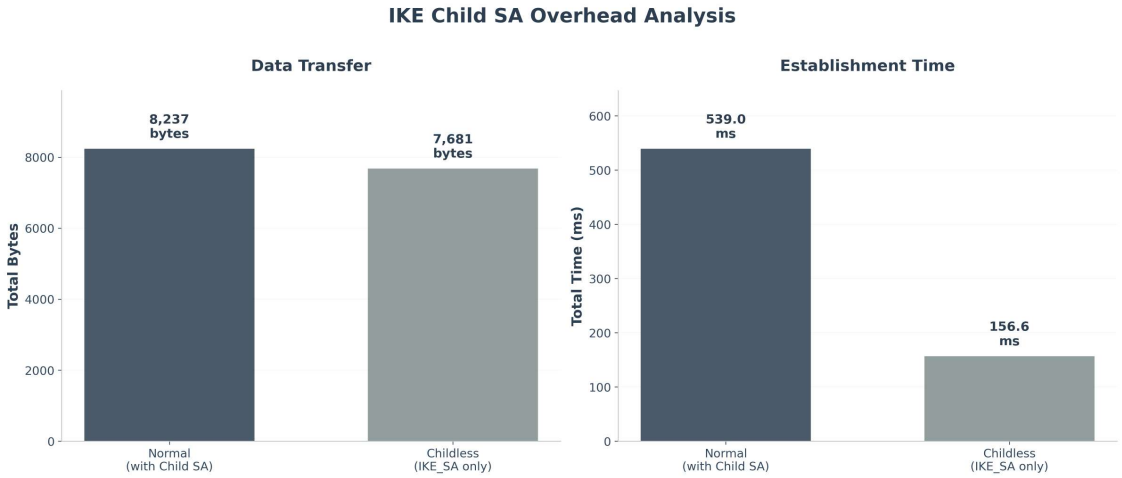

Figure 8.17. Data and time overhead of a normal IKE establishment with CHILD SA versus childless mode

8.3.3 Policy evaluation timing analysis

To quantify the computational cost introduced by the policy-driven control plane, we analyse the evaluation time of each OPA rule involved in the security workflow: the `ike/establishment` decision, the `child/create` decision, and the `rekey/ike_sa` downgrade prevention gate. Figure 8.18 reports the breakdown of the average latency observed for each policy phase, distinguishing the time spent parsing the input, compiling the Rego query, evaluating the policy logic, and the residual runtime overhead.

The evaluation of the IKE establishment policy is, by construction, the most time-consuming stage. This rule represents the core of the PDP's security assessment and consequently carries out the widest range of checks. Specifically, the decision:

- authenticate and classify the remote peer identity;
- determine whether the requested connection is admissible given the declared policy set;
- assess whether the negotiated IKE security level satisfies the minimum requirements for that peer or service;
- validate the remote certificate, including level requirements and subject matching;

- derive the authorised child template to be enforced by the gateway.

For these reasons, its cost dominates the evaluation process, averaging 10.48 ms, with more than 92 % of this time attributed to actual query evaluation.

On the contrary, the subsequent `child/create` and `rekey/ike_sa` stages are considerably more lightweight. Both rules serve as consistency checks, effectively performing pattern-matching against state that has already been authorised.

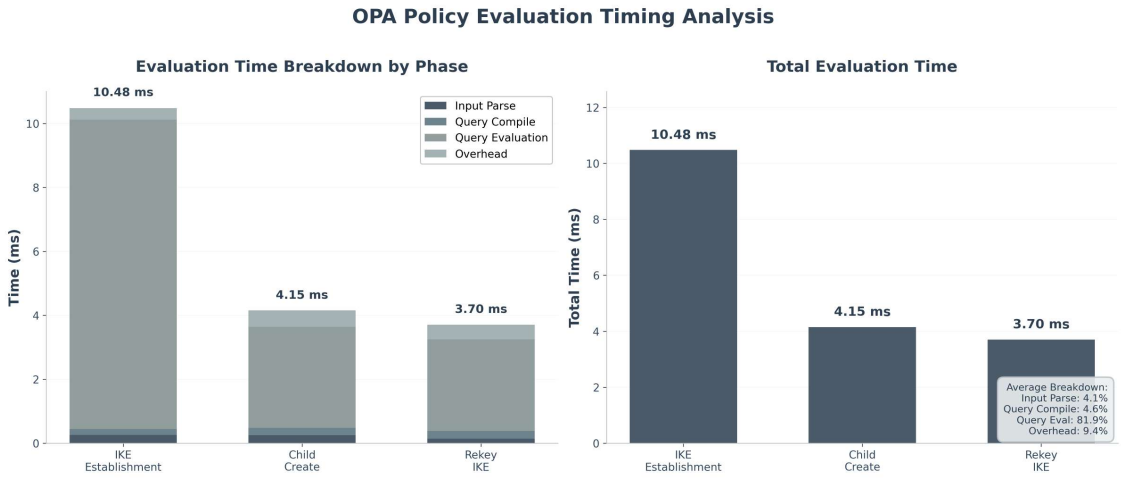

Figure 8.18. Breakdown of OPA evaluation times across the IKE establishment, child creation and rekey security gates

8.3.4 Comprehensive evaluation across security levels

This section consolidates the multi-level performance analysis by examining, in a unified manner, the timing characteristics, network overhead, and fragmentation behaviour observed across the four security levels. The measurements derive from controlled experiments conducted through the benchmarking framework executed via the `run_benchmark_suite.sh` script. Packets are captured on the `gateway` with `tcpdump` into PCAP traces, then parsed with `tshark` and correlated with `vici` and OPA audit logs to extract timing and traffic metrics. Although, it is important to highlight that the timing values exhibit a degree of non-determinism. Such variability primarily stems from the packet capture methodology, the behaviour of the Docker-based execution environment, and short-lived network congestion during the exchange phases. Consequently, the collected timings should be regarded as representative rather than absolute. It is worth noting that, across all evaluated configurations, the required signature strength is fixed at level 2. This choice provides a satisfactory level of assurance for authentication while avoiding the heavier level 3 option, whose cost is dominated by certificate exchange. At the same time, level 2 remains sufficient to study the differences between security levels arising from the distinct Additional Key Exchanges (ADDKEs) used in each case. However, this decision is easily reversible: since the security level of the KE is decoupled from that of the signature, it is sufficient to adjust the signature level in `service_classes.rego` for the relevant external service, and the new requirement will be propagated automatically during the validation phase.

Traffic, bytes and fragmentation

The aggregate network statistics in figs. 8.19 and 8.20 reflect the intrinsic cost of increasing post-quantum security levels. Packet counts, transferred bytes and the number of IKE fragments all grow consistently from L1 through L4. This trend is especially visible for the child installation stage, where the cryptographic workload directly influences the payload size and fragmentation ratio. At level 4, where ML-KEM-1024, BIKE-5, and HQC-5 are jointly negotiated, the child

establishment accounts for the highest number of fragments (34 fragments in total), and a corresponding traffic volume of approximately 37.1 KiB. Such growth is entirely expected as larger post-quantum public keys and ciphertexts must be transmitted during the ESP negotiation.

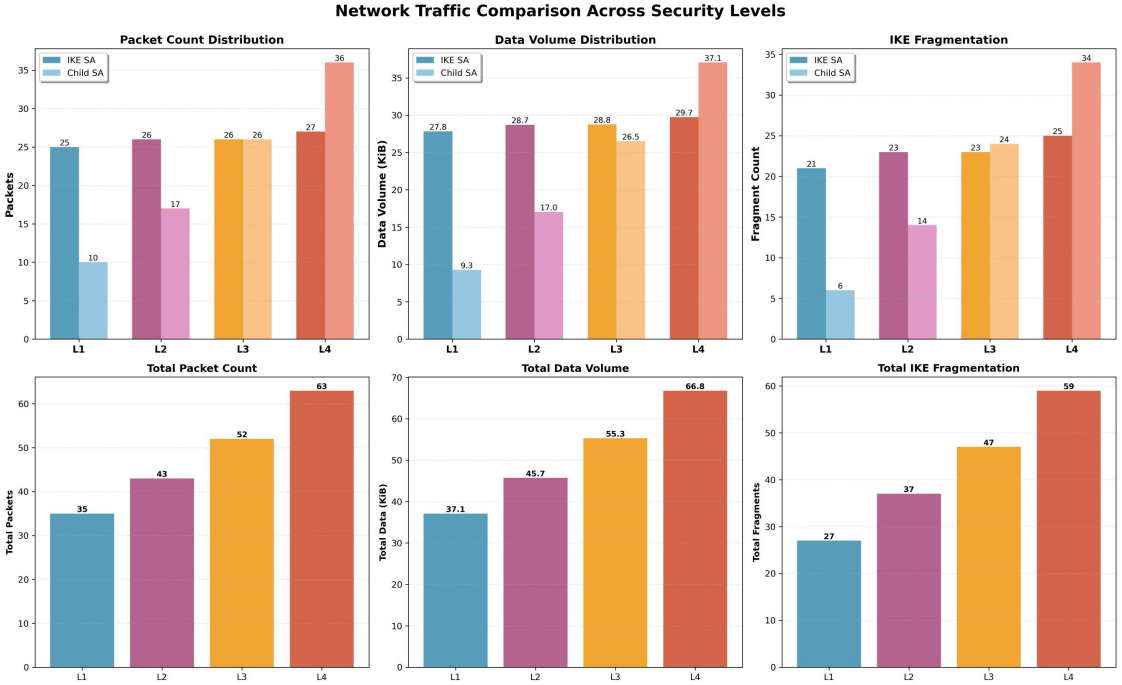

Figure 8.19. Network traffic comparison across all security levels

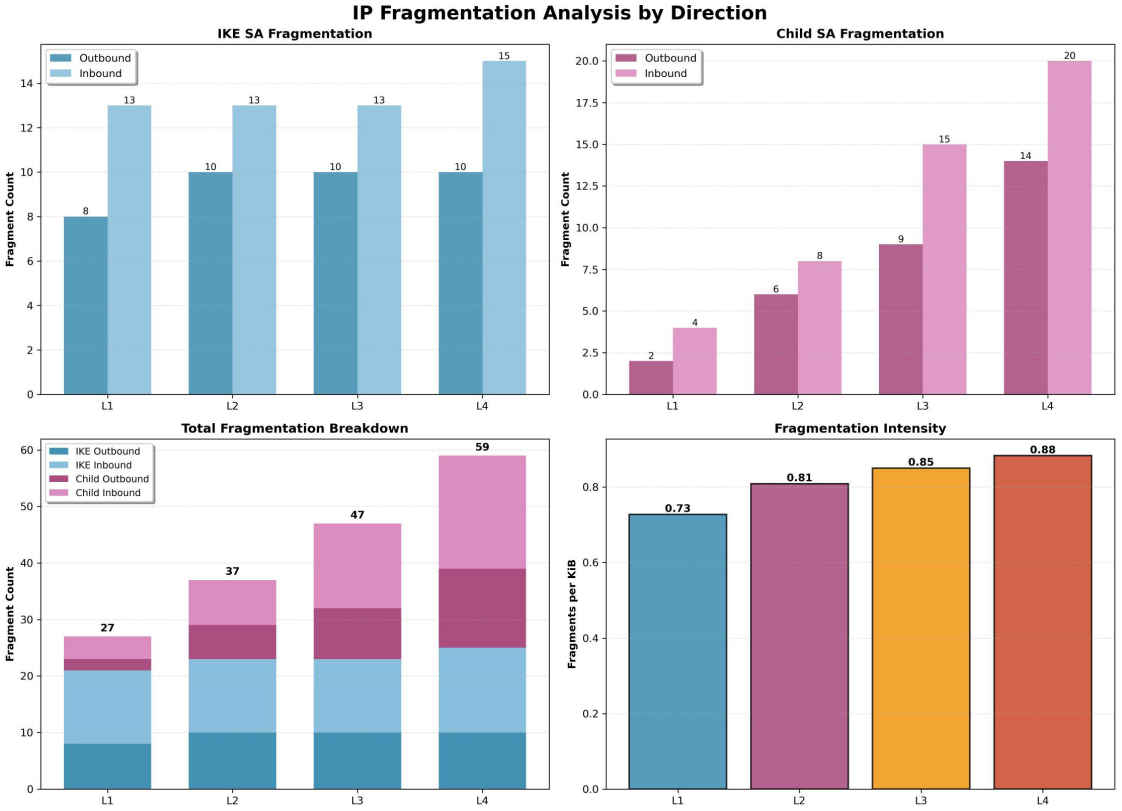

Figure 8.20. IP fragmentation comparison across security levels

Macroscopic timing behaviour

Although the cryptographic payload grows with the level, the global timing bottlenecks remain consistent across all tiers. The most significant contributor to the total duration is the *IKE authentication phase*, dominated by certificate exchange and signature verification. A second non-negligible component is the delay between the PDP's decision and the start of child negotiation, which varies according to system load, scheduling of the `vici-child-manager`, and transient conditions within the Docker network stack.

The child negotiation time (purple bar in fig. 8.21) increases monotonically with the security level, as anticipated, and reaches its maximum at level 4. This reflects the cumulative cost of negotiating three post-quantum KEMs sequentially. The evaluation also accounts for the ex-post validation performed by the `updown` mechanism to ensure that the installed suites strictly match the authorised template, to consider complete the entire secure tunnel establishment.

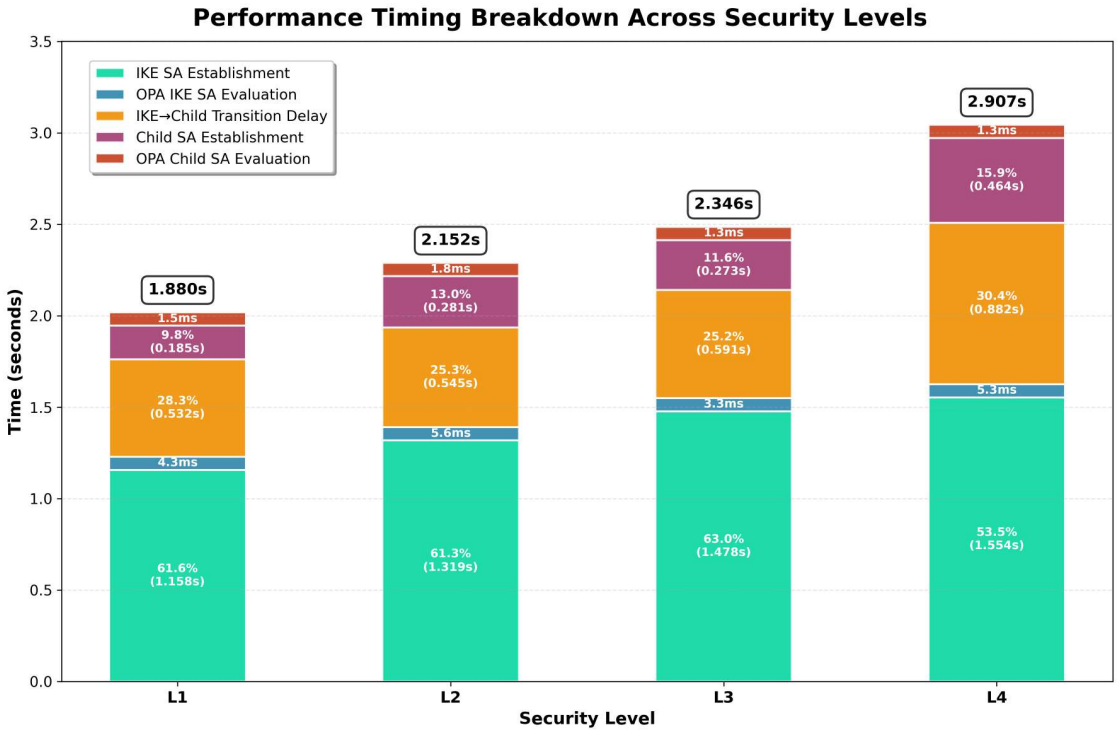

Figure 8.21. Performance timing breakdown across security levels

Overall interpretation

Despite a clear increase in establishment time relative to the classical legacy scenario, where the first child is derived immediately after `IKE_AUTH` with negligible additional negotiation, the proposed solution offers a qualitatively different security posture. In the proposed solution, when compared with the legacy baseline, even the lowest configured level incurs more than a fourfold increase in traffic load and more than 3.7 times the establishment time of the classical exchange, rising to around an eightfold increase in KiB transferred and a sixfold increase in latency at the highest level. Nevertheless, although this additional overhead is expected given the extra work introduced by post-quantum algorithms and certificate handling, it remains justified by the enhanced security posture offered by the system. The policy-driven approach enables complete control over the negotiation pipeline: the PDP can enforce the precise KE, signature and child-level combinations required for each service, while decoupling the establishment of the transmission tunnel from the IKE SA itself. The resulting overhead represents a one-time cost incurred only during the initial establishment, and is justified by the stringent security guarantees needed in the target deployment environment.

Chapter 9

Conclusions

This document has introduced and evaluated a policy-centric, post-quantum aware IPsec gateway. The proposed system is designed to shield legacy infrastructures that cannot yet adopt post-quantum schemes natively, while still allowing them to benefit from quantum-resistant protections. Informed by the analysis of the quantum threat landscape, the NIST PQC standardisation process and the practical obstacles of migrating IPsec and IKEv2, the gateway operates as a translation point: it terminates conventional IKEv2 sessions with a legacy LAN and reestablishes connectivity towards post-quantum capable peers on an external network, enforcing centrally defined cryptographic policies and target security levels along the way.

A central premise of this work is that *cryptographic agility*, rather than the selection of one specific algorithm, is the main design requirement for systems expected to face both classical and quantum-capable adversaries over time. To this end, the gateway does not embed fixed cryptographic choices in its data plane. Instead, it follows a policy-driven architecture in which an external Policy Decision Point (PDP), realised with Open Policy Agent (OPA), determines admissible algorithms, key exchange patterns, signature schemes and per-service security levels. The strongSwan-based gateway acts as a Policy Enforcement Point (PEP), terminating tunnels on both sides and consulting the PDP, before allowing security relevant state changes. This clean separation of roles decouples policy evolution from packet processing, so that changes in security posture can be realised by updating policy modules and data bundles, without modifying or restarting the enforcement component.

The prototype has been validated through functional tests that exercise the full negotiation workflow. The experiments show that the gateway correctly intercepts authenticated IKE exchanges, submits a rich context to the PDP, and proceeds only when an internally consistent child configuration is authorised. When a proposed cryptographic suite fails to meet the minimum requirements, the gateway rejects the IKE association and returns a diagnostic notification that can be logged and interpreted by the peer. Child SA installation is shown to derive exclusively from OPA-approved templates, which are recorded and cross-checked against the actually negotiated traffic selectors and ESP transforms. Any deviations cause a fail-closed teardown. Rekey validation further confirms that downgrade attempts are systematically blocked, and that new IKE SAs are accepted only when they preserve or raise the previously achieved security level.

Furthermore, the performance evaluation provides a quantitative perspective on the costs of adopting post-quantum primitives and policy-driven control. Measurements show that performing multiple unauthenticated ADDKE rounds during IKE SA setup substantially increases handshake latency and traffic volume, and makes the protocol more exposed to denial-of-service. This observation justifies constraining IKE itself to a single standardised KEM while moving additional post-quantum work into an authenticated, policy-gated child establishment phase. Additional benchmarks quantify the contribution of the PDP: even for the most complex rule, which performs peer classification, admissibility checks, level verification, certificate validation and child-template derivation, the average evaluation time remains in the order of a few tens of milliseconds, with lighter rules incurring only a few milliseconds. Across four configured security levels, packet counts, bytes and fragmentation grow as larger post-quantum keys and ciphertexts are used, yet

the overall timing profile remains dominated by certificate handling and the actual delay from the OPA response to the mandated child initialisation, with PDP evaluations contributing only a modest fraction. These results suggest that a policy-driven, post-quantum IPsec gateway is practically deployable, with overheads that are visible but acceptable for many site-to-site and high-value service scenarios.

This work demonstrates comprehensively that it is both technically and operationally achievable to build a post-quantum IPsec gateway that combines crypto-agility, fine-grained policy control and rich observability, while still maintaining interoperability with existing IKEv2 deployments. At the same time, the evaluation has been limited to a controlled laboratory setting. Real-world deployments will need to consider a broader range of hardware platforms, traffic profiles, failure modes, long-term data-plane performance and highly available, horizontally scalable PDP designs. Addressing these aspects, and generalising the policy-centric pattern beyond IPsec, for example to post-quantum aware TLS termination, service meshes and application-layer gateways, defines a natural agenda for future work and a coherent route towards quantum-safe, crypto-agile network security at scale.

Bibliography

[1] M. Giles, "Explainer: What is a quantum computer?." MIT Technology Review, January 2019, https://www.technologyreview.com/2019/01/29/66141/what-is-quantum-computing/

[2] IBM, "What is quantum computing?." https://www.ibm.com/topics/quantum-computing

[3] S. E. Yunakovsky, M. Kot, N. Pozhar, D. Nabokov, M. Kudinov, A. Guglya, E. O. Kiktenko, E. Kolycheva, A. Borisov, and A. K. Fedorov, "Towards security recommendations for public-key infrastructures for production environments in the post-quantum era", EPJ Quantum Technology, vol. 8, May 2021, DOI 10.1140/epjqt/s40507-021-00104-z

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM Journal on Computing, vol. 26, October 1997, pp. 1484–1509, DOI 10.1137/S0097539795293172

[5] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits", Quantum, vol. 5, April 2021, p. 433, DOI 10.22331/q-2021-04-15-433

[6] M. Giles, "Explainer: What is post-quantum cryptography?." MIT Technology Review, July 2019, https://www.technologyreview.com/2019/07/12/134211/explainer-what-is-post-quantum-cryptography/

[7] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process." NIST IR 8413-upd1, July 2022, DOI 10.6028/NIST.IR.8413-upd1

[8] NIST, "Announcing Approval of Three Federal Information Processing Standards (FIPS) for Post-Quantum Cryptography." NIST CSRC, August 2024, https://csrc.nist.gov/News/2024/postquantum-cryptography-fips-approved

[9] S. Kent, "IP Encapsulating Security Payload (ESP)." RFC-4303, December 2005, DOI 10.17487/RFC4303

[10] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)." RFC-7296, October 2014, DOI 10.17487/RFC7296

[11] V. Smyslov, "Intermediate Exchange in the Internet Key Exchange Protocol Version 2 (IKEv2)." RFC-9242, May 2022, DOI 10.17487/RFC9242

[12] C. J. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, and V. Smyslov, "Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)." RFC-9370, May 2023, DOI 10.17487/RFC9370

[13] Jay Gambetta, "The hardware and software for the era of quantum utility is here." IBM Quantum Blog, 2023, https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[14] Microsoft, "Microsoft's Majorana 1 chip carves new path for quantum computing." Microsoft Source (Newsroom), February 2025, https://news.microsoft.com/source/features/innovation/microsofts-majorana-1-chip-carves-new-path-for-quantum-computing/

[15] M. Mosca and M. Piani, "Quantum threat timeline report 2024." Global Risk Institute, 2024, https://globalriskinstitute.org/publication/2024-quantum-threat-timeline-report/

[16] G. Greenwald, "XKeyscore: NSA tool collects 'nearly everything a user does on the internet'." The Guardian, July 2013, https://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data

[17] L. K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search", STOC '96: 28th Annual ACM Symposium on Theory of Computing, Philadelphia (PA, USA), May 22-24, 1996, pp. 212–219, DOI 10.1145/237814.237866

[18] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance", Nature, vol. 414, December 2001, pp. 883–887, DOI 10.1038/414883a

[19] European Telecommunications Standards Institute, "Quantum Computing Impact on security of ICT Systems; Recommendations on Business Continuity and Algorithm Selections." ETSI EG 203 310, June 2016, https://www.etsi.org/deliver/etsi_eg/203300_203399/203310/01.01.01_60/eg_203310v010101p.pdf

[20] European Telecommunications Standards Institute, "Quantum-Safe threat assessment." ETSI GR QSC 004, March 2017, https://www.etsi.org/deliver/etsi_gr/qsc/001_099/004/01.01.01_60/gr_qsc004v010101p.pdf

[21] John Proos and Christof Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves", QIC 3, no. 4, 2003, pp. 317–344, DOI 10.48550/arxiv.quant-ph/0301141

[22] A. Banerjee, T. Reddy.K, D. Schoinianakis, T. Hollebeek, and M. Ounsworth, "Post-Quantum Cryptography for Engineers." Internet-Draft, IETF, August 2025, https://datatracker.ietf.org/doc/draft-ietf-pquip-pqc-engineers/14/

[23] E. Barker, "Recommendation for Key Management: Part 1 – General." NIST SP 800-57, pt.1, rev.5, May 2020, DOI 10.6028/NIST.SP.800-57pt1r5

[24] D. Coppersmith, "Another birthday attack", CRYPTO '85: Advances in Cryptology, Santa Barbara, (CA, USA), August 18-22, 1985, pp. 14–17, DOI 10.1007/3-540-39799-X_2

[25] D. Ott, C. Peikert, *et al.*, "Identifying Research Challenges in Post-Quantum Cryptography Migration and Cryptographic Agility." arXiv:1909.07353, September 2019, DOI 10.48550/arXiv.1909.07353

[26] Open Quantum Safe Project, "The Liboqs Library." https://openquantumsafe.org/liboqs/

[27] N. Alnahawi, N. Schmitt, A. Wiesmaier, A. Heinemann, and T. Grasmeyer, "On the state of crypto-agility." Cryptology ePrint Archive, Paper 2023/487, 2023, https://eprint.iacr.org/2023/487

[28] E. Barker, L. Chen, D. Cooper, D. Moody, A. Regenscheid, and M. Souppaya, "Considerations for Achieving Crypto Agility." NIST CSWP 39 2pd, July 2025, DOI 10.6028/NIST.CSWP.39.2pd

[29] D. Flo, M. Pala, and B. Hale, "Terminology for Post-Quantum Traditional Hybrid Schemes." RFC-9794, June 2025, DOI 10.17487/RFC9794

[30] M. Ounsworth, J. Gray, M. Pala, J. Klaußner, and S. Fluhrer, "Composite ML-DSA for use in X.509 Public Key Infrastructure." Internet-Draft, IETF, October 2025, https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/12/

[31] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard." NIST FIPS 204, August 13, 2024, DOI 10.6028/NIST.FIPS.204

[32] D. Micciancio and O. Regev, "Lattice-based Cryptography", Post-Quantum Cryptography (D. J. Bernstein, J. Buchmann, and E. Dahmen, eds.), pp. 147–191, Springer, 2009, DOI 10.1007/978-3-540-88702-7_5

[33] R. Overbeck and N. Sendrier, "Code-based Cryptography", Post-Quantum Cryptography (D. J. Bernstein, J. Buchmann, and E. Dahmen, eds.), pp. 95–145, Springer, 2009, DOI 10.1007/978-3-540-88702-7_4

[34] J. Ding and B.-Y. Yang, "Multivariate Public Key Cryptography", Post-Quantum Cryptography (D. J. Bernstein, J. Buchmann, and E. Dahmen, eds.), pp. 193–241, Springer, 2009, DOI 10.1007/978-3-540-88702-7_6

[35] C. Dods, N. P. Smart, and M. Stam, "Hash based digital signature schemes", Cryptography and Coding (N. P. Smart, ed.), Berlin, Heidelberg, 2005, pp. 96–115, DOI 10.1007/11586821_8

[36] C. Peng, J. Chen, S. Zeadally, and D. He, "Isogeny-based cryptography: A promising post-quantum technique", IT Professional, vol. 21, no. 6, 2019, pp. 27–32, DOI 10.1109/MITP.2019.2943136

[37] N. I. of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism (FIPS 203)." NIST FIPS 203, August 13, 2024, DOI 10.6028/NIST.FIPS.203

[38] N. I. of Standards and Technology, "Module-Lattice-Based Digital Signature Standard (FIPS 204)." NIST FIPS 204, August 13, 2024, DOI 10.6028/NIST.FIPS.204
[39] N. I. of Standards and Technology, "Hamming Quasi-Cyclic Key Encapsulation Mechanism (FIPS 205)." NIST FIPS 205, August 13, 2024, DOI 10.6028/NIST.FIPS.205
[40] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme." RFC-8391, May 2018, DOI 10.17487/RFC8391
[41] Y. Wang, R. Paccagnella, E. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86", Proceedings of the USENIX Security Symposium (USENIX), 2022. https://www.hertzbleed.com/hertzbleed.pdf
[42] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Hutchinson, A. Jalali, K. Karabina, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, *et al.*, "Supersingular Isogeny Key Encapsulation (SIKE) Specification." NIST PQC Submission, September 2022, https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/SIKE-spec.pdf
[43] G. Alagic, M. Bros, P. Ciadoux, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, H. Silberg, D. Smith-Tone, and N. Waller, "Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process." NIST IR 8545, March 2025, DOI 10.6028/NIST.IR.8545
[44] S. Crocker, D. D. D. Clark, R. T. Braden, and C. Huitema, "Report of IAB Workshop on Security in the Internet Architecture - February 8-10, 1994." RFC-1636, June 1994, DOI 10.17487/RFC1636
[45] S. Kent, "IP Authentication Header." RFC-4302, December 2005, DOI 10.17487/RFC4302
[46] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol." RFC-4306, December 2005, DOI 10.17487/RFC4306
[47] P. Wouters, D. Migault, J. P. Mattsson, Y. Nir, and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)." RFC-8221, October 2017, DOI 10.17487/RFC8221
[48] D. Carrel and D. Harkins, "The Internet Key Exchange (IKE)." RFC-2409, November 1998, DOI 10.17487/RFC2409
[49] J. Turner, M. J. Schertler, M. S. Schneider, and D. Maughan, "Internet Security Association and Key Management Protocol (ISAKMP)." RFC-2408, November 1998, DOI 10.17487/RFC2408
[50] H. Orman, "The OAKLEY Key Determination Protocol." RFC-2412, November 1998, DOI 10.17487/RFC2412
[51] V. Smyslov, "Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation." RFC-7383, November 2014, DOI 10.17487/RFC7383
[52] R. Jenwar, D. Hui, H. Tschofenig, and Y. Nir, "A Childless Initiation of the Internet Key Exchange Version 2 (IKEv2) Security Association (SA)." RFC-6023, October 2010, DOI 10.17487/RFC6023
[53] strongSwan, https://strongswan.org/
[54] Open Policy Agent, https://www.openpolicyagent.org/
[55] R. K. Zhao, N. H. Sultan, P. Yialeloglou, D. Liu, D. Liebowitz, and J. Pieprzyk, "MIKA: A Minimalist Approach to Hybrid Key Exchange", 2024 21st Annual International Conference on Privacy, Security and Trust (PST), 2024, pp. 1–11, DOI 10.1109/PST62714.2024.10788054
[56] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework", Computer Standards & Interfaces, vol. 66, 2019, p. 103357, DOI 10.1016/j.csi.2019.103357
[57] R. Marin-Lopez, G. Lopez, and F. Pereniguez-Garcia, "A YANG Data Model for IPsec Flow Protection Based on Software-Defined Networking (SDN)." RFC-9061, July 2021, DOI 10.17487/RFC9061
[58] F. Ciravegna, G. Bruno, and A. Lioy, "IKE-less IPsec for Centralized Management of Network Security", CEUR Workshop Proceedings, 2024, pp. 1–13. https://ceur-ws.org/Vol-3731/paper35.pdf
[59] M. D. Baushke, "Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)." RFC-9142, January 2022, DOI 10.17487/RFC9142
[60] Prometheus, https://prometheus.io/

[61] Grafana, https://grafana.com/
[62] The OpenSSL project, http://www.openssl.org/

Appendix A

User Manual

A.1 System setup

This appendix describes how to install, configure, and operate the prototype developed for this thesis. The system is composed of a set of containerised services that collectively implement a policy-driven secure network based on `strongSwan` for IPsec tunnelling and the OPA as PDP. The deployment includes the policy enforcement gateway (`pep-gateway`) running `strongSwan` with the `ext-auth` plugin, one or more initiator hosts (e.g. `host-legacy`), application servers such as `banka`, auxiliary peers, and a suite of observability components including Prometheus, Grafana, and a dedicated decision-logger service.

A.1.1 Software prerequisites

Docker engine and Docker compose

Install Docker using the official repository. The following commands configure the repository, import Docker's signing key, and install both `docker-ce` and the Docker Compose plugin:

```
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
    -o /etc/apt/keyrings/docker.asc
sudo install -m 0644 /etc/apt/keyrings/docker.asc \
    /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.asc] \
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo \"$VERSION_CODENAME\") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin
```

Running Docker without `sudo`

To execute Docker as an unprivileged user, add your account to the `docker` group:

```
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Verify the installation:

```
docker run hello-world
```

Networking tools

Tools such as `tcpdump` and `tshark` are required for packet-level debugging and for later performance evaluation:

```
sudo apt update
sudo apt install tcpdump tshark
```

Project files

Clone the project repository:

```
git clone https://github.com/PQ-IPsec-Gateway/thesis.git <DIR>
cd <DIR>
```

The repository includes a `docker-compose.yml` file defining all components: the decision-logger, CTI service, OPA , Prometheus, Grafana, the policy-enforcing gateway, the various hosts, and auxiliary peers. Each service contains its own configuration directory, which provides the `strongSwan` and `swanctl` profiles for that container, along with certificates and daemon's scripts.

A.2 Building and starting the system

A.2.1 Building Docker images

From the project root directory, build all service images:

```
docker compose build
```

This command reconstructs all Docker images according to the definitions in `docker-compose.yml`. Rebuilding is required whenever Dockerfiles or build contexts change.

A.2.2 Launching the environment

Start all services in detached mode:

```
docker compose up -d
```

List running services:

```
docker compose ps
```

Stop or tear down the environment:

```
docker compose stop
docker compose down
```

A.2.3 Inspecting container status

To confirm correct initialisation:

```
docker compose logs <service>
```

Check that all essential components are active.

A.3 Loading strongSwan configuration

Each `strongSwan`-based component uses `swanctl` to load IKE/CHILD SA configurations and credentials via the VICI interface.

A.3.1 Loading configurations and credentials

Run the following inside each `strongSwan` container:

```
docker exec <container> swanctl --load-all
```

A.3.2 Listing available connection profiles

```
docker exec <container> swanctl --list-conns
```

This command outputs all configured IKE and CHILD connection definitions, such as `legacy-to-gateway`.

A.4 Establishing VPN tunnels

A.4.1 Initiating the IKE SA

As an outbound example, from the `host-legacy` container, initiate the IKE negotiation:

```
docker exec host-legacy swanctl --initiate --ike legacy-to-gateway
```

A.4.2 Inspecting Security Associations

Check established IKE and CHILD SAs:

```
docker exec host-legacy swanctl --list-sas
docker exec pep-gateway swanctl --list-sas
docker exec banka swanctl --list-sas
```

Entries should appear as `ESTABLISHED` with valid SPI pairs.

A.4.3 Testing connectivity

Verify the encrypted path:

```
docker exec host-legacy ping -c 4 198.51.100.10
```

ESP traffic can be inspected using `tcpdump` or `tshark`.

A.5 Inspecting logs and OPA decisions

A.5.1 Gateway authorisation logs

The gateway's `ext-auth` module delegates authorisation to OPA. View authorisation events:

```
docker exec pep-gateway tail -f /var/log/ext-auth.log
```

A.5.2 Child SA installation logs

`strongSwan` logs VICI-triggered CHILD SA installations:

```
docker exec pep-gateway \
  grep "VICI initiate" /var/log/vici-child-manager.log
```

A.5.3 IKE SA denial with vendor-specific notification

To trigger a policy denial from an unauthorised peer, initiate an IKE connection from `partnerb` as follows:

```
docker exec partnerb swanctl --initiate --ike partnerb-responder
```

In this scenario, the gateway rejects the `IKE_AUTH` request and returns an additional vendor-specific notification payload that encodes both the denial reason and the associated security-level classification.

To inspect the resulting notification on the peer side, run:

```
docker exec banka sh -c "grep -A5 -B5 'A0 01' \
    /var/log/charon.log | grep '<partnerb-responder|'"
```

A.5.4 OPA audit logs

The decision-logger stores all PDP decisions. Logs are available under:

```
docker exec decision-logger \
    tail -f /app/logs/decisions-YYYY-MM-DD.log
```

A.6 Visualising metrics with Grafana

Grafana is provided for real-time observability. After deployment, access: `http://localhost:3000`. The default credentials are `admin/pqgw_admin`. Import dashboards via:

Dashboards → *Import* → *Upload JSON*.

Dashboards display `strongSwan` metrics, OPA decision statistics and network traffic counters.

A.7 Testing and troubleshooting

A.7.1 Performance Tests

The performance experiments described in this thesis are each associated with a dedicated branch of the project repository, namely `test-classic`, `test-ike-fragmentation` and `test-benchmark`. Each branch contains a minimally adapted variant of the system, together with lightweight instrumentation and helper scripts, allowing the corresponding test campaigns to be executed in a controlled and reproducible manner.

`test-classic`

The experiment is launched with:

```
bash scripts/simple-ike-comparison.sh
```

Internally, the script deploys the appropriate `swanctl` configuration, restarts the relevant containers, and reloads the `strongSwan` settings before triggering the handshake. A packet capture is started on the gateway, and the resulting PCAP is processed with a small Python script that extracts packet sizes, IKE message types, and IP fragmentation. The outputs from the normal and childless runs are then compared to quantify time-level and byte-level differences.

`test-ike-fragmentation`

This branch provides a tailored environment for analysing fragmentation behaviour and per-phase timing during the IKE exchange. The suite is executed through: `./scripts/capture-and-analyze-ike.sh pep-gateway gw-wan-out-bankA 198.51.100.2`

The script clears existing SAs, initiates a timestamped `tshark` capture on ports 500/4500, triggers the handshake, and subsequently parses the PCAP to recover message boundaries, intermediate ADDKE exchanges, and precise request/response timings. The output provides fine-grained measurements of per- phase latency and fragmentation, enabling comparisons between multi-KEM and single-KEM configurations.

`test-benchmark`

The most comprehensive evaluation is conducted in the `test-benchmark` branch, which correlates three independent data sources: raw PCAP captures, `strongSwan` VICI logs, and OPA audit records. The benchmark is executed using:

```
./scripts/run-benchmark.sh
```

The script resets the environment, and based on the level specified in the code, initiates a full tunnel establishment, captures all on-wire traffic, and records state transitions via VICI while OPA produces detailed timing reports for IKE and CHILD policy decisions. These heterogeneous data streams are then aligned to derive end-to-end timing, traffic volume, fragmentation counts, and policy-evaluation metrics, all of which are stored in structured JSON format for subsequent analysis.

A.7.2 Rebuilding and cleaning the environment

Rebuild images:

```
docker compose build --no-cache
```

Remove containers and volumes:

```
docker compose down --volumes
docker compose up -d
```

A.7.3 Diagnosing connection issues

- ensure each `strongSwan` container has reloaded its configuration via `swanctl --load-all`;
- inspect `/var/log/charon.log` for negotiation errors;
- verify that OPA policies authorise the requested connection;
- confirm certificate trust chains and identities match on both peers.

Appendix B

Developer's Reference Guide

This appendix describes the internal organisation of the thesis prototype from a developer's standpoint. It concentrates on the integration between `strongSwan` and `Open Policy Agent` in the `pep-gateway`, the custom patches applied to `strongSwan`, and the policy model that governs the creation and adaptation of tunnels.

The aim of this chapter is to equip a future maintainer with sufficient detail to understand where and how authorisation decisions are enforced, extend the deployment with additional legacy hosts or external peers, adjust the security levels and cryptographic algorithms enforced by the gateway and keep the local patches aligned with upstream `strongSwan`.

B.1 strongSwan–OPA integration

The policy enforcement point is realised in the `pep-gateway` container. From the perspective of `strongSwan`, the integration is built on three main elements:

1. a patched instance of the `ext_auth` plugin in `strongSwan`, implemented in `src/libcharon/plugins/ext_auth/ext_auth_listener.c`;
2. an helper, `opa-check-auth`, which bridges IKEv2's execution environment with the PDP;
3. auxiliary scripts and services inside `pep-gateway` (e.g. `vici-child-manager` and the `updown` hooks) that implement tunnel provisioning and re-provisioning according to OPA's decisions.

B.2 strongSwan `ext_auth` patch

B.2.1 File location and high-level responsibilities

The core of the C-level integration resides in the modified `ext_auth_listener` implementation:

- file: `src/libcharon/plugins/ext_auth/ext_auth_listener.c`;
- plugin: `ext_auth`, loaded by the IKE daemon `charon`.

The initial part of the patch extends the include set and introduces a small in-memory structure used to record pending OPA hints, indexed by the IKE `unique_id`:

Listing B.1. Additional includes and hint entry definition in `ext_auth_listener.c`

```
#include <stdlib.h>
```

```
#include <fcntl.h>
#include <string.h>
#include <ctype.h>

#include <utils/chunk.h>
#include <utils/enum.h>
#include <collections/linked_list.h>
#include <threading/mutex.h>
#include <encoding/message.h>
#include <crypto/proposal/proposal.h>
#include <crypto/transform.h>
#include <crypto/prfs/prf.h>
#include <crypto/key_exchange.h>
#include <credentials/certificates/certificate.h>
#include <credentials/auth_cfg.h>

typedef struct {
    uint32_t unique_id;
    char *required_ke;
    bool pending_notify;
} hint_entry_t;

#define NOTIFY_REQUIRED_LEVEL 0xA001

typedef struct private_ext_auth_listener_t private_ext_auth_listener_t;

struct private_ext_auth_listener_t {
    ext_auth_listener_t public;

    /** Path to authorization program */
    char *script;

    /** Pending hints (hint_entry_t*) keyed by ike unique_id */
    linked_list_t *hints;

    /** Protect access to hints */
    mutex_t *mutex;
};
```

The `hints` list keeps, for each IKE_SA (identified by `unique_id`), the pending *required level* value received from OPA and a boolean flag `pending_notify` which is later consulted when deciding whether to attach a custom `NOTIFY` payload to the outbound IKE_AUTH.

Regarding the peer certificate extraction, the helper function `acquire_peer_cert()` iterates over the current `auth_cfg` chain to retrieve the peer's end-entity certificate, which is then exported both inline (as PEM in an environment variable) and via a temporary file:

Listing B.2. Peer certificate acquisition from the current auth round

```
static certificate_t* acquire_peer_cert(ike_sa_t *ike_sa)
{
    enumerator_t *enumerator;
    auth_cfg_t *cfg;
    certificate_t *cert = NULL;

    enumerator = ike_sa->create_auth_cfg_enumerator(ike_sa, FALSE);
    if (!enumerator)
    {
```

```
        return NULL;
    }
    while (enumerator->enumerate(enumerator, &cfg))
    {
        cert = cfg->get(cfg, AUTH_RULE_SUBJECT_CERT);
        if (cert)
        {
            break;
        }
    }
    enumerator->destroy(enumerator);
    return cert;
}
```

Later in the authorisation path, this certificate is encoded as PEM and exposed to the external script:

Listing B.3. Exporting the peer certificate to the environment and a temporary file

```
peer_cert = acquire_peer_cert(ike_sa);
if (peer_cert && peer_cert->get_encoding(peer_cert, CERT_PEM, &pem))
{
    push_env(envp, countof(envp), "PLUTO_PEER_CERT_PEM=%.*s",
             (int)pem.len, pem.ptr);
    char template[] = "/tmp/ext-auth-peer-XXXXXX.pem";
    int fd = mkstemp(template);
    if (fd >= 0)
    {
        ssize_t written = write(fd, pem.ptr, pem.len);
        close(fd);
        if (written == (ssize_t)pem.len)
        {
            push_env(envp, countof(envp), "PLUTO_PEER_CERT=%s", template);
            tmp_cert_path = strdup(template);
        }
        else
        {
            unlink(template);
        }
    }
    chunk_free(&pem);
}
```

The variables `PLUTO_PEER_CERT_PEM` and `PLUTO_PEER_CERT` are then consumed by `opa-check-auth`, which embed the certificate into the JSON input passed to the PDP.

Moreover, the patch also enriches the authorisation environment with detailed information on the negotiated algorithms. This information is derived from the `proposal_t` associated with the IKE_SA:

Listing B.4. Exporting negotiated PRF and key exchange methods

```
proposal = ike_sa->get_proposal(ike_sa);
if (proposal)
{
    u_int idx;
    uint16_t alg = 0, key_size = 0;
    const char *name;
```

```
if (proposal->get_algorithm(proposal, PSEUDO_RANDOM_FUNCTION,
                            &alg, &key_size))
{
    (void)key_size;
    name = enum_to_name(pseudo_random_function_names, alg);
    if (name && *name)
    {
        push_env(envp, countof(envp), "PLUTO_IKE_PRF=%s", name);
    }
}
if (proposal->get_algorithm(proposal, KEY_EXCHANGE_METHOD,
                            &alg, &key_size))
{
    (void)key_size;
    name = enum_to_name(key_exchange_method_names, alg);
    if (name && *name)
    {
        push_env(envp, countof(envp), "PLUTO_IKE_DH=%s", name);
    }
}
for (idx = 0; idx < 3; idx++)
{
    transform_type_t type = ADDITIONAL_KEY_EXCHANGE_1 + idx;
    if (proposal->get_algorithm(proposal, type, &alg, &key_size))
    {
        (void)key_size;
        name = enum_to_name(key_exchange_method_names, alg);
        if (name && *name)
        {
            push_env(envp, countof(envp),
                    "PLUTO_IKE_KE%d=%s", idx + 1, name);
```

Consequently, each authorisation decision receives an environment containing:

- IKE identities (`IKE_LOCAL_ID`, `IKE_REMOTE_ID`);
- the negotiated PRF (`PLUTO_IKE_PRF`);
- the negotiated key exchange / DH group (`PLUTO_IKE_DH`);
- up to three additional key exchange identifiers (`PLUTO_IKE_KE1`, `PLUTO_IKE_KE2`, `PLUTO_IKE_KE3`);
- the peer certificate in PEM form.

Informations used by OPA to compute the security level, as described in the policy modules.

B.2.2 OPA hints and `REQUIRED_LEVEL` notify

The exchange between OPA and the gateway is intentionally richer than a simple boolean *allow/-deny* outcome. The helper script may also emit an "OPA hint" indicating the *required level* for a given IKE_SA. The patched `ext_auth` listener parses this hint and stores it in a `hint_entry_t` structure:

Listing B.5. Parsing OPA hints from the helper script

```
char *hint = strstr(resp, "OPA_HINT");
if (hint)
{
    uint32_t hint_id = 0;
    hint_level[0] = '\0';
    if (sscanf(hint, "OPA_HINT unique_id=%u required_ke=%31s",
               &hint_id, hint_level) >= 2)
    {
        DBG1(DBG_CHD, "ext-auth: parsed OPA hint unique_id=%u (local=%u) level
             '%s'",
             hint_id, unique_id, hint_level);
        if (hint_id == unique_id)
        {
            DBG1(DBG_CHD, "ext-auth: parsed hint for IKE_SA %u (hint=%u) level
                 '%s'",
                 unique_id, hint_id, hint_level);
            store_required_ke(this, hint_id, hint_level);
        }
    }
}
```

If the script returns a failure code (i.e. the policy engine rejects the current level), the listener marks the hint as pending and defers the actual notification to the *next* outbound IKE_AUTH message:

Listing B.6. Scheduling and attaching the REQUIRED_LEVEL notify

```
if (*success)
{
    clear_hint(this, unique_id);
}
else
{
    set_pending_notify(this, unique_id, TRUE);
}

/* ... */

METHOD(listener_t, message, bool,
    private_ext_auth_listener_t *this, ike_sa_t *ike_sa, message_t *message,
    bool incoming, bool plain)
{
    uint32_t unique_id = ike_sa->get_unique_id(ike_sa);

    if (!plain)
    {
        return TRUE;
    }

    if (incoming && message->get_exchange_type(message) == IKE_AUTH)
    {
        /* incoming REQUIRED_LEVEL notify: store the required level */
        notify_payload_t *notify = message->get_notify(message,
                                                       NOTIFY_REQUIRED_LEVEL);
        /* parse value and call store_required_ke(...) */
        /* ... */
        return TRUE;
```

```
    }

    if (!incoming && message->get_exchange_type(message) == IKE_AUTH)
    {
        hint_entry_t *entry;

        this->mutex->lock(this->mutex);
        entry = find_hint(this, unique_id);
        if (entry && entry->pending_notify && entry->required_ke &&
            entry->required_ke[0])
        {
            chunk_t payload = chunk_create(entry->required_ke,
                                           strlen(entry->required_ke));
            DBG1(DBG_CHD, "ext-auth: attaching required level '%s' for IKE_SA %
                u",
                 entry->required_ke, unique_id);
            message->add_notify(message, FALSE,
                                NOTIFY_REQUIRED_LEVEL, payload);
            this->hints->remove(this->hints, entry, NULL);
            this->mutex->unlock(this->mutex);
            destroy_hint_entry(entry);
            return TRUE;
        }
        this->mutex->unlock(this->mutex);
    }
    return TRUE;
}
```

The constructor for the `ext_auth` listener is extended to allocate and initialise both the hint list and its mutex:

Listing B.7. Listener creation with hint list initialization

```
METHOD(ext_auth_listener_t, create, ext_auth_listener_t*,
    char *script)
{
    private_ext_auth_listener_t *this;

    INIT(this,
        .public = {
            .listener = {
                .authorize = _authorize,
                .message = _message,
            },
            .destroy = _destroy,
        },
        .script = script,
        .hints = linked_list_create(),
        .mutex = mutex_create(MUTEX_TYPE_DEFAULT),
    );

    if (!this->hints || !this->mutex)
    {
        if (this->hints)
        {
            this->hints->destroy(this->hints);
        }
```

```
        if (this->mutex)
        {
            this->mutex->destroy(this->mutex);
        }
        free(this);
        return NULL;
    }

    return &this->public;
}
```

The corresponding `destroy` method is responsible for releasing any remaining hint entries and deallocating the synchronisation primitives.

B.3 `opa-check-auth` and helper scripts

B.3.1 Authorisation script interface

The `opa-check-auth` script is called by `ext_auth` with the environment described above. From a developer's viewpoint, the essential contract is:

- exit code `0` ⇒ authorisation *granted*;
- non-zero exit code ⇒ authorisation *denied*;
- optional diagnostic and hint lines printed on `stdout`, for example:

Listing B.8. Example output from `opa-check-auth`

```
OPA_DECISION allow=true level="L2"
OPA_HINT unique_id=42 required_ke=L3 required_sig=L2
```

The `OPA_HINT` line is parsed by the `ext_auth` patch as shown in Listing B.5, eventually leading to a `REQUIRED_LEVEL` notify being attached to an outbound IKE_AUTH.

B.4 Connection naming and tunnel provisioning

B.4.1 Naming conventions for connections and hosts

To simplify routing and provisioning logic, connection names in the `strongSwan` configuration are chosen with specific semantics rather than arbitrarily. The project adopts the following conventions:

- connections between the legacy host and the gateway explicitly contain the string `"legacy"` in their name;
- connections between the gateway and external peers explicitly encode the direction using suffixes such as `"outbound"` or `"inbound"`.

These conventions are consumed by:

- the `tunnel-provisioning` logic, which use this information, and also the knowledge of the traffic selectors, to understand that, there is an opening for an outbound connection from the legacy subnet and provide a "just-in-time" post-quantum tunnel counterpart;

- the policy layer, which use this information as additional helper to map connection names and peer identifiers to service classes and security levels.

Moreover, thanks to this technique, due to encoding semantic information directly into connection names (e.g. legacy vs. peer, inbound vs. outbound), the provisioning logic avoids relying exclusively on IP addressing to determine the role and direction of each tunnel.

B.5 OPA policy modules

The core IKE establishment policy logic is expressed in four Rego modules under the `ike` namespace:

- `ike.service_classes` (`service_classes.rego`);
- `ike.establishment` (`ike_establishment.rego`);
- `ike.child_templates` (`child_templates.rego`);
- `ike.certificates` (`certificate_validation.rego`).

Collectively, these modules:

1. classify internal subnets and external partners into *service classes*;
2. define the required IKE key exchange level (`KE-Lx`) for each internal-external pair;
3. choose the appropriate CHILD_SA template;
4. validate the peer certificate according to post-quantum signature and public-key requirements.

B.5.1 `service_classes.rego`

The module `ike.service_classes` defines a declarative mapping from IP prefixes and peers to service classes and cryptographic requirements:

Listing B.9. Top-level mapping in `service_classes.rego`

```
package ike.service_classes

import rego.v1

# INTERNAL SERVICE CLASSES - Subnet-based Classification

internal_service_classes := {
    "legacy_internal": {
        "subnet": "10.200.0.0/24",
        "security_profile": "legacy",
        "crypto_capabilities": ["RSA-2048", "ECDH-P256", "ECDSA-P256"],
        "description": "Internal legacy hosts using traditional cryptography",
    },
    ...
}

external_partner_classes := {
    "bankA": {
        "subnets": ["198.51.100.10/32"],
```

```
        "service_type": "payments",
        "min_ke_level": "KE-L3",
        "min_sig_level": "SIG-L3",
        "min_pubkey_level": "SIG-L3",
        "allowed_issuers": {
            "C=CH, O=Cyber, CN=Cyber Root CA",
            "O=PQ-Gateway Lab, CN=PQ-Gateway IKE CA PQ",
            ...
        },
        "description": "External bank A (payments, high assurance)"
    },
    ...
}
```

Instead of listing every internal host explicitly, internal classes are defined at the level of *subnets* (for instance, `10.200.0.0/24` for legacy systems). For each external partner (e.g. `bankA`, `partnerB`, `opsC`) then specifies:

- one or more IP prefixes;
- a `service_type` (for example *payments*, *erp*, *hr*);
- minimum levels `min_ke_level`, `min_sig_level`, `min_pubkey_level`;
- a set of `allowed_issuers` (distinguished names of trusted CAs).

B.5.2 `ike_establishment.rego`

The `ike.establishment` module is the main decision point. It imports the other rego modules, to enrich the decision, maintain modularity and fast changes.

Listing B.10. Imports in `ike_establishment.rego`

```
package ike.establishment

import rego.v1
import data.ike.certificates
import data.ike.child_templates
import data.ike.peer_mapping
import data.ike.service_classes
```

Mapping IKE suites to KE levels For each IKE_SA, a canonical string representing the full suite is constructed and used as a key into the `ke_suites` mapping:

Listing B.11. KE suite mapping in `ike_establishment.rego`

```
# Each KE level is defined by EXACT suite combinations, not individual
    algorithms

ke_suites := {
    # KE-L1 Suites (NIST Level 1 - HR, low-sensitivity)
    "aes128gcm16-sha256-ecp256-mlkem512-none-none": "KE-L1",
    "aes128gcm16-sha256-modp2048-mlkem512-none-none": "KE-L1",

    # KE-L2 Suites (NIST Level 3 - ERP, moderate)
    "aes192gcm16-sha384-ecp256-mlkem768-none-none": "KE-L2",
```

```
    "aes192gcm16-sha384-ecp384-mlkem768-none-none": "KE-L2",
    "aes192gcm16-sha384-x25519-mlkem768-none-none": "KE-L2",

    # KE-L3 Suites (NIST Level 3+ - Payments, enhanced classic)
    "aes256gcm16-sha512-ecp384-mlkem768-none-none": "KE-L3",
    "aes256gcm16-sha512-ecp521-mlkem768-none-none": "KE-L3",
    ...
}
```

A specific helper function, `build_suite_string(input.ike)`, then concatenates the negotiated algorithms into the string used to query `ke_suites`, leading to the *achieved* level, `ke_achieved`.

Partner-specific templates For each partner and level, fully specified templates are provided.

Listing B.12. Example inbound template for `partnerB`

```
responder_templates := {
    "partnerB": {
        "KE-L2": {
            "name": "inbound-legacy-erp-L2",
            "local_ts": "10.200.0.0/24",
            "remote_ts": "198.51.100.11/32",
            "child_level": "CHILD-L2",
            "esp_proposals": child_security_levels["CHILD-L2"].esp_proposals,
            "rekey_time": "90s",
            "reqid": 103,
            "start_action": "none",
            "updown": "/usr/local/sbin/updown-verifier.sh",
            "description": "PartnerB inbound ERP tunnel (L2 security)",
        },
        "KE-L3": {
            "name": "inbound-legacy-erp-L3",
            ...
        },
    },
    ...
}
```

This architecture permits the introduction of additional partners or child levels without requiring alterations to the overall template configuration, hence enabling modifications to minor components without compromising the integrity of the whole system.

B.5.3 `certificate_validation.rego`

The `ike.certificates` module encodes the post-quantum requirements for X.509 certificates used during IKE exchanges.

The module defines mappings from OIDs to signature and public-key algorithms and their base levels:

Listing B.13. Example OID → algorithm/level mapping

```
sig_alg_base_level := {
    # Pure ML-DSA (NIST standardized)
    "mldsa44": "SIG-L1",
    "mldsa65": "SIG-L2",
```

```
    "mldsa87": "SIG-L3",

    # ML-DSA44 composites (NIST Level 2 - HR, low-sensitivity)
    "mldsa44-rsa2048-pss-sha256": "SIG-L1",
    "mldsa44-rsa2048-pkcs15-sha256": "SIG-L1",
    "mldsa44-ecdsa-p256-sha256": "SIG-L1",
    "mldsa44-ed25519-sha512": "SIG-L1-SUF",
    ...
}

sig_level_hierarchy := {
    "SIG-L0": 0,
    "SIG-L1": 1,
    "SIG-L1-SUF": 2,
    "SIG-L2": 3,
    "SIG-L3": 4,
    "SIG-L3-SUF": 5,
}
```

The functions `get_sig_alg(oid)` and `get_pubkey_alg(oid)` map the OIDs present in the certificate to canonical algorithm names.

Per-peer requirements (minimum signature and public-key levels) are derived from the previously described entries in `service_classes.external_partner_classes`. The module exposes, among others, the following helper:

Listing B.14. Peer requirements in `certificate_validation.rego`

```
get_cert_requirements(peer_name) := reqs if {
    partner := service_classes.external_partner_classes[peer_name]
    reqs := {
        "min_sig_level": partner.min_sig_level,
        "min_pubkey_level": partner.min_pubkey_level,
        "allowed_issuers": partner.allowed_issuers,
    }
}
```

Evolving the security model To adjust the *ordering of levels* or the algorithms associated with each level, it is sufficient to update:

- `ke_suites` and the KE level associations in `ike_establishment.rego`;
- `child_security_levels` and `ke_to_child_level` in `child_templates.rego`;
- `sig_alg_base_level`, `pubkey_base_level` and `sig_level_hierarchy` in `certificate_val.rego`.

The base policy logic that combines these levels remains unchanged. Only the configuration tables are modified, keeping the system extensible and relatively easy to maintain. This makes it possible to refine the security profile (e.g. by deprecating an algorithm or promoting a stronger suite) without touching the strongSwan patch or the control-plane scripts.

B.6 Applying patches and extending the system

B.6.1 Patch directory in `pep-gateway`

The `pep-gateway` container includes an internal `patches` directory which holds all modifications applied on top of the upstream `strongSwan` sources.

During the Docker image build process, these patches are applied to the `strongSwan` source tree before compilation.

To introduce a new C-level modification (e.g. extending the notifier behaviour or adjusting the exported environment), the recommended workflow is:

1. locate the source file to be modified within the container build context;
2. copy it to a "`_mod`" variant, apply the desired edits, and generate a unified diff;
3. store the resulting patch under `pep-gateway/patches`;
4. rebuild the Docker image so that the patch is applied during the build.

A generic pattern is:

Listing B.15. Generic patch creation workflow

```
cd /tmp
git clone --depth 1 --branch 6.0.0beta6 \
  https://github.com/strongswan/strongswan.git
cd strongswan

cp src/libcharon/plugins/ext_auth/ext_auth_listener.c \
   src/libcharon/plugins/ext_auth/ext_auth_listener_mod.c

# Apply your modifications to ext_auth_listener_mod.c

diff -Naur \
  src/libcharon/plugins/ext_auth/ext_auth_listener.c \
  src/libcharon/plugins/ext_auth/ext_auth_listener_mod.c \
  > ext-auth-listener.patch

mv ext-auth-listener.patch /path/to/pep-gateway/patches/

cd /tmp && rm -rf strongswan

# rebuild the Docker image to apply the patch
cd /path/to/pep-gateway
docker build -t pep-gateway
```

B.6.2 Adding new connections and hosts

From a configuration perspective, adding a new legacy subnet or external peer involves:

1. **defining the connection in strongSwan**, be compliant to the naming conventions;
2. **updating** `service_classes.rego` to:
   - register the new legacy subnet or partner;
   - configure its default and permissible security levels;
   - optionally define per-subnet and per-partner requirements.