INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

A quantum-safe Signature Scheme for IKEv2 based on Isogenies

Markus Jürgens

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Masterarbeit

A quantum-safe Signature Scheme for IKEv2 based on Isogenies

Markus Jürgens

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Sophia Grundner-Culemann
	Dr. Tobias Guggemos
Abgabetermin:	18. Januar 2021

Abstract

With the field of quantum computing emerging fast and as a result of technology enhancements, it only seems to be a matter of time, until quantum computers will be able to break a large number of cryptographic algorithms that secure the internet of today. Although quantum computers are currently not powerful enough to actually execute attacks on those cryptographic algorithms, now is the right time to think about how to prevent this potential future threat. Post-quantum cryptography is the field of research which deals with that kind of threat.

The IPSec protocol-suite is one of the protocols securing the internet and it is prone to be broken with the rise of sufficiently powerful quantum computers. Ongoing work exists to secure the initial key exchange of IPSec done by the Internet Key Exchange Protocol version 2 (IKEv2), from which all keys for further encrypted communication are derived. This prevents attackers to capture IPSec-packets in transit and use a quantum computer to decrypt those packages later. The authentication mechanism of IKEv2 nevertheless can be attacked by exploiting the incorporated non-quantum-resistant signature scheme. This enables an attacker to to claim a false identity when establishing an IPSec-connection.

This work aims to provide a version of IKEv2 extended by a signature scheme that is resistant to attacks driven by quantum computers. This signature scheme relies on the problem of finding isogenies between supersingular elliptic curves and is assumed to be quantum-resistant.

We give an introduction to isogeny based cryptography, including a brief explanation of the mathematical foundations it is based on. Further we discuss signature schemes based on isogenies which evolve from Unruh's transformation of non-interactive zero knowledge proofs.

The work contributes a detailed analysis of requirements for PQ-Signature IKEv2, to comply with constraints that exist within the IKEv2-protocol and also to ensure interoperability and backwards compatibility. The resulting protocol specification reflects the elaborated requirements and allows two peers that try to communicate via IPSec to authenticate in a quantum-safe manner. A proof-of-concept implementation based on the aforementioned protocol is proposed. Finally an extensive evaluation of the protocol design as well as of the implementation is given, discussing the advantages and limitations of the new approach.

Contents

1 Introduction

Ongoing research in the field of quantum computing tries to build a computer based on quantum physics, a so-called quantum computer. If these machines are powerful enough, they are able to solve some problems significantly faster than classic computers we have available today. We refer to this as achieving quantum supremacy. It is not yet possible to build a quantum computer as several problems still exist which need to be mitigated. A quantum computer operates on *qbits* which is the equivalent of a bit of a classic computer with the difference that qbits can be in a superposition state. This means they are not 0 or 1 but "somewhere inbetween". Researchers laid out seven stages in the process of building a powerful enough to reach quantum supremacy quantum computer in [DS13] as shown in Figure 1.1 (QND in the third step is the abbreviation of "quantum non-demolition"). The arrow directing from the third to the fourth stage indicates the current state of research towards a quantum computer. It is currently possible to build quantum computers reaching the third stage and development is done towards reaching the further stages ([CMS$^+$15], [RPH$^+$15], [KBF$^+$15]). Although nobody can tell when the first large-scale quantum computer will be available, it seems to be a matter of time until we get there.

While we expect quantum computers to bring advances in a variety of research fields such as Artificial Intelligence, it does pose a threat to the field of cryptography. In 1994 Peter Shor developed an algorithm that is capable of solving the Integer-Factorization Problem as well as the Discrete Logarithm Problem (DLP) in polynomial time, given a sufficiently powerful quantum computer [Sho94]. It poses a serious threat to cryptographic systems relying on either the Integer-Factorization Problem or the DLP, which is a majority of the Public Key Cryptographic Systems (PKCSs), including RSA, DSA/ECDSA and DH/ECDH.

Post-Quantum Cryptography is the field of research dealing with the threat to classic cryptography by quantum computers. It utilizes algorithms that run on classic computers

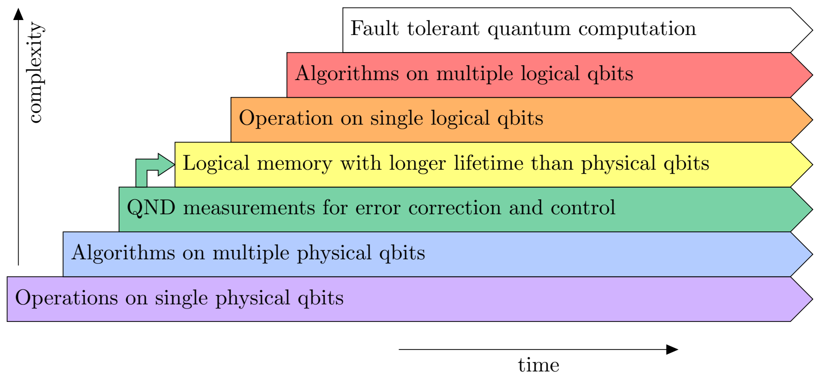
complexity
Fault tolerant quantum computation
Algorithms on multiple logical qbits
Operation on single logical qbits
Logical memory with longer lifetime than physical qbits
QND measurements for error correction and control
Algorithms on multiple physical qbits
Operations on single physical qbits
time

Figure 1.1: Seven stages in the development of quantum information processing according to [DS13]

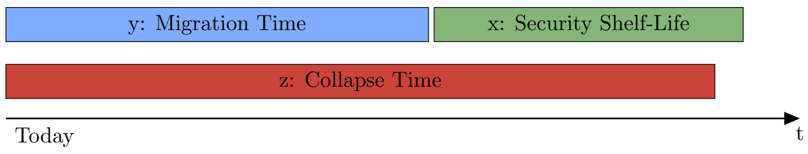

Figure 1.2: Evaluation of Necessity of Action

but rely on problems that are assumed to be hard to solve for quantum computers

Since we cannot say when the first sufficiently powerful quantum computer will be available, we need to evaluate the risk we are exposed to by using classic cryptography. Michele Mosca developed a framework that helps to decide, when it is time to take action and use post-quantum algorithms [Mos18], which is shown in Figure 1.2. The *Migration Time* y is the time we need to fully replace classic cryptosystems with quantum-safe cryptosystems. The *Security Shelf-Life* x is the time, we want the cryptosystem to hold out against attacks. In the context of encryption this means that the encrypted data can not be decrypted by an attacker for the next x years. The third parameter *Collapse Time* z is the time until our cryptosystems collapse due to the availability of a sufficiently powerful quantum computer. The question we need to answer to estimate the risk is if $x + y < z$. If this expression is true, there is no need for action right now. Otherwise, as indicated in the figure, where $z < x + y$, the point in time to start migrating to more secure cryptographic systems has already passed and the risk of the cryptographic system being broken during the timespan described by x increases. This risk framework of Mosca does of course not provide any guarantees as we need to estimate the collapse time z and the migration time y depends on the complexity of the operational environment. However the framework can guide in the process of figuring out when migration to quantum-safe cryptographic systems should be started.

One of the security protocols that is broken by the arrival of a sufficient powerful quantum computer is the IPSec protocol suite, as the initial key exchange heavily relies on PKCS and is prone to being broken by Shor's algorithm. Ongoing work exists to negotiate quantum-safe keys within IKEv2 ([Hei19] [TTB$^+$20]). This ensures confidentiality of IPSec communication by encrypting the payload with the quantum-resistant keys negotiated by IKEv2. Establishing quantum-resistant authenticity by incorporating a quantum-safe signature scheme in IKEv2 is not considered. This seems to be less critical because exploiting the authentication mechanism would require a quantum computer breaking authenticity in real-time. Nevertheless, as soon as quantum computers are available, the threat of exploiting the authentication mechanism of IKEv2 is clearly realistic. It would enable an attacker in possession of a quantum computer to execute a man-in-the-middle attack. When Alice wants to talk to Bob, the attacker can prove to Alice to own the identity of Bob and the other way round, by breaking the signature schemes used by the two peers. This work aims to secure the authentication mechanism of IKEv2 against an attacker utilizing a quantum computer.

To achieve this goal a closer look at a specific signature scheme was taken that relies on the problem of finding isogenies between supersingular elliptic curves. Isogeny-based cryptosystems generally incorporate short public keys, compared to other post-quantum cryptographic systems. And the problem of finding isogenies between supersingular elliptic curves is assumed to be hard for quantum computers to solve because of the mathematical complexity it is based on. Furthermore, isogeny-based cryptography offers a key-exchange as well as an encryption and a signing mechanism, making it a potential candidate to fully replace RSA.

Contribution

This work gives an introduction to isogeny-based cryptography including mathematical foundation. We provide an analysis of the requirements to integrate quantum-resistant authentication in IKEv2. Further we design an IKEv2 protocol extension that integrates quantum-safe authentication. We implement the isogeny-based signature scheme as proof-of-concept and evaluate the implementation. Finally we also evaluate the IKEv2 protocol incoporating quantum-resistant authentication.

Structure of this Work

The work is structured as follows: Chapter 2 will discuss the background and foundations, necessary for the integration of a quantum-safe authentication into IKEv2 as well as show some related work with regards to this topic. Afterwards, Chapter 3 analyses and states requirements for a successful integration. Further it shows how to evolve from an isogeny-based zero-knowledge proof towards a signature scheme. Chapter 4 develops the IKEv2 protocol extension by discussing possible approaches to comply with the previously stated requirements and choosing from those approaches to form the final protocol. Chapter 5 describes the implemented isogeny-based signature scheme and provides insight to an integration of the developed protocol into OpenBSD's iked. Chapter 6 evaluates the isogeny-based signature scheme with regards to performance and security as well as the security of the IKEv2 protocol extension. Finally Chapter 7 gives an outlook on future work and a conclusion of the work.

2 Background and Related Work

Before discussing how to integrate a isogeny-based signature scheme into IKEv2 to achieve a quantum-safe authentication method, we take a look at the beckground of this work and any other related works. In Section 2.1 the background is presented to provide an understanding of the technologies this work is based on. Section 2.2 shows other work which is related to the topic of integrating a quantum-safe signature scheme in IKEv2.

2.1 Background

This Section first describes the cryptographic basics of a Diffie-Hellman key exchange and of signature schemes in Sections 2.1.1 and 2.1.2. Then it discusses the IKEv2 protocol in Section 2.1.3 and finally isogeny-based cryptography in the following Sections 2.1.5 - 2.1.7.

2.1.1 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange is a widespread key exchange mechanism on which large parts of the internet rely on. It was originally proposed by Whitfield Diffie and Martin Hellman in [DH76].

Figure 2.1 shows the classic Diffie-Hellman key exchange. The communication partners (Alice and Bob) wish to generate a secret key that is negotiated over an unsecure communication channel. Therefore Alice and Bob need to agree on a prime p and a generator g. In the next step Alice chooses a secret number a and calculates $A = g^a \mod p$. Bob does the same with his own secret number b calculating $B = g^b \mod p$. Now they send those calculated values A and B over the unsecure communication channel. In the last step, Alice takes the B it received from Bob and calculates $K = B^a \mod p$. Bob does the same with the value A he received from Alice, ending up at the same K as Alice.

The security of the key exchange relies on the problem of finding K, knowing the prime p, the generator g as well as A and B. This problem is called the Diffie-Hellman problem. The fastest known way to solve this problem is to solve the discrete logarithm problem, which means finding a (b), knowing g and $A = g^a$ ($B = g^b$) [KK10, §9.1]. As problems relying on the discrete logarithm problem can be solved with Shor's algorithm [Sho94], those key exchanges do not hold up against attacks in which a quantum computer is involved.

Multiple variants of the Diffie-Hellman key exchange exist, one of them being Elliptic-curve Diffie-Hellman which is described in Section 2.1.4.

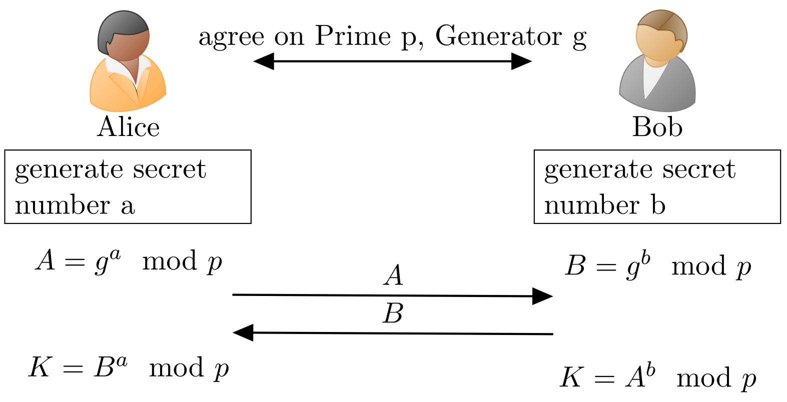

Figure 2.1: Classic Diffie-Hellman key exchange

2.1.2 Digital Signature

The purpose of digital signatures is to provide a method of signing digital data similar to a person signing a document in the non-digital world. By signing a message, the signing party (prover) proves to be the author of the message and the receiving party (verifier) can verify if the prover is the person he claims to be, and also if the message has been changed after the prover signed the data.

Schneier derives properties of digital signatures from the following properties of handwritten signatures as per the below [Sch07, §2.6]:

1. The signature is authentic. The signature convinces the document's recipient that the signer deliberately signed the document.
2. The signature is unforgeable. The signature is proof that the signer, and no one else, deliberately signed the document.
3. The signature is not reusable. The signature is part of the document; an unscrupulous person cannot move the signature to a different document.
4. The signed document is unalterable. After the document is signed, it cannot be altered.
5. The signature cannot be repudiated. The signature and the document are physical things. The signer cannot later claim that he or she didn't sign it.

These properties can be summed up in the below security goals with explanations from [Sch07, §1.1]. The first two points are covered by the goal *Authenticity*, point three and four are summed up in *Integrity*, while the fifth point ties in with the last goal which is *Non-Repudiation*.

Authenticity. It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else.

Integrity. It should be possible for the receiver of a message to verify that is has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one.

Non-Repudiation. A sender should not be able to falsely deny later that he sent a message.

Although signature schemes based on symmetric encryption exist, we focus on the more common signature schemes engaging asymmetric cryptography.

Signature schemes generally include three algorithms, the key generation, the signing and the verifying algorithm. Figure 2.2 shows the signing and the verification procedure, it does not depict the key generation algorithm. Following description explains all three algorithms involved in detail.

Key generation algorithm In the first step the peer that wishes to prove authorship of a message (prover) needs to generate a key pair, consisting of the signing key and a public key. It is essential, that the signing key is kept secret by the prover, whereas the public key has to be made publicly available. As long as the signing key is not compromised, the prover can reuse the key pair for signing digital data.

Signature generation algorithm In the second step, the prover uses his secret key to sign the message he wants to send to the verifier. Depending on the signature scheme, usually the message gets hashed before applying the secret key to the digest of the message. This increases the efficiency of calculating the signature and also ensures integrity of the message. The prover sends the computed signature together with the message.

Signature validation algorithm In the third step, the verifier computes the hash of the message he received. Then he uses the public key of the prover to verify the signature. If the public key applied to the signature matches the hash of the message, the verifier can be sure that the received message has been sent by the prover and has not been changed in transit.

The National Institute of Standards and Technology (NIST) standardized digital signatures in the Digital Signature Standard (DSS)[Bar13]. The NIST proposes three algorithms for signing messages, namely the DSA, RSA digital signature algorithm and the ECDSA. The DSA was published with the first version of DSS in 1994, whereas the NIST added the RSA digital signature algorithm and ECDSA in later publications of DSS. The latest final version dates to 2013, a new version from 2019 is currently available as draft. This draft version drops support for the DSA and adds support for the Edward-curves Digital Signature Algorithm (EdDSA). For detailed information on the specific signature algorithms, see [Bar13] for DSA and ECDSA and [MKJR16] for RSA.

2.1.3 Internet Key Exchange Protocol version 2 (IKEv2)

IKEv2 is a key exchange protocol enabling two peers to establish an IPSec connection [KS05] by dynamically negotiating cryptographic suits to ensure confidentiality, authenticity and integrity. IKEv2 is specified in [KHN$^+$14] and is the successor of IKEv1 [HC98]. The IPSec protocol suite ensures confidentiality, integrity, and authenticity of the transmitted data, and furthermore provides access control. An IPSec connection generally implements Security Associations (SAs) to establishing a secure channel for communication. A SA is a simplex logical connection between two peers, that protects the data to be transferred by the cryptographic suite it incorporates. It is the task of IKEv2 to negotiate the cryptographic suite and establish the SAs used for the IPSec communication.

IKEv2 is strictly based on request/response message pairs, meaning every request from the initiator expects a response from the responder. It is the requester's responsibility to ensure

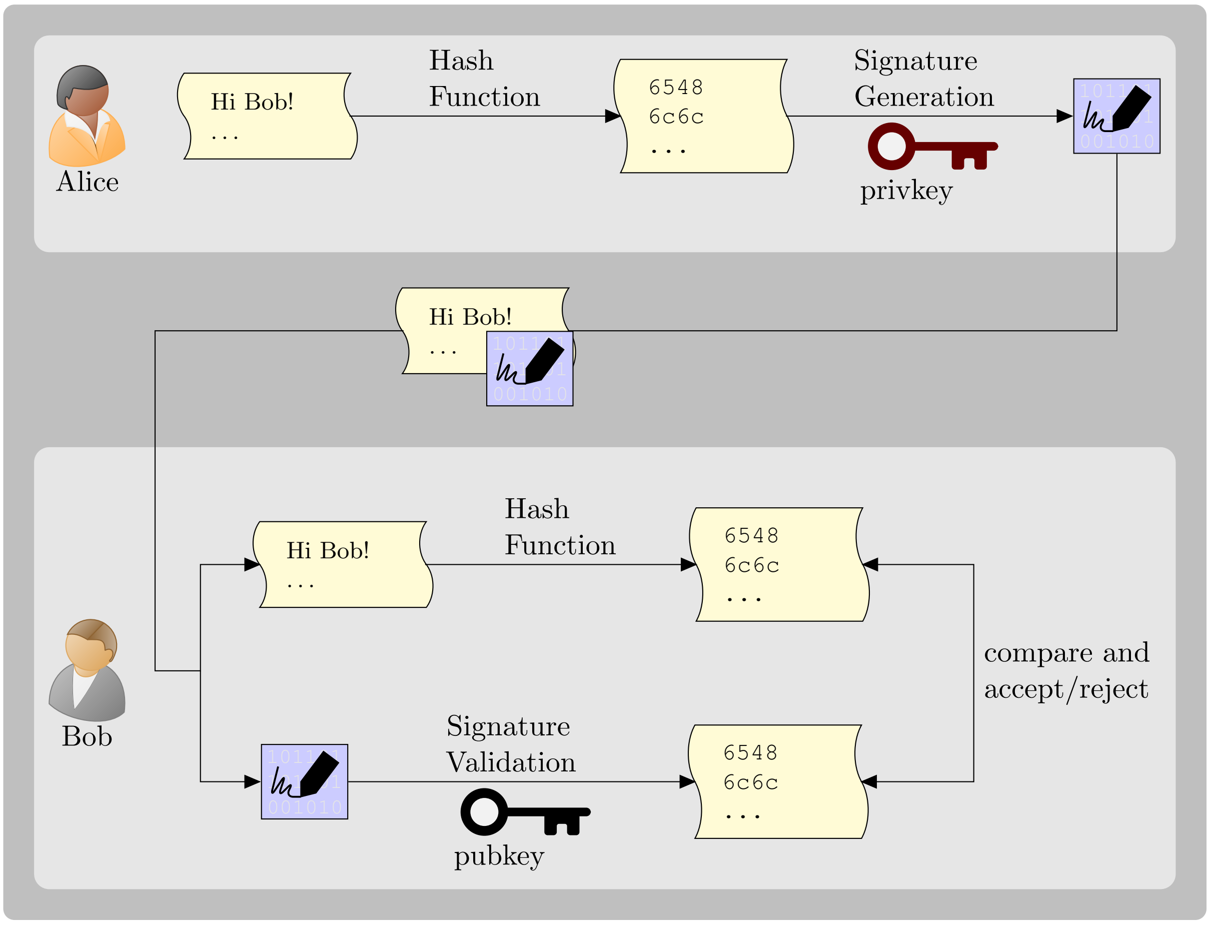

Figure 2.2: Digital Signature

```
Initiator                                               Responder
-----------------------------------------------------------------------------
HDR, SAi1, KEi, Ni -->
                                          <-- HDR, SAr1, KEr, Nr, [CERTREQ]
```

Figure 2.3: IKE_SA_INIT

reliability of the exchanged messages, i.e. if he does not receive a response to his request, he has to retransmit his request or abandon the connection.

IKE_SA_INIT

The first message pair in the IKEv2 protocol is the IKE_SA_INIT exchange, shown in Figure 2.3. The `HDR` field is the header of the message. Figure 2.4 shows the structure of the Header field. It contains the Security Parameter Index (SPI) of the initiator, allowing a clear assignment of the message to a SA. Further it contains information of the used version of IKE as well as the Exchange Type (IKE_SA_INIT), a message ID and some other information. Note that the responder's SPI is zero at this point in the protocol, as the initiator does not have any information of the SPI identifying a SA on the responder's side. `SAi1` contains the cryptographic algorithms which the initiator supports for establishing a SA. It consists of one or more proposals, each containing so called transforms. In the IKE_SA_INIT exchange these transforms are generally the Diffie-Hellman group, an integrity check algorithm, a Pseudo Random Function (PRF) algorithm and an encryption algorithm. `KEi` sends the Diffie-Hellman public value of the initiator and `Ni` contains its nonce. The nonce is a randomly generated bit string that is used later for the generation of SKEYSEED (described in the IKE_AUTH exchange). The responder answers again with the header `HDR` which now additionally contains the SPI of the responder. The responder chooses one of the proposals of cryptographic algorithms the initiator suggested and returns it in the `SAr1` payload. Further the responder sends his Diffie-Hellman public value in `KEr` and its nonce in `Nr`. The responder can optionally send the `CERTREQ` payload, providing the initiator with Certificate Authorities (CAs) he trusts.

IKE_AUTH

As a result of the IKE_SA_INIT exchange, the two parties have negotiated cryptographic parameters, exchanged nonces and did a DH key exchange. Both parties generate the SKEYSEED by applying the negotiated PRF to the concatenation of the nonces `{Ni|Nr}` and the shared secret from the Diffie-Hellman key exchange from the IKE_SA_INIT exchange. SKEYSEED is the starting point from which all other keys incorporated in IKEv2 are derived. From this point on, the communication partners can communicate over an encrypted channel, but authenticity is not yet ensured.

The next message exchange, namely the IKE_AUTH exchange ensures authenticity of the newly initialized SAs and is shown in Figure 2.5.

Again the `HDR` field is the generic header shown in Figure 2.4. The `SK {...}` payload encapsulates data that is encrypted by the sending SA with a key derived from the SKEYSEED

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IKE SA Initiator’s SPI                  |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       IKE SA Responder’s SPI                  |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Payload | MjVer | MnVer | Exchange Type |     Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Message ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Length                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.4: IKE Header format

```
Initiator                                         Responder
-----------------------------------------------------------------------------
HDR, SK {IDi, [CERT,] [CERTREQ,]
         [IDr,] AUTH, SAi2,
         TSi, TSr} -->
                                              <-- HDR, SK {IDr, [CERT,] AUTH,
                                                          SAr2, TSi, TSr}
```

Figure 2.5: IKE_AUTH exchange

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Auth Method   |                RESERVED                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                      Authentication Data                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.6: Authentication Payload format

Table 2.1: IKEv2 Authentication Method

Value	**Authentication Method**	**Reference**
0	Reserved	[KHN+14]
1	RSA Digital Signature	[KHN+14]
2	Shared Key Message Integrity Code	[KHN+14]
3	DSS Digital Signature	[KHN+14]
4-8	Unassigned	[KHN+14]
9	ECDSA with SHA-256 on the P-256 curve	[FS07]
10	ECDSA with SHA-384 on the P-384 curve	[FS07]
11	ECDSA with SHA-512 on the P-521 curve	[FS07]
12	Generic Secure Password Authentication Method	[Kiv11]
13	NULL Authentication	[SW15]
14	Digital Signature	[KS15]
15-200	Unassigned	
201-255	Private use	[KHN+14]

from the IKE_SA_INIT exchange. `IDi` is the identity the initiator claims. If the responder requested a certificate, the initiator adds one or more certificates in the `CERT` payload and eventually also requests certificates from the responder by adding the `CERTREQ` payload. The `IDr` payload is optional and specifies the responder's identity which the initiator wants to communicate with in case, the responding peer has more than one identity. Further the initiator proves his claimed `IDi` with the `AUTH` field. The `AUTH` payload is shown in Figure 2.6, where the `Auth Method` field specifies the method that is used for authentication and the `Authentication Data` is the actual signed data, that is sent to the other peer.

The `Auth Method` is one of the following values from Table 2.1, taken from [IAN].

The `Authentication Data` field is data that is computed according to the `Auth Method`. The input data on which the signature scheme computes the signature is the own IKE_SA_INIT message (if more than one IKE_SA_INIT has been exchanged its the last IKE_SA_INIT

```
Initiator                                             Responder
-------------------------------------------------------------------------------
HDR, SK {SA, Ni, [KEi,]
           TSi, TSr} -->
                                                      <-- HDR, SK {SA, Nr, [KEr,]
                                                                   TSi, TSr}
```

Figure 2.7: Creation of new Child SA with the CREATE_CHILD_SA exchange

message) concatenated with the other peers nonce `N_r/N_i` and `prf(SK_pi, IDi')` for the initiator and `prf(SK_pr, IDr')` for the responder. `SK_pi/SK_pr` is a key derived from the negotiated SKEYSEED by the initiator/responder. `IDi'/IDr'` is the entire `IDi/IDr` payload without the generic payload header. The `SAi2` payload is for negotiating the cryptographic algorithms just like the `SAi` payload in the IKE_SA_INIT request but this time for the first child SA that is established right after successful authentication. The child SAs are finally the SAs that transmit the user data in contrast to the previous established SA that has been established as a secure configuration channel for negotiating the child SAs. The `TSi/TSr` (Traffic Selector initiator/Traffic Selector responder) payload enables the communication partners to exchange information of their IPSec policy to the other peer. For more information, see RFC 7296 [KHN+14, §2.9].

The response from the responder again sends the `HDR` field and some encrypted data encapsulated in the `SK` payload. `IDr` is the identity the responder claims. `CERT` contains one or more certificates if the initiator requested so. `AUTH` is the Authentication payload already described above. The `SAr2` payload is the accepted offer of the algorithms received in the `SAi2` payload. If the responder agrees on the Traffic Selectors he received from the initiator, he sends identical values of `TSi` and `TSr` back to the initiator.

From this point on, the two peers have established a SA, created a Child SA and can communicate encrypted, ensuring confidentiality, integrity and authenticity.

The two exchanges IKE_SA_INIT and IKE_AUTH are often referred to as the "IKE Phase 1" or as the initial exchanges.

There are two more exchange types, namely the CREATE_CHILD_SA and the INFORMATIONAL exchange.

CREATE_CHILD_SA

The purpose of the CREATE_CHILD_SA exchange is to create new Child SAs or to rekey an already existing pair of SAs or Child SAs. Rekeying means creating a new pair of SAs and then deleting the old SAs. The exchange consists of one request initiated by one of the communication partners and one response sent by the responder.

Figure 2.7 shows the exchange for the creation of a new child SA. The `HDR` field is the generic IKEv2 header as shown in Figure 2.4. The `SK` payload again encapsulates the encrypted content. This consists of the cryptographic algorithms offer `SA`, a Nonce `Ni`, optional a Diffie-Hellman value in `KEi` and the Traffic Selectors for the new Child SA. The responder's message contains the `HDR` payload and the encrypted data, encapsulated in the `SK` payload. This

```
Initiator                                        Responder
----------------------------------------------------------------------------
HDR, SK {SA, Ni, KEi} -->
                                                 <-- HDR, SK {SA, Nr, KEr}
```

Figure 2.8: Rekeying SAs with the CREATE_CHILD_SA exchange

```
Initiator                                        Responder
----------------------------------------------------------------------------
HDR, SK {N(REKEY_SA), SA, Ni, [KEi,]
           TSi, TSr} -->
                                                 <-- HDR, SK {SA, Nr, [KEr,]
                                                              TSi, TSr}
```

Figure 2.9: Rekeying of a Child SA with the CREATE_CHILD_SA exchange

contains the accepted offer of cryptographic algorithms in `SA`, a Nonce `Nr`, a Diffie-Hellman value `KEr` and the Traffic Selectors. The `KEr` is only included, if the initiator included the `KEi` payload.

In case of the rekeying of SAs, the exchanged messages look slightly different as shown in Figure 2.8. The `SA` payload proposes cryptographic algorithms, and the `Ni` again contains a nonce. The `KEi` payload is now mandatory. Additionally the initiator provides a SPI in the SA payload. The responder accepts a cryptographic algorithm in `SA`, sends its nonce `Nr` and also sends a Diffie-Hellman value in `KEr`. In the `SA` payload the responder provides his new SPI.

The rekeying of Child SAs as shown in Figure 2.9 is identical to the creation of Child SAs, except the initiator has to include a NOTIFY payload `N(REKEY_SA)` indicating that an existing Child SAs should be rekeyed. The Child SA to be rekeyed is identified by the SPI in the `SA` payload.

INFORMATIONAL

The last exchange in IKEv2 is the INFORMATIONAL exchange allows to send `NOTIFY`, `DELETE` or `CONFIGURATION` payloads. It is also possible to send an INFORMATIONAL message without any payload. The receiver of an INFORMATIONAL message has to respond to the message, otherwise the initiator will consider the message to be lost in transit and resend it. This also enables one party to check if the other end is still alive by sending an empty INFORMATIONAL request expecting an empty INFORMATIONAL response. Note that the INFORMATIONAL exchange must happen after the initial exchanges, so the exchange can happen encrypted and with authenticity ensured. Figure 2.10 shows the exchange in detail.

```
Initiator                                                   Responder
-----------------------------------------------------------------------------
HDR, SK {[N,] [D,]
            [CP,]...} -->
                                                            <-- HDR, SK {[N,] [D,]
                                                                           [CP,]...}
```

Figure 2.10: INFORMATIONAL exchange

2.1.4 Elliptic Curve Cryptography

The idea to use elliptic curves in cryptographic systems has its origin in the independent work by Miller [Mil85] and Kobliz [Kob87]. Cryptographic suits exist that allow key-establishment (i.e. ECDH), signing (i.e. ECDSA) and encryption of messages incorporating elliptic curves. To obtain an understanding of ECC, we first take a look at the mathematical foundation. Afterwards we discuss the security and show the ECDH key-establishment as example of a crypto system based on elliptic curves. At the end we show the advantages of ECC over the earlier public-key cryptosystems like RSA and DSA.

Mathematical Foundation

Definition 2.1 (Elliptic Curve over a Finite Field)**.** An elliptic curve E over the finite field $\mathbb{F}_p$ (with characteristic $\neq 2, 3$) is defined by the set of points fulfilling the equation

$$y^2 = x^3 + ax + b, \qquad \text{where } a, b \in \mathbb{F}_p \qquad \text{and } -(4a^3 + 27b^2) \neq 0$$

The statement $-(4a^3 + 27b^2)$ is the discriminant of the curve E and if the expression is not equal to zero, we ensure that the curve is non-singular (for further details see [Sil09, Proposition 1.4 (a), §III.1]). In addition to the set of points fulfilling the above mentioned equation, we add the so-called *point-at-infinity*, denoted $\mathcal{O}$.

Definition 2.1 allows us to define an operation + on an elliptic curve E. This operation enables us to add two points $P, Q \in E$ on the curve E as follows:

Definition 2.2 (Composition Law [Sil09, §III.2])**.** Let $P, Q \in E$, let L be the line through P and Q (if $P = Q$, let L be the tangent line to E at P), and let R be the third point of intersection of L with E. Let L' be the line through R and $\mathcal{O}$. Then L' intersects E at $R, \mathcal{O}$, and a third point. We denote that third point by $P + Q$.

Figure 2.11 illustrates the addition of the points P and Q on the left half and the addition of the point P to itself on the right half.

Following are properties of the composition law showing that the Group (E,+) is an abelian group.

Proposition 2.3 (Properties of the composition law [Sil09, Proposition 2.2. §III.2])**.**

(a) If a line L intersects E at the (not necessarily distinct) points P, Q, R, then

$$(P + Q) + R = \mathcal{O}$$

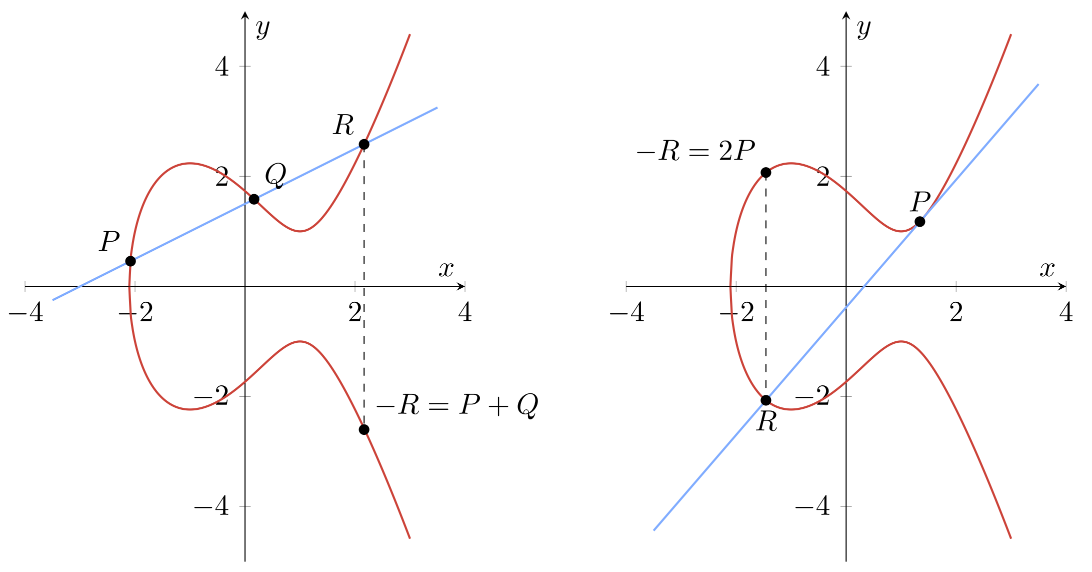

Figure 2.11: Addition of Points and Point-Doubling on an Elliptic Curve

(b) **Identity Element:** $P + \mathcal{O} = P$ for all $P \in E$.

(c) **Commutativity:** $P + Q = Q + P$ for all $P, Q \in E$.

(d) **Inverse Element:** Let $P \in E$. There is a point of E, denoted by $-P$, satisfying

$$P + (-P) = \mathcal{O}$$

.

(e) **Associativity:** Let $P, Q, R \in E$. Then

$$(P + Q) + R = P + (Q + R)$$

.

The proof for the proposition above can be found in [Sil09, §III.2] as it exceeds the scope of this work. The ability to add points on an elliptic curve E and the fact that $(E, +)$ is an abelian group enables us to also multiplicate a point on the curve by a constant value n by adding the point n-times to itself.

$$nP = \underbrace{P + ... + P}_{n\text{-times}}$$

Security Considerations

The security of elliptic curve cryptography is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) stating that it is hard to find n, knowing two points on the elliptic curve P and nP. The NIST provides ECC parameter sets for ECDSA in [Bar13] and for key-establishment schemes in [BCR+18] to ensure the security of the used elliptic curves.

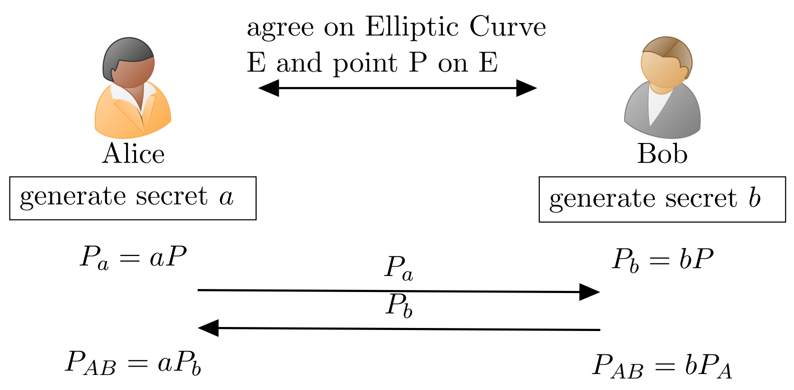

Figure 2.12: Elliptic-curve Diffie-Hellman key exchange

Example: ECDH

Figure 2.12 shows a Diffie-Hellman key-establishment mechanism as shown in Section 2.1.1 incorporating elliptic curves. Alice and Bob agree on an elliptic curve E and on a point P on E. The process of key generation remains the same as with the classic Diffie-Hellman, but instead of exponentiating the generator, both parties multiply the point P with their secret value a respectively b. The intermediate value (P_a/P_b) is again transferred to the other party, which again multiplies the received value with its own secret (a/b).

The security of ECDH relies on the problem to find a (b) from knowing E, P and P_a (P_b), which is the ECDLP as shown above. This problem can again be reduced to the discrete logarithm problem, meaning solving the discrete logarithm problem would reveal the secret factor a (b) and enable an attacker to compute P_{AB}. This is why ECDH does not provide additional security over the DH key exchange.

Advantages of ECC

Generally ECC does not add security to a crypto system by introducing a more complex problem. ECC relies on the hardness of the DLP, which can be broken by a quantum computer using Shor's algorithm in polynomial time. Nevertheless ECC does have several advantages over the early public-key cryptographic systems like RSA and DSA which are shown in the following description.

Key Size: The key sizes of ECC suits are not only significantly smaller than the keys of other public-key cryptographic systems but also scale linear with their security level in contrast to the superlinear growth of other cryptographic systems. NIST provides a table in [Bar20, §5.6] allowing a comparison of key lengths of early cryptographic suits and ECC cryptography at the same security level. They claim that a 256-bit key of ECC provides the same security level (128 bit) as a 3072-bit key of an early public-key system like DSA. This gap gets even bigger with higher security levels, where at a security level of 256 bit, ECC requires an 512-bit key whereas DSA would require a 15360-bit key. This makes ECC particularly suitable in environments with memory constraints. Additionally small key sizes reduce data sent over the network making

Table 2.2: comparison of public key sizes (in bytes) of NIST submissions

Security category	1	3	5
SIKE	330	462	564
BIKE-1	2541	4964	8188
BIKE-2	1270	2482	4094
BIKE-3	2757	5421	9033
McEliece	261120	524160	1044992
NewHope	928		1824

ECC suitable in environments with low bandwidth or lossy networks.

Efficiency: The efficiency of the computation is increased when using ECC systems. Although the metric of computational complexity per security bit is slightly higher for ECC suits, the smaller keys overcompensate this complexity resulting in an overall faster computation [cas]. This enables devices with weak computational capabilities to also use the scheme with acceptable time requirements.

In summary therefore, it can be said that ECC minimizes memory requirements as well as network traffic and computation time, all without weakening the cryptographic strength of the used system. This makes ECC suitable for a wide range of devices, especially for low-level devices as smart cards or IOT edge devices.

2.1.5 Isogeny-Based Cryptography

Isogeny-based cryptography is an approach to achieve quantum resistant cryptography using elliptic curve cryptography. Its cryptographic strength is a result of the difficulty to find an isogeny between two supersingular isogenous elliptic curves. The widest known implementation of isogeny-based cryptography is SIKE[sik], which is a candidate of the Post-Quantum Cryptography Standardization of NIST.

Isogeny-based cryptography was first introduced by Jao and De Feo [JDF11] in 2011, which makes it a very young approach compared to other cryptographic systems. As the past has shown, the maturity of a cryptographic system can only be proven by using it over a long period of time without it being broken by attackers. At the moment there is no known algorithm that can solve the problem of finding an explicit isogeny between two elliptic curves in sub-exponential time. Therefore isogeny-based cryptography is a promising candidate to hold up against classical and quantum computer attacks and thus, to establish quantum resistant cryptography.

Isogeny-based cryptography differentiates from other cryptographic systems by having significantly smaller public key sizes at the same level of security. Table 2.2 shows the public key sizes of some of the more popular NIST-submissions. The security category has been specified in the “Call for Proposals” published by NIST in 2016. If an approach aims for the security category one, the expense to break the system should be comparable to that one, breaking a block cipher with a 128-bit key. The expense to break a category three cryptographic system should be comparable to the computational resources needed for a key search on a block chipher with a 192-bit key. Finally to break a category five system, one should need at least the effort it takes for a key search on a block cipher with a 256-bit key[NIS16]. Comparing public key sizes shows that key sizes of BIKE are between four to 16 times bigger than SIKE public keys, McEliece keys are even bigger (factor ∼1800). The public key sizes of the NewHope system are closer to the ones of SIKE but still factor three to four bigger. Those large key sizes can lead to problems, transmitting them over the network. For example the maximal UDP packet size is 65535 bytes, which leaves 65507 bytes for the payload after subtracting 20 bytes for the IP header and 8 bytes for the UDP header [Pos80]. If the data to be sent exceeds the UDP packet size, it gets fragmented, which potentially leads to problems with network middleware. Furthermore, UDP makes no guarantees if and in what order packets reach their destination, so it is hard for the receiver to reassemble data it received in more than one UDP packet. The situation gets even worse if we look at IKEv2, where the maximum packet size IKEv2 implementations have to support is only 1280 bytes long[KHN+14, p. 24]. Isogeny-based cryptography enables the user to maintain key sizes that do hardly differ from the key sizes used in classical cryptography. Further IP fragmentation and problems arising from its use can be avoided.

2.1.6 Isogenies

To understand isogeny-based cryptography we need to have a look at the mathematical foundations it is based on. First we take a look at **isogenies**.

Definition 2.4 ([Sil09, III.4])**.** Let E_1 and E_2 be elliptic curves defined over $\mathbb{F}_q$. An isogeny from E_1 to E_2 is a homomorphism

$$\phi : E_1 \to E_2 \quad \text{satisfying} \quad \phi(\mathcal{O}) = \mathcal{O}.$$

Two elliptic curves E_1 and E_2 are *isogenous* if and only if there is an isogeny from E_1 to E_2 with $\phi(E_1) \neq \{\mathcal{O}\}$.

Following is the definition of the **ℓ-torsion-subgroup**.

Definition 2.5 ([Sil09, III.4])**.** Let E be an elliptic curve and let $\ell \in \mathbb{Z}$ with $\ell \geq 1$. The *ℓ-torsion-subgroup of E*, denoted by $E[\ell]$, is the set of points of E of order ℓ,

$$E[\ell] = \{P \in E : [\ell]P = \mathcal{O}\}$$

This means, there is a set of points on E that, multiplied with ℓ, result in the identity $\mathcal{O}$ of the elliptic curve. This set of points is called the *ℓ-torsion-subgroup*.

Further if we have a separable isogeny ϕ with kernel of size ℓ, we define the **degree of the isogeny** ϕ to be ℓ [Sil09, III.4.10c]. Additionally the kernel of the isogeny $\ker \phi$ with kernel size ℓ is a subgroup of the ℓ-torsion-subgroup of the elliptic curve E.

$$\ker \phi \subseteq E[\ell]$$

Proposition 2.6 ([Sil09, III.4.12])**.** Let E be an elliptic curve and let Φ be a finite subgroup of E. There are a unique elliptic curve E' and a separable isogeny

$$\phi : E \to E' \qquad \text{satisfying} \qquad \ker \phi = \Phi.$$

With the above proposition, we can ensure that for every subgroup $H \subseteq E[\ell]$ there is an up to isomorphism unique elliptic curve E' and a separable isogeny $\phi : E \to E'$ with $\ker \phi = H$. The phrase “up to isomorphism” means that there are more elliptic curves that satisfy given conditions, but those curves are all isomorphic. Vélu presents explicit formulas for the computation of E' [Sil09, Remark 4.13.3], [Vél71].

Under certain conditions, we can specify the ℓ-torsion-subgroup in greater detail. If p generating the finite subgroup $\mathbb{F}_q$ does not divide ℓ ($p \nmid \ell$) and $\ell \neq 0$ and $p > 0$, then we can denote the ℓ-torsion-subgroup $E[\ell]$ as follows [Sil09, III Corollary 6.4]:

$$E[\ell] = \mathbb{Z}/\ell\mathbb{Z} \times \mathbb{Z}/\ell\mathbb{Z}$$

Using that property, it can be shown that exactly $\ell + 1$ subgroups of $E[\ell]$ of order ℓ exist. Combining that with the previous shown proposition 2.6, we have:

Theorem 2.7. For every ℓ-torsion-subgroup $E[\ell]$ of an elliptic curve E defined over $\mathbb{F}_q$, where $p \nmid \ell$ and ℓ is prime we have exactly $\ell + 1$ up to isomorphism unique isogenies of degree ℓ.

Not every degree-ℓ isogeny is defined over $\mathbb{F}_q$ but over $\overline{\mathbb{F}_q}$, which would cause problems implementing such a cryptographic system, because we cannot predefine a fixed finite field $\mathbb{F}_q$. $\overline{\mathbb{F}_q}$ would have to be dynamically adapted, calculating the degree-ℓ isogenies. To prevent this situation, we use supersingular elliptic curves.

Definition 2.8 ([Sil09, V.3.1(a)(i)])**.** The ℓ-torsion-subgroup $E[\ell]$ of an elliptic curve $E/\mathbb{F}_q$ is either

1. $E[\ell] = \mathcal{O}$
2. $E[\ell] = \mathbb{Z}/\ell\mathbb{Z}$

In the first case we say, E is **supersingular**, in the second case, E is ordinary.

For $p \geq 5$ there exists a finite number of supersingular elliptic curves over $\overline{\mathbb{F}_q}$ which can be calculated as follows [Sil09, V.4.1(c)]:

$$\lfloor p/12 \rfloor + \begin{cases} 0 & if \quad p \equiv 1(\mod 12) \\ 1 & if \quad p \equiv 5(\mod 12) \\ 1 & if \quad p \equiv 7(\mod 12) \\ 2 & if \quad p \equiv 11(\mod 12) \end{cases} \tag{2.1}$$

This set of supersingular elliptic curves is called **isogeny class**.

According to the theorem [Sil09, V.3.1], giving equivalent definitions of supersingularity, an elliptic curve is supersingular if and only if the multiplication-by-ℓ map $[\ell] : E \to E$ is purely inseparable and $j(E) \in \mathbb{F}_{p^2}$. $j(E)$ is the **j-invariant** of the elliptic curve E, which is defined for elliptic curves in the Weierstrass notation as follows:

Definition 2.9.

$$j(E) = 1728 \frac{4a^3}{4a^3 + 27b^2}$$

If and only if two elliptic curves have the same j-invariant, those two curves are isomorphic. Since $j(E) \in \mathbb{F}_{p^2}$ in the supersingular case, we ensure, that every up to isomorphism unique elliptic curve is defined over $\mathbb{F}_{p^2}$. Further if $p^2 \mid q$, the elliptic curves are guaranteed to be defined over $\mathbb{F}_q$ instead of $\overline{\mathbb{F}_q}$. This is what simplifies the implementation of this cryptographic system without weakening its strength.

2.1.7 Isogeny Graphs

An isogeny graph contains elliptic curves represented by their j-invariants as nodes and isogenies to other elliptic curves are shown by edges connecting those two curves. Those edges are undirectional due to the existence of the dual isogeny [Sil09, III.6.1], which states that to every isogeny ϕ of degree ℓ that maps one elliptic curve E to another elliptic curve E' there is a dual isogeny ϕ' of degree ℓ that satisfies $\phi \circ \phi' = [\ell]$. Note that for a finite field of definition $\mathbb{F}_q$ a finite set of isomorphic elliptic curves exists (see Equation 2.1). This enables us to calculate every j-invariant of every up to isomorphism elliptic curve defined within the predefined finite field. For every kernel size ℓ we get a different isogeny graph, which preserves the nodes being the j-invariants, but has different edges representing isogenies. Note that as discussed in Theorem 2.7 every node has $\ell + 1$ edges. Figure 2.13 shows the 9 supersingular j-invariants of the finite field $\mathbb{R}_{109^2}$ and the 2-isogenies ($\ell = 2$) connecting those j-invariants.

An important property of isogeny graphs of supersingular elliptic curves is that they are ramanujan graphs, as proven by Pizer [Piz90][Piz98]. Ramanujan graphs are optimal expander graphs, i.e. they mix rapidly. That means, taking a relatively small amount of random steps in the graph can lead to any vertex in the graph.

This is an important property for cryptography, as one cannot restrict the set of potential destination vertices by knowing the starting vertex.

2.2 Related Work

This Section outlines work that is related to the integration of a isogeny-based signature scheme into IKEv2. First we will discuss how isogenies between supersingular elliptic curves can leverage the Diffie-Hellman key exchange to a quantum-safe key exchange. Further we will take a look at ongoing work in IKEv2 to establish a quantum-safe key exchange.

2.2.1 Supersingular Isogeny Diffie-Hellman

In 2011 Jao et al. proposed a Diffie-Hellman key exchange based on the problem of finding isogenies between supersingular elliptic curves often referred to as Supersingular Isogeny Diffie-Hellman (SIDH)[JDF11]. Figure 2.14 shows the necessary steps for the key exchange with SIDH.

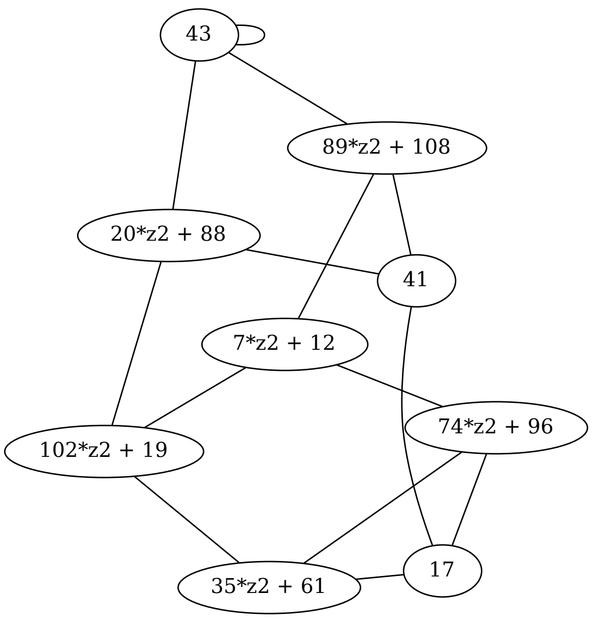

Figure 2.13: Nine j-invariants of all supersingular elliptic curves defined over finite field $\mathbb{F}_{109^2}$ and their 2-isogenies

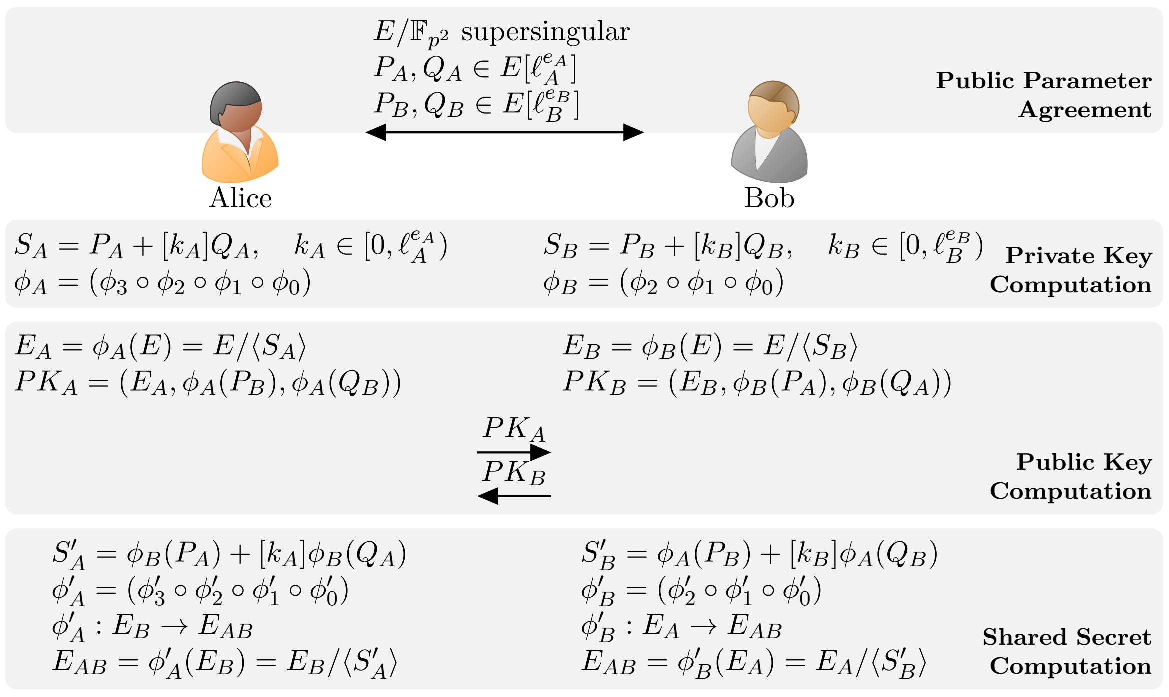
$E/\mathbb{F}_{p^2}$ supersingular
$P_A, Q_A \in E[\ell_A^{e_A}]$
$P_B, Q_B \in E[\ell_B^{e_B}]$
Public Parameter Agreement
Alice
Bob
$S_A = P_A + [k_A]Q_A, \quad k_A \in [0, \ell_A^{e_A})$
$\phi_A = (\phi_3 \circ \phi_2 \circ \phi_1 \circ \phi_0)$
$S_B = P_B + [k_B]Q_B, \quad k_B \in [0, \ell_B^{e_B})$
$\phi_B = (\phi_2 \circ \phi_1 \circ \phi_0)$
Private Key Computation
$E_A = \phi_A(E) = E/\langle S_A \rangle$
$PK_A = (E_A, \phi_A(P_B), \phi_A(Q_B))$
$E_B = \phi_B(E) = E/\langle S_B \rangle$
$PK_B = (E_B, \phi_B(P_A), \phi_B(Q_A))$
PK_A
PK_B
Public Key Computation
$S'_A = \phi_B(P_A) + [k_A]\phi_B(Q_A)$
$\phi'_A = (\phi'_3 \circ \phi'_2 \circ \phi'_1 \circ \phi'_0)$
$\phi'_A : E_B \to E_{AB}$
$E_{AB} = \phi'_A(E_B) = E_B/\langle S'_A \rangle$
$S'_B = \phi_A(P_B) + [k_B]\phi_A(Q_B)$
$\phi'_B = (\phi'_2 \circ \phi'_1 \circ \phi'_0)$
$\phi'_B : E_A \to E_{AB}$
$E_{AB} = \phi'_B(E_A) = E_A/\langle S'_B \rangle$
Shared Secret Computation

Figure 2.14: Supersingular Isogeny key exchange

Public Parameter Agreement First Alice and Bob agree on an elliptic curve defined over $\mathbb{F}_{p^2}$. p needs to be a prime and generated as follows:

$$p = \ell_A^{e_A} \ell_B^{e_B} \cdot f \pm 1$$

ℓ needs to be a small prime, usually $\ell_A = 2$ and $\ell_B = 3$ as these are the two smallest primes. f is a factor which was intended for more flexibility, however, in current implementations of SIDH this factor is 1. The added or subtracted one ensures, p is prime.

Further Alice and Bob agree on two points P_A and P_B that are generators of the $\ell_A^{e_A}$-torsion subgroup $E[\ell_A^{e_A}]$ and on two points P_B and P_Q that are generators of the $\ell_B^{e_B}$-torsion subgroup $E[\ell_B^{e_B}]$.

Secret Key Computation In the next step Alice and Bob compute their secret generator points. Alice does this by choosing a random value k_A in the range from 0 to $\ell_A^{e_A}$. Then she calculates $S_A = P_A + [k_A]Q_A$, which represents Alice's secret generator point. Alice generates her secret key by computing a composition of e_A degree-ℓ_A isogenies $\phi_A = (\phi_{e_A-1} \circ \phi_{e_A-2} \circ \ldots \circ \phi_0)$ with kernel S_A. This calculation can be thought of taking e_A random steps in the isogeny graph. The secret key is an isogeny $\phi_A : E \to E_A$, which can map points on the initial curve E to points on the curve $E_A = \phi_A(E) = E/\langle S_A \rangle$.

Bob performs the same process of choosing a random value k_B in the range from 0 to $\ell_B^{e_B}$ and then calculating $S_B = P_B + [k_B]Q_B$, representing Bob's secret generator point. With S_B, Bob can compute e_B degree-ℓ_B isogenies $\phi_B = (\phi_{e_B-1} \circ \phi_{e_B-2} \circ \ldots \circ \phi_0)$ resulting in Bob's private key $\phi_B : E \to E_B$.

Public Key Computation Then Alice and Bob continue computing their public keys. Alice's public key PK_A is the concatenation of the curve $E_A = \phi_A(E)$ and the points $\phi_A(P_B)$ and $\phi_A(Q_B)$ which are the two points from Bobs $\ell_B^{e_B}$-torsion group after applying the isogeny ϕ_A to them. Bob concatenates $E_B = \phi_B(E)$ and the two points $\phi_B(P_A)$ and $\phi_B(Q_A)$ to build his public key PK_B. Alice and Bob exchange their public keys PK_A and PK_B.

Shared Secret Computation Now Alice uses the points $\phi_B(P_A)$ and $\phi_B(P_A)$ she received from Bob and generates a second secret generator point S'_A. Using that point S'_A, Alice computes another composition of e_A degree-ℓ_A isogenies resulting in the Isogeny ϕ'_A, which she applies to the curve E_B, she also received from Bob. This leads Alice to the curve $E_{AB} = \phi'_A(E_B) = E_B/\langle S'_A \rangle$. Bob does the same, composing e_B degree-ℓ_B isogenies resulting in an isogeny ϕ'_B which Bob then applies to E_A leading him to E_{AB}. At that point Alice and Bob have arrived at the same elliptic curve E_{AB} from which they can derive their shared secret.

2.2.2 Towards a quantum-safe key exchange in IKEv2

There is ongoing work on IKEv2 with regards to a quantum-safe key exchange. Taking this work into account helps with understanding problems introduced by quantum-safe algorithms. Also found solutions of this work might assist in solving challenges of quantum-resistant authentication.

Heider elaborated a quantum-safe key exchange protocol in IKEv2 in his masters thesis[Hei19]. In his work, he enabled IKEv2 to do a hybrid key exchange meaning after a (not neccessarily) classic key exchange up to 7 other potentially quantum-safe Key Encapsulation Mechanisms (KEMs) can be incorporated. This does not only ensure to achieve at least the same security as the traditional IKEv2 protocol but enables the operator to use more than one quantum-safe KEM. By this, quantum-security is still ensured even in the case where one KEM gets broken.

IKE_INTERMEDIATE

The hybrid key exchange needs to incorporate public keys of sizes that are too big to operate without IKEv2 fragmentation. Usually the key exchange algorithms are negotitated in the IKE_SA_INIT exchange. The IKE_SA_INIT exchange is not subject to fragmentation, as both peers must have agreed on the usage of fragmentation which is only possible after the first exchange. This means the IKE_AUTH request is the first message that can be fragmented by IKEv2. That is why the negotiation of the additional key exchanges can not take place in the IKE_SA_INIT exchange. Heider discussed possible options, how else to realize the negotiation of multiple key exchanges with the result of utilizing a new type of IKEv2 exchange, namely the IKE_INTERMEDIATE which is currently subject to standardization in the Internet-Draft "draft-ietf-ipsecme-ikev2-intermediate-05" [Smy20]. The initial key exchange gets negotiated as usual in the IKE_SA_INIT exchange, whereas every additional key exchange get negotiated in a separate IKE_INTERMEDIATE exchange. Note, that every IKE_INTERMEDIATE exchange uses keying material from its preceding exchange. That way it is ensured, that every negotiated key exchange affects the keying material used for the establishment of the SA.

3 Towards a Signature Scheme for IKEv2

Integrating a digital signature scheme in IKEv2 inherits different constraints and limitations. The goal is to operate IKEv2 with a quantum-safe signature scheme without noteworthy changes for the user in setting up a SA. This chapter discusses requirements for a digital signature scheme to achieve this objective. Further we present a signature scheme, fulfilling the stated requirements and resulting in a candidate for integration in IKEv2.

3.1 Requirements analysis

Before establishing requirements for the integration of a signature scheme into IKEv2, we need to analyse the current state of IKEv2 as well as go into the technical development of IKEv2.

3.1.1 Classic IKEv2 Authentication

The authentication mechanism, IKEv2 provides was designed with very few possible signature algorithms in mind. The first specification of IKEv2 (RFC 4306[Kau05]) from 2005 only mentions RSA and DSS as signature algorithms for authentication.

The DSS method is bound to SHA-1 as hash algorithm (see RFC 7296 [KHN$^+$14, §3.8]), which is not considered a secure hash function nowadays. SHA-1 has been object to several theoretical attacks and in 2015 Stevens et al. presented the first practical attack which carries out a collision attack from a freestart[SKP15]. In their work they argue that it is within the scope of criminal organizations to practically break SHA-1 by estimating the cost of of the computation on Amazon EC2 Instances.

Although RSA was meant to only use SHA-1 and therefore suffering from the same drawbacks as DSS, newer IKEv2 specifications (RFC 5996[KHNE10] and RFC 7296[KHN$^+$14]) state that IKEv2 implementations can use certificates to negotiate other hash functions. However, this is just a hack to circumvent the lack of flexibility of the RSA authentication method.

RFC 4754[FS07] adds another authentication method to IKEv2, namely ECDSA. This standard extends the IKEv2 protocol by three authentication methods, which differ only by the elliptic curve they operate on and by the used hash function. Note that either elliptic curve parameter is bound to one specific hash algorithm.

3.1.2 RFC 7427 - Digital Signature

Due to the inflexible integration of further signature algorithms, RFC 7427[KS15] introduces a more generic “Digital Signature” authentication method, which tries to mitigate the issues mentioned above. It uses one authentication method which has been defined by the IANA with a value of 14. If the peers use this authentication method, the `Authentication Data` payload contains more than just the raw signature value, most notably an `AlgorithmIdentifier ASN.1 Object`, specifying the used algorithm, and the `Signature Value` itself. The hash algorithm is negotiated within the `IKE_SA_INIT` exchange in the form of a notification payload. It contains on the one hand confirmation that the peer supports the `Digital Signature` authentication method and on the other hand the supported hash algorithms in the `Notification Data`. The negotiation of the signature algorithm itself remains unspecified in RFC 7427, however, three suggestions how to find an agreement are made within the document. The first suggestion utilizes the `IDr` payload of the `IKe_AUTH` request, the second suggestion uses the `CERTREQ` payloads and the third proposed option is for the responder to check the key-type which the initiator has used and simply use the same algorithm. For more details on algorithm negotiation see RFC 7427[KS15].

3.1.3 IKEv2 Size Constraints

The size of public keys and the signature data of quantum-safe signature algorithms exceeds their counterparts of classic signature schemes like RSA and (EC)DSA by far. We need to consider two size limitations, the IKEv2 protocol implicates.

Maximum Transmission Unit Size

Since IKEv2 is based on UDP, packets with a size bigger then the Maximum Transmission Unit (MTU) get fragmented on the IP level. Thereby issues with networking middleware as well as firewalls arise, resulting in IP packets being potentially dropped. For IPv4 this MTU can be as low as 68 bytes (see RFC 791[Pos81, §3.2 p25]) whereas IPv6 has to support 1280 bytes (see RFC 8200[DH17, §5 p25]). In practice these values are usually higher, in Ethernet networks the MTU often has a value of 1500 bytes. The IKEv2 standard defined in RFC 7296 requires an implementation to support at least 1280 bytes (see RFC 7296[KHN$^+$14, §2 p24]).

IKE Fragmentation

The problem of IP-level fragmentation of packages bigger than the MTU is addressed by implementing fragmentation of IKEv2 packets as part of the IKEv2 protocol itself (see RFC 7383[Smy14]), so that on IP-level the packages are small enough not to be fragmented. This approach can only be applied after both communication partners have agreed on fragmenting within IKEv2. As a result IKEv2 fragmentation can not be used at the `IKE_SA_INIT` exchange.

64kiB Limit

The second limit is the maximum payload size of an IKEv2 structure. The payload size is stored in a 16 bit field allowing a maximum value of 2^{16} Byte = 65536 Byte = 64 kiByte. IKEv2 fragmentation is not a solution to this limitation because the message structure does not change and the fragmentation takes place on binary data just before sending the message. To be able for a peer to reconstruct the data, IKEv2 needs to put together all the received fragments and parse the headers to reconstruct the payload data. Therefore it is necessary for the payload headers to reflect the actual size of the payload and we need to adhere to a maximum length of 64kiB for each payload.

Support for Payload Sizes bigger than 64kiB

To deal with the 64kiB limit another Internet-Draft "draft-tjhai-ikev2-beyond-64k-limit-00"[THS20] is currently discussed and on its way to get standardized. In this draft two approaches to bypass the described limit are proposed. The first approach is to incorporate the "Hash and URL" method which hashes the big data payload and serves the actual data via HTTP from a web server. Only the hash and the URL where the data can be downloaded from are sent within the IKEv2 protocol. The other peer can then download payload sizes bigger than 64kiB from the first peer's web server.

The second approach is "Payload Fragmentation" which uses a similar approach as the IKEv2 fragmentation. The Internet-Draft presents two possible Payload Fragmentation variants each having its advantages and disadvantages. The first approach basically splits a big payload up into smaller payloads with a size smaller than 64kiB but leaves the smaller payloads part of one IKEv2 message. The IKEv2 message fragmentation mechanism handles the fragmentation of the message before sending it on the wire.

The second approach also splits the big payload up into smaller payloads but in contrast to the first variant, these payload fragments are sent as separate IKEv2 messages. After the peer received an IKEv2 message, it acknowledges it. Only after having received the acknowledgement of the previously sent IKEv2 message does the initiator continue to send the next IKEv2 message containing the next part of the payload.

3.1.4 Hybrid Authentication

Research is ongoing to integrate a quantum-safe key exchange in IKEv2 ([Hei19], [Smy20], [TTB$^+$20]) which adds a hybrid key exchange to the key exchange mechanism of IKEv2. The target of this work is to not depend on the security of one single KEM. Most of the quantum-safe cryptographic suites are quite young compared to classic cryptographic systems. As the security of such a system can ultimately only be proven by not being broken over a long time, the confidence in new quantum-safe algorithms is not as high as the confidence in cryptographic systems that have existed for a long time and withstood cryptographic analysis. This is why previously mentioned work incorporates first a classic key exchange which is considered highly secure against an attacker in possession of just a classic computer. The following one or more key exchanges are usually quantum-safe key exchanges adding security against an attacker in possession of a quantum computer. This procedure ensures that, if one of the new quantum-safe algorithms is rendered unsecure, there is still the security of at least the classic algorithm.

This approach is incorporated at the key exchange and can also be transferred to the authentication in IKEv2. The idea is to use a classic authentication mechanism like RSA in a first round of authentication. Then, in one or more following rounds of authentication, newer quantum-safe authentication mechanisms with a lower degree of confidence can be incorporated. Again, in a worst case scenario, where a new algorithm is broken, we can still rely on the classic security of the first authentication round. This ensures that there is no reduction of security by introducing new quantum-safe signature mechanisms, even in case of the new mechanism being broken by a newly found attack.

3.2 A Digital Signature Scheme based on supersingular Isogenies

In [DFJP14] De Feo et al. published an updated version of [JDF11], where they extended their cryptosystems based on supersingular elliptic curve isogenies by a zero-knowledge proof of identity. This will be the starting point towards a signature scheme that can finally get integrated into the IKEv2 protocol.

This work will explain zero-knowledge proofs in general and the zero-knowledge proof of identity proposed by De Feo et al. in greater detail. We will also show that their proof meets the structure of sigma protocols with special properties, namely completeness, special soundness and honest verifier zero-knowledge. Further we will discuss Non-interactive Zero-knowledge Proofs (NIZKs) and Unruhs transformation of the proof of identity proposed by De Feo et al. to a NIZK [Unr15]. One last modification will be applied to transform the NIZK to a signature scheme, not only proving an identity but proving authenticity of a message.

3.2.1 Zero-knowledge proof of identity

A zero-knowledge proof of identity is an interactive proof of a secret where the prover does not reveal any information about secret.

Figure 3.1 shows a zero-knowledge proof which makes it easy to understand those proofs. In order to prove to Vic that she is in possession of a secret to open the door, Peggy enters the cave randomly via side A or B. Vic does not see which side she chooses. Vic then tells Peggy on which side she should appear and observes, if she actually appears where Vic told her to. If Peggy can appear at the correct side for a defined amount of rounds, Vic can assume that she indeed knows the secret to open the door in the cave.

Note that Peggy could also enter the cave together with Vic, leave him at the branching point and enter one branch and come out at the other branch. This would prove knowledge of the secret to open the door without revealing the secret to Vic or anyone else, but Peggy could not deny knowing the secret to a third party secretly watching the scene.

Also important to note is that this proof only works for Vic because third parties have no way of knowing whether Peggy and Vic have made an agreement in advance on where Peggy should appear, or not.

Zero-knowledge Proof by De Feo et al.

The zero-knowledge proof of identity proposed by De Feo et al. ([DFJP14]) composes private and public parameters as follows: The prover Peggy chooses a secret point S that generates the kernel of the isogeny $\phi : E \to E/\langle S \rangle$. The public parameters are:

- $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$

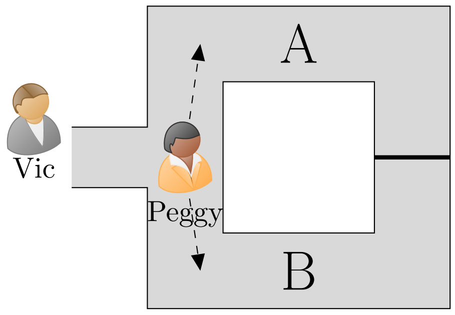

Figure 3.1: Zero-knowledge Proof

- supersingular elliptic curves $E(\mathbb{F}_{p^2})$ and $E/\langle S\rangle$
- generator points P_B, Q_B of the $\ell_B^{e_B}$-torsion subgroup $E[\ell_B^{e_B}]$
- image points $\phi(P_B)$ and $\phi(Q_B)$ of the generator points P_B, Q_B

The secret parameter is the isogeny $\phi : E \to E/\langle S\rangle$.

The protocol looks as follows [YAJ+17, §2.1]:

1. Peggy picks random point R of order $\ell_B^{e_B}$
2. Peggy computes the isogeny $\psi : E \to E/\langle R\rangle$
3. Peggy computes the isogeny $\psi' : E/\langle S\rangle \to E/\langle S, R\rangle$ or the isogeny $\phi' : E/\langle R\rangle \to E/\langle S, R\rangle$
4. **Commitment:** Peggy sends com $= (E_1, E_2)$, where $E_1 = E/\langle R\rangle$ and $E_2 = E/\langle S, R\rangle$ to Vic (verifier)
5. **Challenge:** Vic sends randomly chosen bit ch $\in 0, 1$ to Peggy
6. **Response:** If ch $= 0$, Peggy sends resp $= (R, \phi(R))$, else Peggy sends resp $= \psi(S)$
7. If ch $= 0$, Vic verifies that R and $\phi(R)$ have order $\ell_B^{e_B}$ and generate the kernels of the isogenies $E \to E_1$ and $E \to E_2$ respectively
   If ch $= 1$, Vic verifies that $\psi(S)$ has order $\ell_A^{e_A}$ and generates the kernel for the isogeny $E_1 \to E_2$

Figure 3.2 shows Peggy's computations indicating that the isogenies she sends in her response to Vic depend on the challenge bit. On the left side in the figure, Peggy reveals the randomly chosen point R and the point $\phi(R)$ after applying the secret isogeny ϕ to that point. This enables Vic to compute the isogeny ψ by knowing R and to compute the isogeny ψ' by knowing $\phi(R)$. On the right side Peggy reveals $\psi(S)$, enabling Vic to compute the isogeny ϕ'.

In both cases, Vic is able to compute $E/\langle R, S\rangle$ with the help of the response message from Peggy and compare the resulting curve with the one of the commitment message from Peggy. If both curves match, Vic can be sure that Peggy is in possession of the Secret Point S. This protocol is repeated several times and only if all rounds result in the verification of Vic, we can assume Peggy's identity to be proven.

The downside of zero-knowledge proofs is the need for multiple interactions. To integrate a signature scheme in IKEv2 the interactions need to be reduced to the absolute minimum. This

$$\begin{tikzcd}
E \arrow[r, dashed, "\phi"] \arrow[d, "\psi"'] & E/\langle S\rangle \arrow[d, "\psi'"] \\
E/\langle R\rangle \arrow[r, dashed, "\phi'"] & E/\langle S, R\rangle
\end{tikzcd}$$

ch=0

$$\begin{tikzcd}
E \arrow[r, dashed, "\phi"] \arrow[d, dashed, "\psi"'] & E/\langle S\rangle \arrow[d, dashed, "\psi'"] \\
E/\langle R\rangle \arrow[r, "\phi'"] & E/\langle S, R\rangle
\end{tikzcd}$$

ch=1

Figure 3.2: Zero-knowledge proof and published isogenies [YAJ+17]

zero-knowledge proof would require one message exchange per round it executes, rendering the proof useless for protocols that run on distributed peers with traffic and bandwidth limitations, like IKEv2. To reduce the amount of interactions, there is Unruh's transformation, which transforms special zero-knowledge proofs to NIZKs. These special zero-knowledge proofs need to be sigma protocols complying with the properties *completeness*, *special soundness* and *honest verifier zero-knowledge.* The next subsection discusses these protocols and properties.

3.2.2 Sigma protocol

A sigma protocol is an interactive zero-knowledge proof generally engaging three messages. The first is the commitment $com \leftarrow P_1(x, w)$, where the prover sends a claim x and a witness of that claim w to the verifier. The second message sent from the verifier to the prover is the challenge $ch \overset{\$}{\leftarrow} N_{ch}$ where $N_{ch} = \{0, 1\}$ usually. The dollar-sign indicates that the challenge is a randomly chosen value from the set N_{ch}. The prover sends the third message, namely the response message $resp \leftarrow P_2(ch)$, revealing information which depends on the challenge bit. The verifier then accepts or rejects the claim of the prover. Having a look at the zero-knowledge proof from the previous Section 3.2.1, we see that the proof engages those three messages making it a sigma protocol.

Sigma protocols may have the three following properties [YAJ+17, §3.1]:

Completeness: If the prover knows a witness to his claim, the verifier accepts the claim.

Special soundness: There exists a polynomial time extractor E_Σ such that, given any pair of valid interactions (com,ch,resp) and (com,ch',resp') with ch≠ch' that the verifier accepts, E_Σ can compute a witness to the claim

Honest-verifier zero-knowledge: There is a polynomial time simulator S_Σ with outputs of the form (com,ch,resp) that are indistinguishable from valid interactions between a prover and an honest verifier by any quantum polynomial time algorithm.

De Feo et al. prove in their work [DFJP14, §6.2] that their zero-knowledge proof meets *completeness*, *soundness* and *honest-verifier zero-knowledge.* Further Yoo et al prove that the proof also has *special soundness.*

3.2.3 Unruhs Transformation towards a non-interactive Proof System

A non-interactive proof of identity aims to achieve the same functionality as the zero-knowledge proof of identity, but without the need for messages containing a challenge and a response to the challenge.

Formally we can describe a proof system by taking a $\mathcal{R}$ that is a fixed decidable relation on bitstrings[Unr15, §2.1]. The prover can claim x if there is a witness w so that $(x, w) \in R$. In the non-interactive case, we have two algorithms, $\mathcal{P}(x, w)$ and $\mathcal{V}(x, \pi)$. $\mathcal{P}(x, w)$ takes the claim x and the witness w and outputs a proof π. $\mathcal{V}(x, \pi)$ takes the claim x and the proof π and accepts or rejects the claim x.

Unruh's states following in [Unr15, Corollary 14]:

Corollary. If there is a sigma-protocol Σ that is complete and HVZK and has special soundness, then there exists a non-interactive zero-knowledge proof system with simulation-sound online extractability in the random oracle model.

Online extractability is the property that guarantees, that the extractor does not have to rewind the prover to extract the witness from the proof. Simulation-soundness ensures that one can not prove a statement if one is only in possession of a proof of a related statement. That property is essential to signature schemes from non-interactive proof systems, as we do not want a third party to be able to capture Peggy's signed data, amend it and then forward it to Vic with a valid signature belonging to Peggy.

Due to the fact that the zero-knowledge proof of identity proposed by De Feo et al. fulfills all requirements to be able to apply Unruh's construction, we can obtain a non-interactive proof system from it.

Unruhs Transformation

The transformation proposed by Unruh [Unr15] deals with the problem of eliminating the need for the challenge and the response messages, that sigma protocols engage. Other transformations exists, most notably by Fiat-Shamir [FS86] and by Fishlin [Fis05], which are both not considered safe in a quantum adversary setting, at least not without having strict constraints affecting particularly efficiency.

Unruh introduces a transformation that is based on the Fiat-Shamir transformation but modified to hold up against quantum attackers. Generally, Unruh's construct gets around the interactions of zero-knowledge proofs by calculating every response, that a challenge might request. The challenge and respective response which are finally used are determined by a challenge hash that is generated from some parameters, including a hash of all possible responses of all rounds. This is the main difference to the Fiat-Shamir transformation which generates the challenge hash only from the commitments of each round.

Algorithm 3.1 and Algorithm 3.2 show the proving algorithm $\mathcal{P}(x, w)$ and the verifying algorithm $\mathcal{V}(x, \pi)$ of the NIZK resulting from a sigma protocol in full detail.

Algorithm 3.1: Prover $\mathcal{P}(x, w)$ [Unr15, Fig. 1]

```
Input: $(x, w)$ with $(x, w) \in R$
// Create $t \cdot m$ proofs $(com_i, ch_{i,j}, resp_{i,j})$
for $i = 1$ to $t$ do
    $com_i \leftarrow P^1_\Sigma(x, w)$
    for $j = 1$ to $m$ do
        $ch_{i,j} \xleftarrow{\$} N_{ch} \setminus \{ch_{i,1}, \dots ch_{i,j-1}\}$
        $resp_{i,j} \leftarrow P^2_\Sigma(ch_{i,j})$
// Commit to responses
for $i = 1$ to $t$ do
    for $j = 1$ to $m$ do
        $h_{i,j} := G(resp_{i,j})$
// Get challenge by hashing
$J_1 \| \dots \| J_t := H(x, (com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j})$
// Return Proof (only some responses)
return $\pi := ((com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j}, (resp_{i,J_i})_i)$
```

Algorithm 3.2: Verifier $\mathcal{V}(x, \pi)$ [Unr15, Fig. 1]

```
Input: $(x, \pi)$ with $\pi = ((com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j}, (resp_i)_i)$
$J_1 \| \dots \| J_t := H(x, (com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j})$
for $i = 1$ to $t$ do
    check $ch_{i,1}, \dots, ch_{i,m}$ pairwise distinct
for $i = 1$ to $t$ do
    check $V_\Sigma(x, com_i, ch_{i,J_i}, resp_i) = 1$
for $i = 1$ to $t$ do
    check $h_{i,J_i} = G(resp_i)$
if all checks succeed then
    return 1
```

3.2.4 Signature Scheme

As discussed in Section 2.1.2 a signature scheme usually consists of three algorithms.

1. Keygen(λ): generates a key pair (sk, pk) based on a security parameter λ.
2. Sign(sk, m): generates a signature σ using a message m and the signing key sk.
3. Verify(pk, m, σ): verifies the signature σ using the prover's public key, the message m and the signature σ itself.

To transform a non-interactive proof of identity to a signature scheme, the claim x has to consist of the public key pk and the message m. The witness w of the claim x is the signing key sk. R needs to ignore the message m, so that $((pk, m), w) \in R$ if and only if (pk, w) is a valid key pair generated by the Keygen algorithm.

In summary, we get a signature scheme with the three algorithms Keygen(λ), Sign$(sk, m) = P((pk, m), sk)$ and Verify$(pk, m, \sigma) = V((pk, m), \sigma)$.

Signature Scheme based on Isogenies [YAJ$^+$17]

Applying Unruh's construction to the zero-knowledge proof Σ shown in Section 3.2.1 by De Feo et al. we obtain a non-interactive proof of identity, which we again can transfer into a signature scheme.

The **public parameters** of the signature scheme are the same as in the zero-knowledge proof discussed in Section 3.2.1:

- $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$
- supersingular elliptic curves $E(\mathbb{F}_{p^2})$ and $E/\langle S \rangle$
- generator points P_B, Q_B of the $\ell_B^{e_B}$-torsion subgroup $E[\ell_B^{e_B}]$
- image points $\phi(P_B)$ and $\phi(Q_B)$ of the generator points P_B, Q_B

The **key generation** makes use of a random point S of order ℓ^{e_A}, which allows the computation of the isogeny $\phi : E \to E/\langle S \rangle$. The public key composes to $pk = (E/\langle S \rangle, \phi(P_B), \phi(Q_B))$ whereas the signing key $sk = S$. The **signing** follows the equation

$$\text{Sign}(sk, m) = P((pk, m), sk)$$

and the **verification** follows

$$\text{Verify}(pk, m, \sigma) = V((pk, m)\sigma)$$

The key generation, signing and verifying algorithms are shown in full detail in Algorithm 3.3, Algorithm 3.4 and Algorithm 3.5.

3.3 Requirements

From the IKEv2 status quo and current developments mainly in the field of quantum-safe key exchanges in IKEv2 discussed in Section 3.1 and after understanding the internal functionality

Algorithm 3.3: Key generation algorithm [YAJ$^+$17, §4.1]

```
Input: $\lambda$
Pick a random point $S$ of order $\ell_A^{e_A}$
Compute the isogeny $\phi : E \to E/\langle S \rangle$
$pk \leftarrow (E/\langle S \rangle, \phi(P_B), \phi(P_Q))$
$sk \leftarrow S$
return $(pk, sk)$
```

Algorithm 3.4: Signing algorithm [YAJ$^+$17, §4.1]

```
Input: sk, m
for $i = 1$ to $2\lambda$ do
    Pick a random point $R$ of order $\ell_B^{e_B}$
    Compute the isogeny $\psi : E \to E/\langle R \rangle$
    Compute either $\phi' : E/\langle R \rangle \to E/\langle R, S \rangle$ or $\psi' : E/\langle S \rangle \to E/\langle R, S \rangle$
    $(E_1, E_2) \leftarrow (E/\langle R \rangle, E/\langle R, S \rangle)$
    $com_i \leftarrow (E_1, E_2)$
    $ch_{i,0} \overset{\$}{\leftarrow} 0, 1$
    $(resp_{i,0}, resp_{i,1}) \leftarrow ((R, \phi(R)), \psi(S))$
    if $ch_{i,0} = 1$ then
        $swap(resp_{i,0}, resp_{i,1})$
    $h_{i,j} \leftarrow G(resp_{i,j})$
$J_1 \| \dots \| J_{2\lambda} := H(pk, m, (com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j})$
return $\sigma \leftarrow ((com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j}, (resp_{i,J_i})_i)$
```

Algorithm 3.5: Verifying algorithm [YAJ$^+$17, §4.1]

```
Input: pk, m, $\sigma$
$J_1 \| \dots \| J_{2\lambda} := H(pk, m, (com_i)_i, (ch_{i,j})_{i,j}, (h_{i,j})_{i,j})$
for $i = 1$to$2\lambda$ do
    check $h_{i,j} = G(resp_{i,J_i})$
    if $ch_{i,J_i} = 0$ then
        Parse $(R, \phi(R)) \leftarrow resp_{i,J_i}$
        check $R, \phi(R)$ have order $\ell_B^{e_B}$
        check $R$ generates the kernel of the isogeny $E \to E_1$
        check $\phi(R)$ generates the kernel of the isogeny $E/\langle S \rangle \to E_2$
    else
        Parse $\psi(S) \leftarrow resp_{i,J_i}$
        check $\psi(S)$ has order $\ell_A^{e_A}$
        check $\psi(S)$ generates the kernel of the isogeny $E_1 \to E_2$
if all checks succeed then
    return 1
```

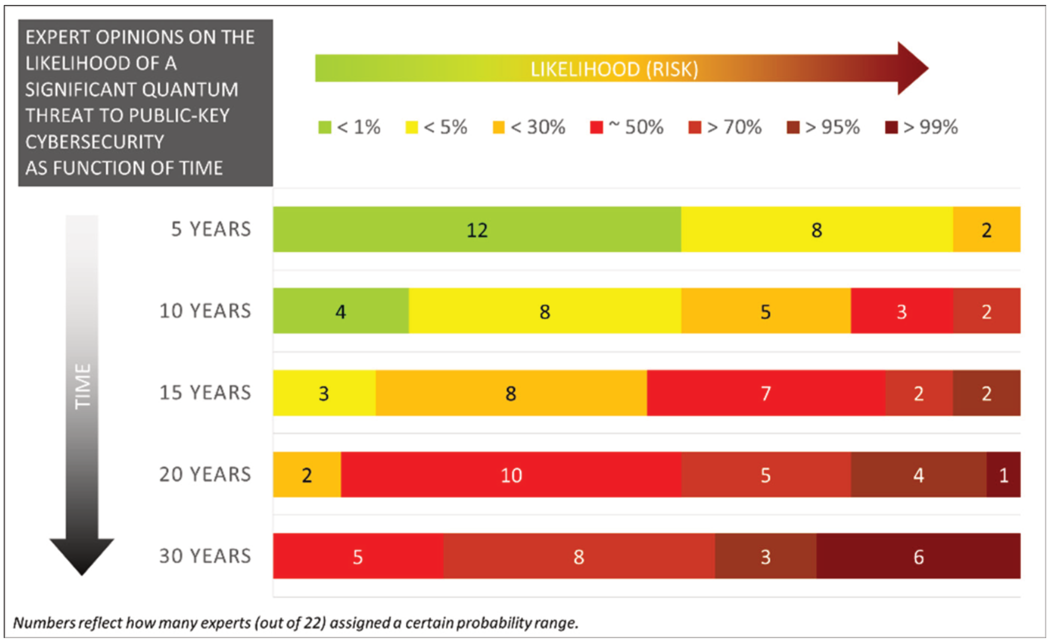

Figure 3.3: Likelihood of Quantum Threat to Public-Key Cryptography [MP19]

of the isogeny-based signature scheme shown in Section 3.2 we can derive requirements for a protocol extension allowing quantum-safe authentication in IKEv2. The following paragraphs extract and elaborate on requirements resulting from the previous Sections.

Quantum Security: Achieving quantum-safe authentication in the `IKE_AUTH` exchange is the main motivation of this work. Section 1 already pointed out the threat a quantum computer poses to the classic public key cryptography. Although it is unclear when a sufficiently powerful quantum computer will be available, the time to prepare for this event is now. Mosca and Piani published a report [MP19] in 2019 where they performed a survey, asking various researchers about the "likelihood of significant quantum threat to public-key cybersecurity as function of time". Their result is shown in Figure 3.3, representing the opinions of 22 researchers. Whilst only two experts estimate the discussed likelihood to be higher than 30% in 2024 (5 years from 2019), in 2034 (15 years from 2019) 19 of the 22 experts chose this likelihood value or higher. All experts estimate the likelihood in 30 years (from 2019) to be at least 50% or higher, 6 of them even settling at a likelihood of >99%. The 15year mark is further interesting as exactly half of the researchers think the likelihood of a quantum threat is smaller than 50% and the other half thinks it is around 50% or higher. This survey represents the opinion of the experts which have taken part and does not guarantee for whatever outcome. It does, however, give a rough idea of the time remaining to get prepared for the quantum threat to public-key cryptography.

In the context of digital signatures, quantum security means an attacker utilizing a quantum computer is not able to successfully claim a third party identity to a communication partner. This requires the signature scheme to use algorithms which are assumed hard to break for quantum computers.

Compliance with protection goals Authenticity, Integrity and Non-Repudiation: In Section 2.1.2 we have shown five properties of digital signatures, i.e. the signature is authentic, unforgeable, not reusable, the signed document is unalterable and the signature cannot be repudiated. These five points have been transferred from handwritten signatures to digital signatures by Schneier [Sch07]. Further we extracted security goals from these properties, being authenticity, integrity and non-repudiation. Only if our signature scheme complies with those security goals, we can ensure compliance to the five stated properties. This would make our signature scheme a reliable way of proving authorship of a message.

Prevention of security reduction: The usage of a quantum-safe signature scheme must not weaken the IKEv2 authentication under any circumstances. Due to the fact that most of the post-quantum algorithms are relatively new compared to classic algorithms like RSA which exist for almost 50 years, the confidence in those newer algorithms is lower than the confidence in algorithms that have been subject to attacks for a long time. However unlikely, it might happen that a cryptographic suite which is assumed to be quantum resistant is broken by a newly invented attack which can be executed efficiently on a classic computer. Therefore we need to ensure that the whole authentication of IKEv2 is not broken by the usage of an unsecure post-quantum signature algorithm We still need to provide (potentially less strong) authenticity, integrity and non-repudiation. The ipsecme working group[1] of the Internet Engineering Task Force (IETF) was facing a similar problem at IKEv2 with the key exchange which led them to a hybrid key exchange [TTB$^+$20]. Also from the IETF comes a draft for standardization of a hybrid key exchange for TLS1.3 eliminating the need to trust a newer quantum-safe algorithm by relying on a classic key exchange previous to quantum safe exchanges [SFG20].

Crypto-agility: As mentioned in Section 3.1.1 the IKEv2 authentication mechanism was designed in an inflexible manner. RFC 7427[KS15] as described in Section 3.1.2 has been introduced to mitigate this inflexibility. While designing a quantum-safe protocol extension, we must not reintroduce this inflexibility. It is even more important to allow the drop-in replacement of other signature algorithms, as we might have to deal with broken signature algorithms due to the immaturity of the quantum-safe algorithms. The NIST has also taken this fact into account by explicitly stating, that the Post-Quantum Cryptography Standardization Process is not a competition but an attempt to standardize several cryptographic suites from which the user can chose the most appropriate for a specific use case [Che17][CCJ$^+$16].

Support for Payloads $>$ 64kiB: As crypto-agility needs to support various different signature algorithms we need to deal with different types of properties of those algorithms. One property, that all of the quantum-safe algorithms have in common is, that they either use much bigger public keys or much bigger signature data than the traditional signature algorithms like RSA or (EC)DSA do. Figure 3.4 shows the public key size and the signature size of the finalists and alternatives from the third round of the NIST post-quantum standardization program and of the isogeny-based signature scheme discussed in Section 3.2. The red points represent the public key size of the respective signature scheme and the blue points represent the signature size. The horizontal line indicates the 64KiB limit, dots over that line equal an exceedance of this limit. We can see

[1]IP Security Maintenance and Extensions (https://datatracker.ietf.org/wg/ipsecme/about/)

Table 3.1: Summary of Requirements

ID	Requirement
SigR1	Quantum-Security
SigR2	Compliance with protection goals Authenticity, Integrity and Non-Repudiation
SigR3	Prevention of security reduction
SigR4	Crypto-agility
SigR5	Support of Payloads > 64kiB
SigR6	Minimalism
SigR7	Practicability

several public keys exceeding the 64KiB limit, but also some of the signatures are bigger than the limit. To enable crypto-agility and to support a broad range of signature schemes it is vital for the IKEv2 extension to support payload sizes bigger than the 64kiB limit, described in Section 3.1.3.

Minimalism: Although the requirements "Prevention of security reduction", "Crypto-agility" and "Support for Payloads > 64kiB" all add complexity to the IKEv2 protocol, we need to keep minimalism in mind. This is critical to the protocol as adding too much complexity can result in design flaws and increases implementation bugs, both resulting in less security.

Practicability: The non-functional requirement "Practicability" needs to be taken into account, as there is a possibility that a protocol is developed, fulfilling all previous requirements but has for instance performance issues which render the protocol useless. Paquin, Stebila and Tamvada benchmarked post-quantum cryptography in TLS [PST20], pointing out that cryptographic suits which fragment across many packets suffer significantly from network package loss at rates of 3-5%. In their work they benchmarked key exchange as well as authentication mechanisms. TLS serves a different purpose in a different environment, however, the effects, Paquin et al. have discovered might also be applicable to the integration of quantum-safe cryptographic suites in IKEv2. Therefore we need to keep an eye on the practicability of the protocol extension. Only deploying and testing the extended protocol in a test environment can show, how it performs and in what scenarios potential drawbacks with regards to practicability are acceptable and what is not.

Table 3.1 provides an overview of the elaborated requirements for a digital signature scheme to be integrated in IKEv2. A unique requirement ID is assigned to each requirement which can be referred to more easily in the further work.

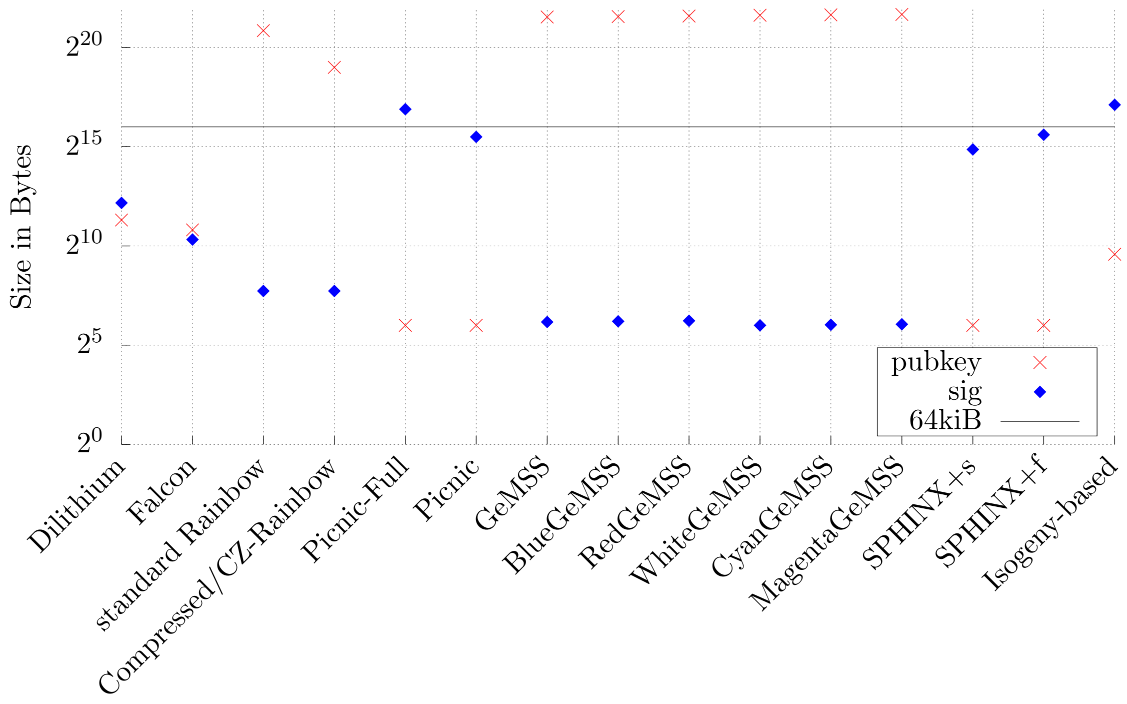

Figure 3.4: Comparison of Public key and Signature Sizes of NIST Round 3 Signature Finalists and Alternatives and Isogeny-based Signature Scheme

4 IKEv2 Protocol Design for quantum-resistant Authentication

This chapter presents a protocol extension of the IKEv2 protocol which provides quantum-resistant authentication. First we will discuss ideas to make the IKEv2 protocol comply with the requirements defined in Section 3.3. To get a comprehensive overview of possible approaches, also ideas that are not part of the final protocol are taken into account. Figure 4.1 gives an overview of the topics that have been considered to develop the protocol. The three top-level topics, *Signature Scheme Integration*, *Hybrid Authentication* and *Payloads >64kiB* are the high level problems that need to be addressed to obtain a protocol which complies with the previously defined requirements. The child topics on the same level represent alternative approaches for dealing with a problem and are directly related to their parent topic. To give an example, the approach *Bulk Transfer* and *Incremental Transfer* are exclusive alternatives to realize *Payload Fragmentation* which is one candidate to solve the *Payloads >64kiB* top-level topic.

Following the discussion of possible approaches to solve the top-level topics, Section 4.4 presents the final design of the elaborated protocol to incorporate quantum-safe authentication in IKEv2. We outline the approaches chosen to address the top-level problems set out, give a detailed explanation of the protocol procedure and evaluate the protocol with regards to the requirements defined in Section 3.3.

4.1 Integration of quantum-resistant Signature Schemes in IKEv2

As described in Section 3.1.1, IKEv2 was designed with only a few authentication mechanisms in mind. We do not look at the Extensible Authentication Protocol (EAP) method to authenticate, as it is irrelevant for this work. Instead we look for a replacement for the implemented authentication methods incorporating signing of data. Originally IKEv2 supported RSA and DSA and has later been extended by ECDSA in RFC 4754 [FS07]. All these mechanisms are tightly integrated into the IKEv2 protocol, making it non-trivial to replace the signature algorithms with newer, more secure ones. To make use of a signature algorithm other than RSA and (EC)DSA, one can choose between two options, either register a new authentication method or use the Digital Signature extension to IKEv2 as proposed in RFC 7424 [KS15]. The two approaches are discussed in the following Sections 4.1.1 and 4.1.2.

4.1.1 Registering a new Authentication Method

The first option, registering a new authentication method requires only small modifications to the protocol. Figure 2.6 shows the authentication payload which consists of the generic

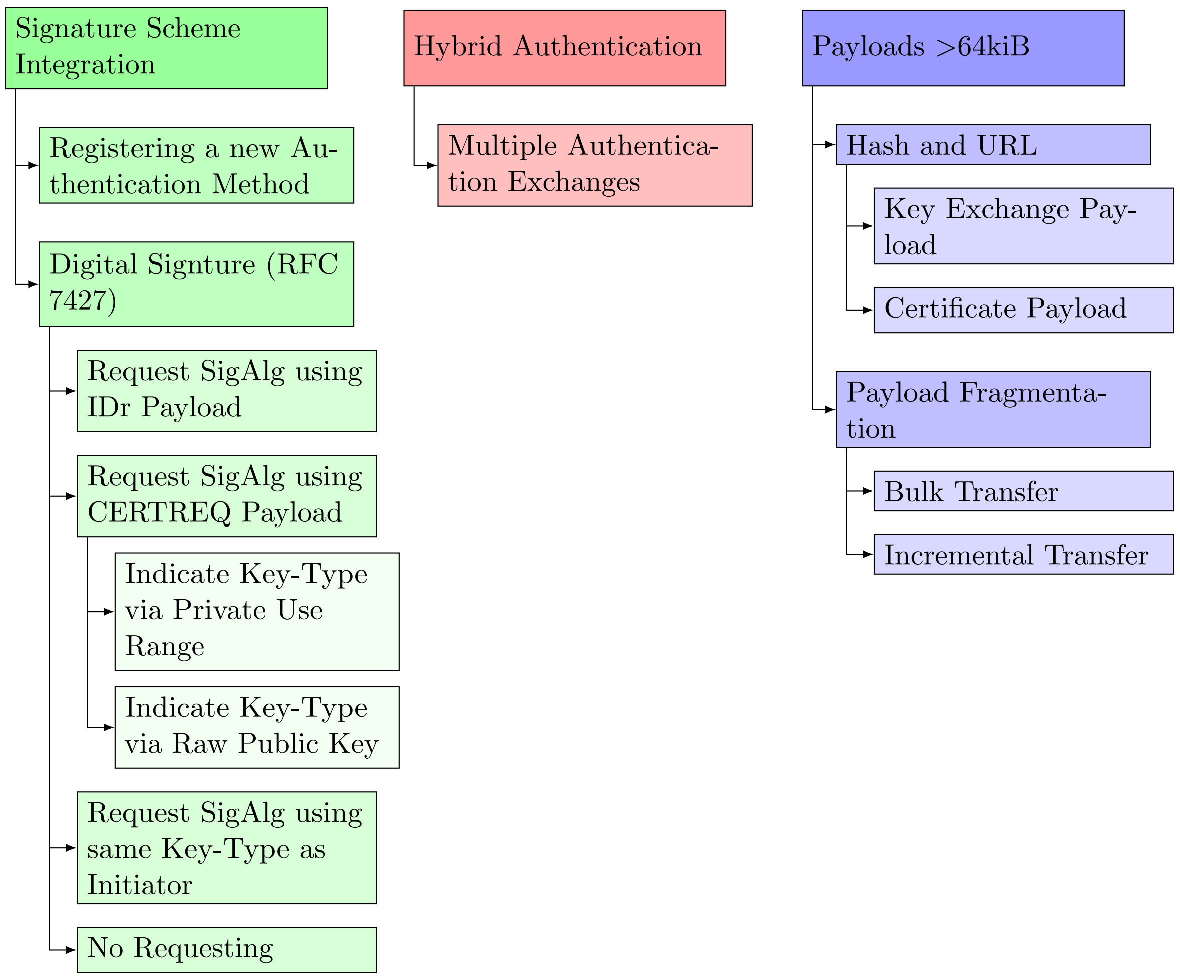
Signature Scheme Integration
Registering a new Authentication Method
Digital Signture (RFC 7427)
Request SigAlg using IDr Payload
Request SigAlg using CERTREQ Payload
Indicate Key-Type via Private Use Range
Indicate Key-Type via Raw Public Key
Request SigAlg using same Key-Type as Initiator
No Requesting
Hybrid Authentication
Multiple Authentication Exchanges
Payloads >64kiB
Hash and URL
Key Exchange Payload
Certificate Payload
Payload Fragmentation
Bulk Transfer
Incremental Transfer

Figure 4.1: Considered Approaches for the Protocol Design

payload header and the `Auth Method` and the `Authentication Data`. Integrating a new signature scheme by registering a new authentication method is as simple as setting the `Auth Method` field to the value of this new method and putting its signature data into the `Authentication Data` field. By looking at the `Auth Method` of the IKE_AUTH message, the responder can determine the new authentication mechanism and verify the signature data accordingly.

The downside of this approach is the inflexibility with regards to the combination of signature schemes and hash algorithms. When using a new authentication method, there is no negotiation of either the hash algorithm or the signature algorithm. This results in a fixed binding of the hash algorithm to the signature algorithm. In other words, every signature algorithm in cooperation with a specific hash algorithm needs its own assigned authentication method. This inflexibility means that the DSA signature algorithm can only be used in conjunction with SHA-1, which is flawed. Also the single ECDSA method is specified in three authentication methods combining different parameter sets with different hash algorithms. Only for the RSA signature algorithm the IKEv2 standard (RFC 7296 [KHN+14]) mentions a workaround to negotiate a more secure hash function via certificate payloads.

As far as new quantum-resistant algorithms are concerned, we should consider crypto-agility, which is the possibility to incorporate more than one algorithm and also to quickly exchange an algorithm which has proven unsecure. Furthermore we should still be able to use classic algorithms which have proven secure against classic attackers over the last decades. So ideally we do not just add one quantum-safe signature scheme to IKEv2 but implement the possibility to select from a pool of available signature schemes, classic ones and some of which are quantum-safe. Adding a pool of signature schemes, each combined with one or more hash algorithms, to IKEv2 via new authentication methods is cumbersome. Also the need for quick adaption conflicts with the static way of registering new authentication methods.

In summary, we can say that registering a new authentication method fulfills the requirements *SigR1 Quantum Security*, *SigR2 Compliance with protection goals Authenticity, Integrity and Non-Repudiation* and *SigR6 Minimalism* as it enables us to use a quantum safe signature scheme ensuring the required protection goals while only being minimal intrusive. However, this approach fails to fulfill *SigR4 Crypto-Agility* as well as *SigR7 Practicability* as the static character of registering new combinations of signature and hash algorithms is neither agile nor practical.

4.1.2 Digital Signature (RFC 7427)

The second option uses the authentication method “Digital Signature” which is registered at the IANA with a value of 14. This approach is described in RFC 7427 [KS15] and tries to mitigate the problems, the first approach has been criticized for, as shown above. This digital signature method has to be supported by both peers before they can use it for authentication purposes.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Protocol ID  |   SPI Size    |      Notify Message Type      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                Security Parameter Index (SPI)                 ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                       Notification Data                       ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.2: Notify Payload Format

Table 4.1: IKEv2 Hash Algorithms

Value	Hash Algorithm	Reference
0	Reserved	[RFC7427]
1	SHA1	[RFC7427]
2	SHA2-256	[RFC7427]
3	SHA2-384	[RFC7427]
4	SHA2-512	[RFC7427]
5	Identity	[RFC8420]
6	STRIBOG_256	[draft-smyslov-ike2-gost-02]
7	STRIBOG_512	[draft-smyslov-ike2-gost-02]
8-1023	Unassigned	
1024-65535	Reserved for Private Use	[RFC7427]

Indicating Support and Negotiating a Hash Algorithm

A peer indicates support for the authentication method by sending a notify payload of type SIGNATURE_HASH_ALGORITHM. The notify payload is described in RFC 7296 as shown in Figure 4.2. The `Protocol ID` field as well as the `SPI Size` field are both set to 0. The `Notify Message Type` is set to 16431, identifying the notify payload as SIGNATURE_HASH_ALGORITHM. The `Nofification Data` contains one or more hash algorithms supported by the peer. Table 4.1 shows the list of available hash algorithms. Note that this list does not show any members of the SHA-3-family as of now, but it can be easily adapted in the future to also support further hash algorithms.

Note that a peer may only use a specific hash algorithm, if the other peer indicated support for this hash algorithm in the SIGNATURE_HASH_ALGORITHM notify payload.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Auth Method   |                RESERVED                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ -+
| ASN.1 Length  | AlgorithmIdentifier ASN.1 object              |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
|                                                               |  |
~        AlgorithmIdentifier ASN.1 object continuing            ~  |
|                                                               |  |- Authenti-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |  cation
|                                                               |  |  Data
~                         Signature Value                       ~  |
|                                                               |  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ -+
```

Figure 4.3: Authentication Payload Format Digital Signature

Authentication Data Format

Using the digital signature authentication method comes with a change in the `Authentication Data` field in the authentication payload. For details on the authentication payload, see Figure 2.6. Authentication methods described in the IKEv2 specification in RFC 7296 [KHN+14] put the raw signature value into the `Authentication Data`. In contrast, by using the digital signature method we prepend two fields to the raw signature value, namely the `ASN.1 Length` and the `AlgorithmIdentifier ASN.1 object`. The format of the authentication payload is shown in Figure 4.3.

By looking at the `AlgorithmIdentifier ASN.1 object` of the authentication payload, the receiving peer knows what signature algorithm has been used for the signature generation and can choose the matching verifying algorithm accordingly. It is important to understand that specifying the `AlgorithmIdentifier ASN.1 object` is not a negotiation of signature schemes but identification of the algorithm that has been used for signature generation. The next paragraph describes methods to agree a signature algorithm.

Selecting the Signature Algorithm

RFC 7427 suggests three methods to agree on the signature algorithm. Neither of them describes an actual negotiation of signature algorithms but helps the initiator to request the use of a specific public/secret key pair from the responder. The initiator has to either know what signature schemes the responder supports by out-of-band communication or guess. If the responder is not able to verify the signature because he does not support the signature algorithm used by the initiator, he responds with a notify payload of error type `AUTHENTICATION_FAILED`. Below are the three proposals from RFC 7427.

Using IDr payload: This method uses the `IDr` payload of the IKE_AUTH request of the initiator. Hereby the initiator requests to talk to the peer with the `IDr` identity of the

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|  RESERVED   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cert Encoding |                                               |
+-+-+-+-+-+-+-+-+                                               |
~                    Certification Authority                    ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.4: Certificate Request Payload Format

other peer. This `IDr` identity must be bound to a key type using a specific signature scheme.

Using CERTREQ payload: The initiator can use the optional payload `CERTREQ` and indicate support of a signature algorithm to the responder by trusting a CA that uses this kind of signature algorithm.

Using same key type as initiator: The responder can use the same type of keys, the initiator used generating his signature. This way the responder ensures that initiator is capable to verify his signature. On the downside, this only works if both peers use public key authentication. If the initiator uses EAP or shared secret authentication, this option is rendered useless.

To circumvent the aforementioned fact, that with either of the shown methods the initiator has to know or guess the supported signature schemes of the responder, the responder could make use of the optional `CERTREQ` payload in the IKE_SA_INIT exchange. As soon as the responder receives the IKE_SA_INIT request including the notify payload SIGNATURE_HASH_SUPPORTED, he knows that the initiator is able to authenticate via the digital signature method. The responder can at that point already add a `CERTREQ` payload, hinting the initiator towards a signature algorithm he supports.

Neither the responder nor the initiator necessarily have to send a CA they trust in the `CERTREQ` payload, they can instead use a different `Cert Encoding`. Figure 4.4 shows the `CERTREQ` payload as defined by RFC 7296 [KHN$^+$14] and Table 4.2 shows the possible `Cert Encoding` values as defined by the IANA [IAN]. The peers can now use one of the cert encodings for private use with values starting at 201 or they can utilize the "Raw Public Key" (15) encoding that has been specified in RFC 7670 [KWT16].

In the first case, both peers have to interpret the cert encoding correctly. The `Certification Authority` field can be left blank, as the receiving peer only needs to know the type of the public key of the signature. This enables the receiving peer to use the secret key belonging to the public key to generate the signature data. The other peer can then verify the signature as he indicated his capability in the `CERTREQ` payload.

The second case using `Certificate Encoding` "Raw Public Key" (15) makes this approach even more generic. The type of the public key is not propagated via the `Certificate Encoding` field but is encoded using the SubjectPublicKeyInfo structure of the PKIX certificate inside the `Certificate Authority` field of the `CERTREQ` payload. The drawback of this approach

Table 4.2: IKEv2 Certificate Encodings

Value	**Certificate Encoding**	**Reference**
0	Reserved	RFC7296
1	PKCS #8 wrapped X.509 certificate	UNSPECIFIED
2	PGP Certificate	UNSPECIFIED
3	DNS Signed Key	UNSPECIFIED
4	X.509 Certificate - Signature	RFC7296
5	Reserved	RFC7296
6	Kerberos Token	UNSPECIFIED
7	Certificate Revocation List (CRL)	RFC7296
8	Authority Revocation List (ARL)	UNSPECIFIED
9	SPKI Certificate	UNSPECIFIED
10	X.509 Certificate - Attribute	UNSPECIFIED
11	Raw RSA Key (DEPRECATED)	RFC7296
12	Hash and URL of X.509 certificate	RFC7296
13	Hash and URL of X.509 bundle	RFC7296
14	OCSP Content	RFC4806
15	Raw Public Key	RFC7670
16-200	Unassigned	
201-255	Private use	RFC7296

is, that the public key is sent within the SubjectPublicKeyInfo structure. Due to the fact that the CERTREQ payload of the responder is in the IKE_SA_INIT response, it can not be fragmented. This might result in fragmentation of the message on IP level and ultimately in the potential loss in transit. Even if this issue could be mitigated, there would still be constraints with regards to the maximum payload size of 64kiB, which might get exceeded by some of the post-quantum signature schemes.

Using the digital signature method from RFC 7427 fulfills *SigR1 Quantum Security*, *SigR2 Compliance with protection goals Integrity, Authenticity and Non-repudiation*, *SigR4 Crypto-agility* and *SigR7 Practicability*, as it enables us to use a quantum-safe signature scheme protecting authenticity, integrity and non-repudiation. It also ensures crypto-agility by allowing drop-in replacements of the signature and the hash algorithms. The method is practical as there is no need for a registration of a new signature scheme but only a configuration change is required to exchange the signature scheme in use. On the downside, the digital signature method adds slightly more overhead compared to the previously described method of registering a new authentication method, conflicting with the requirement *SigR6 Minimalism*.

4.2 Multiple Authentication Exchanges (RFC 4739)

Hybrid authentication is a mechanism of authenticating more than once. In the context of post-quantum cryptography it is a common way to not depend on one cryptographic suite but combine the security of more than one suite. By implementing hybrid authentication, we realize the requirement *SigR3 Prevention of Security Reduction.*

To incorporate hybrid authentication we need to enable more than one IKE_AUTH exchange. RFC 4739 [EK06] specifies an IKEv2 protocol extension to run several rounds of authentication. The communication peers indicate support for multiple authentication exchanges by including a notify payload of type MULTIPLE_AUTH_SUPPORTED (16404). The initiator includes the notify payload in the IKE_AUTH request and the responder already announces the capability in the IKE_SA_INIT message. If both peers indicate support for multiple authentication exchanges, either peer can use the notify payload ANOTHER_AUTH_FOLLOWS (16405) in an IKE_AUTH exchange to let the other peer know, that he wishes to send another IKE_AUTH message. RFC 4739 does not make any assumptions on the type of authentications, it is compatible with signature schemes as well as with pre-shared key authentication and even with EAP authentication. Figure 4.5 shows an exemplary procedure where both peers authenticate twice using the digital signature authentication method as described in Section 4.1.2

Note that the Listing shows both peers authenticating twice. It is not mandatory that the peers authenticate the same number of times. To achieve out target of using a classic signature scheme and then one or more quantum-safe signature schemes, it makes sense to have the peers authenticate symmetrically. In our post-quantum scenario, this Listing might represent both peers first authenticate via a classic algorithm, e.g. RSA, and then use another signature algorithm, e.g. an isogeny-based signature scheme for quantum-resistant authentication.

Using multiple authentication methods requires the peers to know the mechanisms they want to authenticate with beforehand. Negotiation of an authentication mechanism is not possible.

```
Initiator                                        Responder
-----------                                      -----------
1. HDR, SAi1, KE, Ni,
   N(SIGNATURE_HASH_ALGORITHMS)    -->
                                            <--  2. HDR, SAr1, KE, Nr, [CERTREQ],
                                                    N(SIGNATURE_HASH_ALGORITHMS),
                                                    N(MULTIPLE_AUTH_SUPPORTED)
3. HDR, SK { IDi, [CERT+], [CERTREQ],
        [IDr], AUTH, SAi2, TSi, TSr,
        N(MULTIPLE_AUTH_SUPPORTED),
        N(ANOTHER_AUTH_FOLLOWS) }  -->
                                           <--  4. HDR, SK { IDr, [CERT+], AUTH,
                                                        N(ANOTHER_AUTH_FOLLOWS) }
5. HDR, SK { IDi, [CERT+], AUTH }  -->
                                           <--  6. HDR, SK { IDr, [CERT+], AUTH,
                                                             SAr2, TSi, TSr }
```

Figure 4.5: Initiator and Responder authenticating twice using the digital signature authentication method

Also the suggestions how to select a signature scheme proposed in RFC 7427 and discussed in Section 4.1.2 partly do not work anymore. The first option using the `IDr` payload is of no help because we want to use more than one signature scheme to authenticate one ID. Using the `CERTREQ` payload only helps finding a signature scheme for the first of the multiple authentication methods, as only the IKE_SA_INIT response and the first IKE_AUTH request contain the `CERTREQ` payload. If both peers want to use the same signature schemes in their multiple authentications, the third method, using the same key type as the initiator helps the responder to choose which authentication mechanisms to use. However, it seems to be the better alternative to agree on the authentication mechanisms via out-of-band communication.

One might argue that introducing several IKE_AUTH exchanges conflicts with the requirement *SigR6 Minimalism* as it adds significant overhead to the protocol. However, we introduced multiple exchanges to fulfill requirement *SigR3 Prevention of Security Reduction*. We can see that these two requirements are conflicting, we can't yet achieve them at the same time. A decision has to be made which one is more important. Relying on only one of the quantum-safe signature schemes solely is irresponsible as the confidence in those new cryptographic systems is too low. However, it is possible that in future, when we have more confidence in a post-quantum cryptographic system, one quantum-safe authentication method is sufficient to protect against classical attacks and attackers in posession of a quantum computer. Only then can hybrid authentication be discarded for the sake of simplicity.

4.3 Support for Payloads bigger than 64kiB

As some quantum-safe signature schemes generate signature data that exceeds the size of 64kiB, the IKEv2 protocol extension should consider enabling payloads bigger than 64kiB. The issue that a single payload can not be bigger than 64kiB is caused by the `Payload Length` field of the Payload Header being only 16 bit wide. This allows values from 0-65536 bytes. IKEv2 fragmentation can not solve this issue, due to the way it operates on IKEv2 messages.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Payload  |C|F| RESERVED  |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Key Exchange Method      |           RESERVED            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                        Key Exchange Data                      ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.6: Key Exchange Payload

Fragmentation takes all the payloads inside of the `SK{}` payload as binary blob, splits it up into smaller chunks of binary data and encrypts and integrity-protects those smaller chunks in the `SKF{}` payload. For successful reassembly the receiving peer needs to receive all fragments before it can start. Then it looks at each `Payload Length` field to know the length of the payload data. If that length can not be represented by an unsigned 16 bit value, the receiving peer does not know where the payload ends and a new payload begins. Thus the receiving peer will fail to reassemble the fragmented IKEv2 message.

There is an Internet Draft "draft-tjhai-ikev2-beyond-64K-limit" [THS20] that discusses approaches enabling payloads of bigger sizes. It generally proposes two solutions which are discussed in the following sections.

4.3.1 Hash and URL

The first solution is the Hash and URL mechanism. The Internet Draft mentions two options, how to transfer big public keys.

Key Exchange Payload

The first option is to utilize the key exchange payload which is shown in Figure 4.6. Important is the `F` bit in the payload header, which is normally part of the `RESERVED` field. If this bit is set to 0, the `Key Exchange Data` follows the specification of RFC 7296. If this bit is 1, the `Key Exchange Data` field contains the hash and URL value.

Certificate Payload

The second method of using the Hash and URL mechanism is to utilize the Certificate payload. There are two certificate encodings registered at IANA [IAN] that are relevant for Hash and URL, namely "Hash and URL of X.509 certificate" (12) and "Hash and URL of X.509 bundle" (13). The authors of the "draft-tjhai-ikev2-beyond-64K-limit" [THS20] mention, that the `CERT` payload is part of the IKE_AUTH exchange and therefore not the right place for key exchange data. As this work aims to send signature sizes exceeding the 64kiB limit, this argument is irrelevant.

```
Initiator                               Responder
-------------------------------------------------------------------------------
HDR, SAi1, KEi1, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED)*,
  N(INTERMEDIATE_EXCHANGE_SUPPORTED) --->

                                        HDR, SAr1, KEr1, Nr,
                                          N(IKEV2_FRAGMENTATION_SUPPORTED)*,
                                  <---    N(INTERMEDIATE_EXCHANGE_SUPPORTED)

HDR, SK{KEi2.1, KEi2.2, KEi2.3, ...} --->

                                  <---  HDR, SK{KEr2, ...}
*: optional
```

Figure 4.7: Bulk Transfer and Confirmation

Drawback of Hash and URL

The obvious drawback is that the peers must support downloading a file via HTTP and also operate a publicly available web server. This might be acceptable for endpoints that have a public IP, road warriors behind a NAT-Box in contrast technically can not provide their big payloads via a publicly available web server. Firewall rules or policies may also interfere with opening a port to the public to operate a web server. Although there are certain scenarios where the Hash and URL method might be acceptable, for a majority of application scenarios, this approach is at least cumbersome.

The Hash and URL approach not only resolves the requirement *SigR5 Support for payloads >64kiB* but also complies with *SigR6 Minimalism*. As mentioned above the need for the peers to support HTTP and run a public web server does not comply with *SigR7 Practicability*.

4.3.2 Payload Fragmentation

The second solution is “Payload Fragmentation” which uses a similar approach as the IKEv2 fragmentation. The Internet-Draft presents two possible Payload Fragmentation variants each having its advantages and disadvantages.

Bulk Transfer and Confirmation

The first one splits a big payload up into smaller payloads with a size below 64kiB but leaves the smaller payloads part of one IKEv2 message. The IKEv2 message fragmentation mechanism handles the fragmentation of the message before sending it on the wire. The following Figure 4.7 shows an exemplary IKE_INTERMEDIATE exchange with the “Bulk Transfer and Confirmation” taken from “draft-tjhai-ikev2-beyond-64K-limit” [THS20].

```
Initiator                                  Responder
-------------------------------------------------------------------------------
HDR, SAi1, KEi1, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED)*,
  N(INTERMEDIATE_EXCHANGE_SUPPORTED) --->

                                           HDR, SAr1, KEr1, Nr,
                                             N(IKEV2_FRAGMENTATION_SUPPORTED)*,
                                     <---    N(INTERMEDIATE_EXCHANGE_SUPPORTED)

HDR, SK{KEi2.1, ...}      --->

                                     <---  HDR, SK{}

HDR, SK{KEi2.2, ...}      --->

                                     <---  HDR, SK{}

HDR, SK{KEi2.3, ...}      --->

                                     <---  HDR, SK{KEr2, ...}

HDR, SK{}                 --->

*: optional
```

Figure 4.8: Incremental Transfer and Confirmation

Incremental Transfer and Confirmation

The second variant also splits the big payload up into smaller payloads but in contrast to the first variant, those payload fragments are sent as separate IKEv2 messages with their own IKEv2 header `HDR`. After the peer has received an IKEv2 message, it acknowledges the reception by sending an IKEv2 message containing just the header `HDR` and an empty payload `SK{}`. Only after having received the acknowledge of the last sent IKEv2 message, the initiator continues to send the next IKEv2 message containing the next part of the payload. Figure 4.8 again shows an `IKE_INTERMEDIATE` exchange of the "Inctemental Transfer and Confirmation" variant also taken from the "Beyond 64KB" draft.

4.3.3 Leveraging Payload Fragmentation to IKE_AUTH exchange

The "draft-tjhai-ikev2-beyond-64K-limit" mainly focuses on the payload size of the key exchange payload. However, they keep their specification generic enough so it also applies to other payloads. This work requires the capability to send an authentication payload exceeding the 64kiB limit. The IKEv2 initial exchanges with payload fragmentation and bulk transferring of the authentication payload at both peers is shown in Figure 4.9.

The initial exchanges with incremental transfer of the payload fragments applied to the authentication payload at both peers accordingly is shown in Figure 4.10.

Payload fragmentation resolves the requirement *SigR5 Support for Payloads >64kiB* and also

```
Initiator                                Responder
------------------------------------------------------------------------------
HDR, SAi1, KEi, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED) -->
                                     <-- HDR, SAr1, KEr, Nr, [CERTREQ,]
                                            N(IKEV2_FRAGMENTATION_SUPPORTED)

HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,]
         AUTHi.1, AUTHi.2, AUTHi.3 ,
         SAi2, TSi, TSr} -->
                                     <-- HDR, SK {IDr, [CERT,]
                                                  AUTHr.1, AUTHr.2, AUTHr.3,
                                                  SAr2, TSi, TSr}
HDR, SK {} -->
```

Figure 4.9: IKEv2 Initial Exchanges with Payload Fragmentation and Bulk Transfer of Authentication Payload

```
Initiator                                Responder
------------------------------------------------------------------------------
HDR, SAi1, KEi, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED) -->
                                     <-- HDR, SAr1, KEr, Nr, [CERTREQ,]
                                            N(IKEV2_FRAGMENTATION_SUPPORTED)

HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,]
         AUTHi.1, SAi2, TSi, TSr} -->
                                     <-- HDR, SK {}
HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,]
         AUTHi2, SAi2, TSi, TSr} -->
                                     <-- HDR, SK {}
HDR, SK {IDi, [CERT,] [CERTREQ,] [IDr,]
         AUTHi.3, SAi2, TSi, TSr} -->
                                     <-- HDR, SK {IDr, [CERT,]
                                                  AUTHr.1, SAr2, TSi, TSr}
HDR, SK {} -->
                                     <-- HDR, SK {IDr, [CERT,]
                                                  AUTHr.2, SAr2, TSi, TSr}
HDR, SK {} -->
                                     <-- HDR, SK {IDr, [CERT,]
                                                  AUTHr.3, SAr2, TSi, TSr}
HDR, SK {} -->
```

Figure 4.10: IKEv2 Initial Exchanges with Payload Fragmentation and Incremental Transfer of Authentication Payload

Table 4.3: Variety of Approaches towards a quantum-safe Signature Algorithm

Integrating a new Signature Algorithm	
New Authentication Method	RFC 7427 Digital Signature
Requesting a Signature Algorithm	
IDr Payload	CERTREQ Payload
Use Key Type of Initiator	No requesting
Indicating the Key Type in CERTREQ payload	
Private Use Range 201-255	Raw Public Key 15
Hybrid Authentication	
Multiple Authentication Exchanges	
Support Payload > 64kiB	
Hash and URL	Payload Fragmentation
Payload Fragmentation	
Bulk Transfer	Incremental Transfer

serves the requirement *SigR7 Practicability*, as all communication happens within the IKEv2 protocol. But it is not the most minimalistic approach as it adds computational complexity and even additional message exchanges in the case of the incremental transfer. That is the reason why this approach fails to comply with the requirement *SigR6 Minimalism*.

4.4 Final Protocol Design

The previous sections in this chapter showed multiple approaches of how to solve different problems with regards to the integration of quantum-safe authentication in IKEv2. In this section we now present the final protocol by explaining design decisions and showing an exemplary sequence of the protocol. Further we will discuss the combination of a quantum-safe key exchange together with the quantum-safe authentication elaborated in this protocol.

4.4.1 Design Decisions

Table 4.3 summarizes the provided approaches introduced in Sections 4.1 - 4.3 from which we need to choose towards the integration of a quantum-safe authentication in IKEv2.

To integrate a new signature algorithm in the IKEv2 protocol, the Digital Signature method proposed by RFC 7427 [KS15] is chosen. It mitigates most of the design flaws that come with the traditional authentication mechanism as described in Section 4.1.1. Moreover, the hash algorithm negotiation and the options to indicate the signature algorithm provide a good amount of flexibility without introducing too much complexity. Combining these properties with the fact that the Digital Signature method allows a simple exchange of the signature algorithm, aligning well with crypto-agility, makes it a good fit for the protocol extension for integrating a quantum-safe signature scheme into IKEv2.

Since the final protocol will incorporate multiple authentication exchanges, the methods to request a specific signature algorithm from the other peer shown in RFC 7424 [KS15] are rendered useless. Using the `IDr` payload to bind an ID to a signature algorithm does not work as we want to authenticate one ID with more than one signature algorithm. Using the `CERTREQ` payload only allows peers to request an algorithm which they use in the first round of authentication exchanges and is therefore not useful for the following authentication exchanges. For the responder it is possible to always authenticate with the same method as the initiator but has the drawback of lost flexibility. There is no reason for the responder to strictly follow the choices which the initiator made to authenticate itself. This is the reason why there will not be any requesting of a signature algorithm. It either assumes, that both peers are capable of using the same signature schemes or requires the peers to have some out-of-band communication. If one peer authenticates using a method which the other peer does not support, there is the AUTHENTICATION_FAILED error type that can be sent as response and potentially terminate the connection.

Since we do not utilize the `CERTREQ` payload to request a signature algorithm, we consequently do not need to indicate a key type within this payload.

For hybrid authentication multiple authentication exchanges as described in RFC 4739 [EK06] are incorporated. This approach offers the possibility to authenticate with one classic signature algorithm and one or more quantum-safe algorithms. It allows both peers to choose their signature schemes independent from the other peer and even allows the peers to make use of other authentication mechanisms like EAP or pre-shared key authentication.

To be able so send `AUTH` payloads containing signature data bigger than 64kiB, the protocol makes use of payload fragmentation. The alternative, Hash and URL, comes with constraints which drastically reduce the functional scope. As discussed in Section 4.3.1, peers behind a NAT cannot easily operate a publicly accessible web server and firewall and policy restrictions may also prevent peers from running a web server. Although the implementation of payload fragmentation adds complexity to the IKEv2 protocol and the need to transfer big data inside the protocol will slow down the operation, it is still considered the preferred option over the Hash and URL approach.

Within payload fragmentation the bulk transfer option as described in Section 4.3.2 is the preferred method of transmitting big payloads. It adds less complexity to IKEv2 and integrates seamlessly into the already established IKEv2 message fragmentation. The drawback is that the loss of one message fragment over the network requires the re-sending of all fragments again. However, the alternative would be to acknowledge each single message containing a bit of the split up big payload. This adds a much bigger delay to the protocol operation, since each peer has to wait for the next message to arrive over the network. In particularly lossy networks the incremental approach might be the better choice, however, usually the bulk transfer should perform much better. Due to these circumstances the final protocol design does not require one of the two approaches, but allows for both.

Table 4.4 shows the possible approaches to solve the challenges of integrating quantum-safe signature schemes into IKEv2 again, this time with the chosen approaches highlighted in green.

Table 4.4: Approaches chosen towards a quantum-safe Signature Algorithm

Integrating a new Signature Algorithm	
New Authentication Method	RFC 7427 Digital Signature
Requesting a Signature Algorithm	
IDr Payload	CERTREQ Payload
Use Key Type of Initiator	No requesting
Indicating the Key Type in CERTREQ payload	
Private Use Range 201-255	Raw Public Key 15
Hybrid Authentication	
Multiple Authentication Exchanges	
Support Payload > 64kiB	
Hash and URL	Payload Fragmentation
Payload Fragmentation	
Bulk Transfer	Incremental Transfer

4.4.2 Procedure of the Protocol

Figure 4.11 shows the initial exchanges of the elaborated IKEv2 protocol exemplary authenticating first with a classic signature algorithm and then two times with a quantum-safe algorithm. The first authentication mechanism does not need payload fragmentation, as the `AUTH` payload remains under the 64kiB limit. Both quantum-safe authentications in contrast do make use of payload fragmentation, which is indicated by using more than one `AUTH` payload (`AUTHi[23].1, AUTHi[23].2, ...` and `AUTHr[23].1, AUTHr[23].2, ...`).

A similar scenario using the incremental transfer and confirmation approach of payload fragmenting is shown in Figure 4.12. It uses just one classic and one quantum-safe authentication mechanism.

4.4.3 Integration with Quantum-safe Key Exchange in IKEv2

To not only achieve a quantum-safe authentication in IKEv2 but make it entirely quantum-safe, the elaborated protocol can be combined with the work on implementing a quantum-safe key exchange mechanism for IKEv2. As mentioned in Section 2.2.2 Heider's work [Hei19] introducing IKE_INTERMEDIATE as described in [Smy20] provides a full protocol extension to IKEv2. The elaborated protocol for quantum-resistant authentication presented in this work can be used alongside his work, resulting in an IKEv2 protocol which incorporates quantum-safe key exchange and authentication.

```
Initiator                                   Responder
-------------------------------------------------------------------------------
HDR, SAi1, KEi, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED),
  N(SIGNATURE_HASH_ALGORITHMS)      -->
                                        <-- HDR, SAr1, KEr, Nr, [CERTREQ,]
                                              N(IKEV2_FRAGMENTATION_SUPPORTED),
                                              N(SIGNATURE_HASH_ALGORITHMS),
                                              N(MULTIPLE_AUTH_SUPPORTED)

HDR, SK { IDi, [CERT+,] [CERTREQ,]
          [IDr,] AUTHi1, SAi2, TSi, TSr,
          N(MULTIPLE_AUTH_SUPPORTED),
          N(ANOTHER_AUTH_FOLLOWS) } -->
                                        <-- HDR, SK { IDr, [CERT+,] AUTHr1,
                                                      N(ANOTHER_AUTH_FOLLOWS) }
HDR, SK { IDi, [CERT+,]
          AUTHi2.1, AUTHi2.2, ...,
          N(ANOTHER_AUTH_FOLLOWS) } -->
                                        <-- HDR, SK { IDr, [CERT+,]
                                                      AUTHr2.1, AUTHr2.2, ...,
                                                      N(ANOTHER_AUTH_FOLLOWS) }
HDR, SK { IDi, [CERT+,]
          AUTHi3.1, AUTHi3.2, ... } -->
                                        <-- HDR, SK { IDr, [CERT+,]
                                                      AUTHr3.1, AUTHr3.2, ...,
                                                      SAr2, TSi, TSr }
HDR, SK { } -->
```

Figure 4.11: Initial Exchanges with three Authentication Rounds, two of them with Bulk Transmission of Fragmentated Payloads

```
Initiator                                    Responder
-----------------------------------------------------------------------------
HDR, SAi1, KEi, Ni,
  N(IKEV2_FRAGMENTATION_SUPPORTED),
  N(SIGNATURE_HASH_ALGORITHMS)      -->
                                        <-- HDR, SAr1, KEr, Nr, [CERTREQ,]
                                              N(IKEV2_FRAGMENTATION_SUPPORTED),
                                              N(SIGNATURE_HASH_ALGORITHMS),
                                              N(MULTIPLE_AUTH_SUPPORTED)

HDR, SK { IDi, [CERT+,] [CERTREQ,]
          [IDr,] AUTHi1, SAi2, TSi, TSr,
          N(MULTIPLE_AUTH_SUPPORTED),
          N(ANOTHER_AUTH_FOLLOWS) } -->
                                        <-- HDR, SK { IDr, [CERT+], AUTHr1,
                                                      N(ANOTHER_AUTH_FOLLOWS) }
HDR, SK { IDi, [CERT+,] AUTHi2.1 }  -->
                                        <-- HDR, SK {}
HDR, SK { IDi, [CERT+,] AUTHi2.2 }  -->
                                        <-- HDR, SK {}
                                      ...

HDR, SK { IDi, [CERT+,] AUTHi2.n }  -->
                                        <-- HDR, SK {IDr, [CERT+,]
                                                     AUTHr2.1, SAr2, TSi, TSr}
HDR, SK {} -->
                                        <-- HDR, SK {IDr, [CERT+,]
                                                     AUTHr2.2, SAr2, TSi, TSr}
HDR, SK {} -->
                                      ...
                                        <-- HDR, SK {IDr, [CERT+,]
                                                     AUTHr2.n, SAr2, TSi, TSr}
HDR, SK {} -->
```

Figure 4.12: Initial Exchanges with two Authentication Rounds, one of them with Incremental Transmission of Fragmentated Payload

4.4.4 Compliance with Requirements

After having developed the protocol for an integration of quantum-safe authentication into IKEv2, we take a look at the requirements, defined in Section 3.3, to check the compatibility of the protocol with those requirements.

The requirement *SigR1: Quantum Security* has been fulfilled, as the protocol enables us to freely choose several signature schemes that can incorporate signature data bigger than 64kiB. It is up to the signature scheme to ensure the second requirement *SigR2: Compliance with protection goals Authenticity, Integrity and Non-repudiation.* Due to the flexible way of incorporating several signature schemes into the IKE_AUTH exchanges, the requirement is fulfilled if the user uses a signature scheme which complies with these protection goals. In the end it is the user's responsibility to fulfill this requirement, but as the protocol design enables the user to comply with these goals, we therefore consider this requirement as fulfilled. However, note that it is possible for a user to use a signature scheme that does not protect one or more of authenticity, integrity and non-repudiation. The third requirement, *SigR3: Prevention of Security Reduction* is fulfilled by having more than one IKE_AUTH exchange, enabling the peers to first authenticate with a classic signature scheme and then in one or more further exchanges incorporating quantum-safe authentication mechanisms. This way we do have at least the security of the classic algorithm. *SigR4: Crypto-Agility* has also been taken into account by using the "Digital Signature" authentication method in combination with multiple authentication exchanges. This allows the peers to easily replace one or more of the incorporated authentication mechanisms with other mechanisms if needed. The requirement *SigR5: Support for payloads >64kiB* is fulfilled by implementing the work from the IETF draft "Beyond 64KB Limit of IKEv2 Payload" [THS20] in the IKE_AUTH exchange. The last two requirements *SigR6: Minimalism* and *SigR7: Practicability* are non-functional requirements which makes them harder to evaluate. The protocol has been designed with minimalism in mind, however, decisions have been made that do not comply with minimalism, e.g. the decision to not use the Hash and URL approach for payloads bigger than 64kiB. We explained these decisions, but consider them a compromise that had to be made to get a functional, practical and reliable protocol. Although we payed close attention to develop a protocol without design flaws the *Sig7: Practicability* requirement can ultimately only be proven by fully implementing the protocol and performing tests and measurements, simulating a real-world scenario. We consider the last two requirements as not fulfilled.

Table 4.5 shows the requirements defined in Section 3.3 and indicates whether each requirement has been fulfilled or not, as described in the above paragraph. If a requirement has been fulfilled, this is shown by a check mark symbol on green background in the last column. A requirement that has no been met is shown as a "x" on red background.

Table 4.5: Requirements Fullfillment in the final Protocol Design

ID	Requirement	Fulfilled
SigR1	Quantum-Security	✓
SigR2	Compliance with protection goals Authenticity, Integrity and Non-Repudiation	✓
SigR3	Prevention of security reduction	✓
SigR4	Crypto-agility	✓
SigR5	Support of Payloads > 64kiB	✓
SigR6	Minimalism	x
SigR7	Practicability	x

5 Proof of Concept

To show the feasibility of the elaborated protocol towards a quantum-resistant authentication in IKEv2 based on isogenies, an implementation is provided. Section 5.1 provides an implementation of the isogeny-based signature scheme. Section 5.2 discusses the integration of the elaborated protocol into iked, the IKEv2 daemon of OpenBSD.

5.1 Digital Signature Scheme based on Supersingular Isogenies

In [YAJ+17] Yoo et al. not only propose a digital signature scheme based on supersingular isogenies, but they also provide an implementation of their approach[1]. Based on the SIDH library provided by Microsoft [2,3], the implementation utilizes a key generation, a signing and a verifying algorithm, which makes it a promising candidate for a working digital signature scheme. That is the reason why this implementation has been chosen as a starting point for the signature scheme that has been integrated into the IKEv2 protocol. Yoo et al. implement their signature scheme using SIDHp751 of the PQCrypto-SIDH from Microsoft and achieve 128 bits of post-quantum security by choosing to perform 248 rounds of zero-knowledge proof as part of their algorithm. However, taking a closer look at their implementation, some drawbacks and issues become obvious.

5.1.1 Drawbacks and issues of the implementation of Yoo et al.

This section describes drawbacks and issues of the implementation of Yoo et al.

OpenBSD support: The isogenysignature implementation provides support to the operating systems Microsoft Windows and GNU/Linux. It supports the C compiler GNU GCC[gcc] and the Clang Compiler [cla] which makes it possible to port the application to OpenBSD.

Memory management: Not only does the isogenysignature implementation leak memory by not freeing allocated memory after usage but it also accesses memory at uninitialized addresses. This leads to random segmentation faults and undefined behaviour. Although the Linux kernel seems to be quite open to illegal memory access, the implementation of `malloc(3)` in OpenBSD is a lot more stricter. This results in the signature scheme not being usable after having been ported to OpenBSD. Figure 5.1 shows the summary of a Valgrind([val]) analysis of the `kex_test` program offered by the isogenysignature implementation. The sourcecode has not been altered and has been built with following command: `make ARCH=x64 CC=gcc ASM=FALSE GENERIC=FALSE` on a 5.8 Linux kernel and

[1]https://github.com/yhyoo93/isogenysignature
[2]https://www.microsoft.com/en-us/research/project/sidh-library
[3]https://github.com/microsoft/PQCrypto-SIDH

an Intel Core I7-4600M. Note that slight changes have been made in the Makefile, disabling optimization (`-O0`) and adding the debug flag to the compiler command (`-g`). This is necessary to analyse programs with Valgrind.

Unruhs construct: Unruhs construct is not correctly applied in the implementation of the signature scheme. The idea behind Unruhs construct is explained in Section 3.2.3. In summary, the signer creates a hash based on input parameters. Depending on this hash a bit is generated for every round of the underlying NIZK, which determines which response is included in the signature. The verifier computes the same hash (based on the data he gets from the signature) and generates the same bit from the hash for every round. Depending on that bit, the verifier knows, which response he expects in the signature and can verify the response. In contrast to Unruh's construct, the signing algorithm of the isogenysignature implementation puts both responses (one for challenge 0 and one for challenge 1) of each round into the signature data and sends the signature over to the verifier. The verifier then computes a hash from data that differs from the approach of Unruh and depending on that hash he generates a bit. This bit determines which response the verifier checks for validity. Due to the fact that both responses are valid responses (but for different challenges), the verification is always successful. What kind of data has been used to generate the aforementioned hash and how the bit is generated is not essential, as long as the responses contained in the signature are valid responses of their types. This leads to the signature proving that the signer is capable of computing isogenies, but it does not authenticate the author of a message nor does it protect the integrity of a message.

Message signing: The isogenysignature implementation does not sign a message. This issue correlates with the implementation not applying Unruh's construct correctly. The signing algorithm does not generate a challenge hash at all and the verifying algorithm does compute the challenge hash, but not including the message as partial input of the challenge hash. It is mandatory for both parties to generate the hash and also to incorporate the message in the hash generation as this is the mechanism to ensure integrity of the message. If the message has been altered in transit, the verifier notices it by the hash being different from the one the signer calculated. This results in different bits for each round of verification. At the first bit, that differs from the one the signer has used, the verifier would fail his verification and reject the message. If the messages are not incorporated into the challenge hashes, these hashes and therefore also the bits for the rounds of the underlying NIZK can be the same for the signer and the verifier even though the message has been altered in transit. This makes it impossible for the verifier to detect whether the message has been altered in transit or not.

Outdated SIDH library: The version of the SIDH library used for the implementation of the isogeny-based signature scheme could not be clearly identified. The latest version of the SIDH library as at the date of writing this work is v3.3 (2020-10-19). Comparing this version to the one used in the isogenysignature implementation leads to the assumption that an older, nowadays outdated version of the SIDH library has been used. This might have a negative impact on the efficiency of the isogeny based cryptography that is used via that library and therefore lower the performance of the signature scheme.

Unsafe Random Generator: There are several points in the process of the isogeny-based signature scheme where randomness is necessary. Most notably the signing party has to generate a random point R on which he calculates his commitments in every round of the underlying NIZK. The implementation uses the `rand()` function which generates pseudo random numbers based on a seed. The problem is that the behaviour of the rand function is deterministic, which means if given the same seed the function produces the same random number. This is not acceptable in high security scenarios, as the random number depends on the state of a system which might be exploited by an attacker.

```
==5902== LEAK SUMMARY:
==5902==    definitely lost: 8,739,020 bytes in 17,550 blocks
==5902==    indirectly lost: 0 bytes in 0 blocks
==5902==      possibly lost: 0 bytes in 0 blocks
==5902==    still reachable: 0 bytes in 0 blocks
==5902==         suppressed: 0 bytes in 0 blocks
==5902==
==5902== ERROR SUMMARY: 23590 errors from 34 contexts (suppressed: 0 from 0)
```

Listing 5.1: Valgrind memory errors summarized

As the implementation elaborated in this work is just for the purpose of proving the concept, we do not fix the last two drawbacks being "Outdated SIDH Library" and "Unsafe Random Generator".

5.1.2 Porting to OpenBSD

The first task that needed to be done in order to port the signature scheme implementation to OpenBSD was writing a new makefile. The syntax for the makefile of the GNU Make utility differs from the syntax the OpenBSD version of `make(1)` supports. Two of the most notable differences are that inclusion statements, conditional and loop structures start with a single dot in OpenBSD makefiles and that some of the conditional statements (like `ifeq`) are not available in OpenBSD makefiles. Also OpenBSD makefiles enclose variables in curly braces whereas GNU makefiles enclose variable names in parenthesis. For more information on the syntax of the different Make utilities please consult the respective manuals, [bsda] for the OpenBSD makefile syntax and [gnu] for the GNU makefile syntax.

The second step in porting the implementation to OpenBSD is defining preprocessor directives, just like they have been defined for Microsoft Windows and GNU/Linux. The program sometimes uses OS specific system calls which makes it necessary for the program to know on which OS it is currently running. As GNU/Linux and OpenBSD are both largely POSIX compliant, the system calls of GNU/Linux and OpenBSD do not differ and the program calls the same system calls no matter if it is run on GNU/Linux or OpenBSD. Microsoft Windows does not comply with the POSIX standards. Therefore, different system calls are used in most cases.

The third change that has to be made in the process of porting the implementation to OpenBSD was adapting the `#include` preprocessor directives, as some functions in OpenBSD need different inclusions than on other operating systems. This relates e.g. to the `open()` function call that needs the inclusion of the header file `sys/stat.h` which defines the mode a new file is created with.

5.1.3 Improving Memory Management

The biggest issue regarding memory management was the function call `fpcopy751` in the file `fpx.c` which copies a field element of 96 bytes to another field element of the same size. This function was called with an input of 48 bytes which led to the illegal memory access. The issue appears in the SIDH library from Microsoft which is outdated, newer versions of the SIDH library mitigate this problem. This work uses the outdated SIDH library, but mitigated the issue by using the `copy_word()` function also declared in `fpx.c`. This function behaves the same way as the `fpcopy751` function with the difference, that the length of bytes to copy has to be handed over as input parameter. By specifying the right amount of bytes to be copied the illegal memory access problem is resolved.

Another problem that has been detected using Valgrind shows that hardly any allocated memory regions are freed when they are not needed anymore. This fact has also been confirmed by OpenBSD's memory leak detection using `MALLOC_OPTION` "D". It leads to memory leaks which result in high main memory usage and in the worst case in the program crashing. Freeing these memory allocations as soon as they are not needed anymore mitigates this issue and results in a leak free, stable running program.

5.1.4 Applying Unruhs Transformation

In their work [YAJ+17], Yoo et al. describe their signature scheme with three algorithms, namely a key generation algorithm, a signing algorithm and a verifying algorithm. This is intuitively expected when it comes to signature schemes. The signing algorithm and the verifying algorithm are affected by Unruh's Transformation described in Section 3.2.3. Yoo et al. adapt Unruh's transformation correctly to use a cocatenation of the public key and a message (pk, m) instead of a claim of identity (x). However, they do not apply Unruh's transformation to their implementation and, on top of that, their signature data differs from the data they describe in their paper. The first change applied to the signature scheme implementation was to fix the signature data. The left side of the following listing (5.2) shows the modified signature struct. The right side (5.3) shows the signature struct of the original implementation.

```
struct Signature {
    unsigned char
        *com[NUM_ROUNDS][2];
    uint8_t
        *ch[NUM_ROUNDS];
    unsigned char
        *h[NUM_ROUNDS][2];
    unsigned char
        *resp[NUM_ROUNDS];
};
```

Listing 5.2: Modified signature struct

```
struct Signature{
    unsigned char
        *Commitments1[NUM_ROUNDS];
    unsigned char
        *Commitments2[NUM_ROUNDS];
    unsigned char
        *HashResp;
    unsigned char
        *Randoms[NUM_ROUNDS];
    point_proj
        *psiS[NUM_ROUNDS];
};
```

Listing 5.3: Initial signature struct

Note that both structures contain the commitments for each round, just represented in a slightly different way. The original implementation does not contain the challenge bit in the signature struct, which later makes it impossible to know which response is part of the signature. This field has been added to the modified signature structure. Further both structures contain the hash field, again represented in a slightly different way. However, the original implementation fills the field incorrectly as it does not take into account the challenge bit. This issue has been resolved in the modified implementation. Each round `i` of the underlying NIZK has a challenge bit `ch[i]`. If the challenge bit equals zero, the `resp[i][0]` has to be hashed and stored in the `h[i][0]`. If the challenge bit is one, `resp[i][1]` has to be hashed and stored in `h[i][0]`. The other response is also hashed and stored in `h[i][1]`.

Based on the content of the first three data fields of the modified struct, it is possible to generate a challenge hash `H` from which the bitstring `J_1|| ... ||J_i` is derived. This bitstring together with the `ch[i]` determines which of the responses (`resp[i][0] or resp [i][1]`) is part of the actual signature. The resulting response is stored in the last field of the modified signature struct named `resp[i]`. In contrast, the original implementation simply stored both possible responses in the fields `Randoms[i]` and `psiS[i]`, as they do not apply Unruhs construct. This is a significant waste of memory and makes the signature size bigger than it needs to be. But also including both possible responses into the signature data conflicts with the idea behind Unruhs transformation where only one possible response is sent to the verifier, together with necessary information of how to verify it.

Due to the fact, that the two possible responses use a different amount of memory, one has to be cautious in allocating memory for the `resp[i]` field. One idea is to pad the shorter response to use the same amount of memory as the bigger response, however, this again would waste a substantial amount of memory depending on the number of rounds of the underlying NIZK. Allocating a different amount of memory for the `resp[i]` for each round has implications on the receiver side as the verifier does not know how to parse the signature before generating the challenge hash and deriving the bitstring `J_1|| ... ||J_i`.

On the receiver's side, the verifier has to parse the first three data fields being `com[i][2]`, `ch[i]` and `h[i][2]`. These values can be parsed by the verifier, only because the length of those fields is known. The verifier has to generate the challenge hash from the public key of the signer, the message itself and the values of these parsed data fields and derive the bitstring `J_1|| ... ||J_i`. Combining the knowledge of the `ch[i]` and the bitstring, the verifier knows which kind of response is in each field `resp[i]` and can allocate memory and parse the expected response. Then the verifier can check the response for the criteria of the expected type of response. If all those checks succeed, the verifier can be sure that the signer is the one he claims to be and further that the message has not been altered in transit.

5.1.5 Message Integration

The original implementation of Yoo et al. does not incorporate a message which is signed by their signature scheme. Although it is theoretically possible to not sign a message but just prove ownership of a secret key, this is not what we want to achieve with a signature scheme. The integration of a message only changes the challenge hash as it is concatenated with the public key of the signer and other parameters that can be found in Algorithm 3.4 and Algorithm 3.5.

Note that is critical to include the message in the signature scheme to ensure the integrity of the message. The receiver of a message does not only want to know that the sender actually owns the identity he claims to have but also wants to be sure that the message has not been altered in transit.

5.1.6 Providing an API to the Isogenysignature scheme

To provide a well-defined uniform access to the functionality of the signature scheme based on supersingular isogenies, it has been compiled as shared library. The functions implemented by the shared library can be seen from the header file `SISig.h` provided as part of the library. This header file is shown in the following Figure 5.4.

```c
#define OBYTES 48
#define PBYTES 96
#define PRIV_KEY_LEN OBYTES      //Privatekey len in bytes
#define PUB_KEY_LEN 8*PBYTES     //Publickey len in bytes

struct SigData {
    unsigned char *sig;
    unsigned int siglen;
};

int
SISig_P751_Keygen(unsigned char *PrivateKey, unsigned char *PublicKey);

struct SigData *
SISig_P751_Sign(char *msg, unsigned char *PrivateKey, unsigned char *PublicKey);

int
SISig_P751_Verify(char *msg, struct SigData *sigdata, unsigned char *PublicKey);

unsigned char *
SISig_P751_Read_Pubkey(char *file);

unsigned char *
SISig_P751_Read_Privkey(char *file);

int
SISig_P751_Write_Pubkey(unsigned char *PublicKey, char *file);

int
SISig_P751_Write_Privkey(unsigned char *PrivateKey, char *file);
```

Listing 5.4: Functions of Shared Library of Isogeny-based Signature Scheme

The define-statements serve the length of the public key and the private key to programs using the API. The `SigData` struct provides a simple structure containing the signature data itself as well as the length of the data, which simplifies further parsing of the signature data. The first three functions defined by the header file expose the algorithms specifying a signature scheme.

`SISig_P751_Keygen` generates a signing key and a public key, storing these keys in the memory pointed to by the `PrivateKey` and `PublicKey` pointers provided by the function caller. On success, the function returnes 0, otherwise, -1 is returned.

`SISig_P751_Sign` generates the signature data based on a message `msg`, a signing key

`PrivateKey` and a public key `PublicKey`. The function allocates memory for the generated signature and returns a pointer to the `struct SigData` filled with the signature data itself and the length of the signature data. Note that it is the responsibility of the function caller to free the allocated memory for the `struct SigData`, after it is not used anymore, to avoid memory leaks.

`SISig_P751_Verify` takes a message `msg`, a signature `sigdata` and a public key `PublicKey`, to verify the authenticity and integrity of the message. On success the function returns 0, otherwise, -1 is returned.

The remaining four functions `SISig_P751_(Read|Write)_(Priv|Pub)key` read/write the secret/public key from/to the file specified in `file` string. In the case of reading, the result gets stored in a pointer to unsigned chars provided by the function caller. When writing, the functions return 0 on success, otherwise, -1 is returned.

5.2 Integration in OpenBSDs iked

This work does not provide the full implementation of the protocol described in Section 4.4, as this would have exceeded the scope of the work. Nevertheless, analyzing the iked daemon especially with a view on what needs to be adapted to fit to the new elaborated protocol, provides a good starting point for an implementation of the full protocol extension. This chapter gives an overview of the internal structure of iked as well as discusses ways to implement the protocol extension proposed in Chapter 4

5.2.1 OpenBSD iked

The IKEv2 daemon iked [ikeb] which is part of the OpenBSD [ope] base system has been chosen as starting point for the integration of the protocol extension for several reasons. The first reason is that it operates on OpenBSD being an operating system that has been designed with security deeply integrated ([bsdb], [Raa]). Iked follows this philosophy providing a minimalistic and secure codebase that makes maintenance easier and allows for an overview of the security incorporated in the project. It is designed to fully run in user land, for interaction with the kernel the PF_KEY Key Management API PFKEYv2 specified in [MMP98] is used. Iked makes use of three privilege separated processes as shown in Figure 5.1 (taken from [Flo], slightly modified). There is the parent process that is responsible for parsing and loading the configuration and controlling the other two processes. The configuration is on the one hand taken from a configuration file which defaults to `/etc/iked.conf` and on the other hand comes from the additional utility “ikectl” [ikea] which is able to change configuration parameters during the runtime of the iked daemon. The ca process is responsible for operations related with certificates and key actions, whereas the ikev2 process handles message composition and transmission. Interprocess communication is done using imsgs which is a mechanism using sockets for exchanging messages between local processes. The implementation guarantees imsgs are received as the whole message. For more details on imsgs, see [ims].

A simple configuration to establish an IPSec connection between the two networks 192.168.0.0/24 and 192.168.1.0/24 using a DH key exchange and a pre-shared key for authentication might look like shown in Figure 5.2

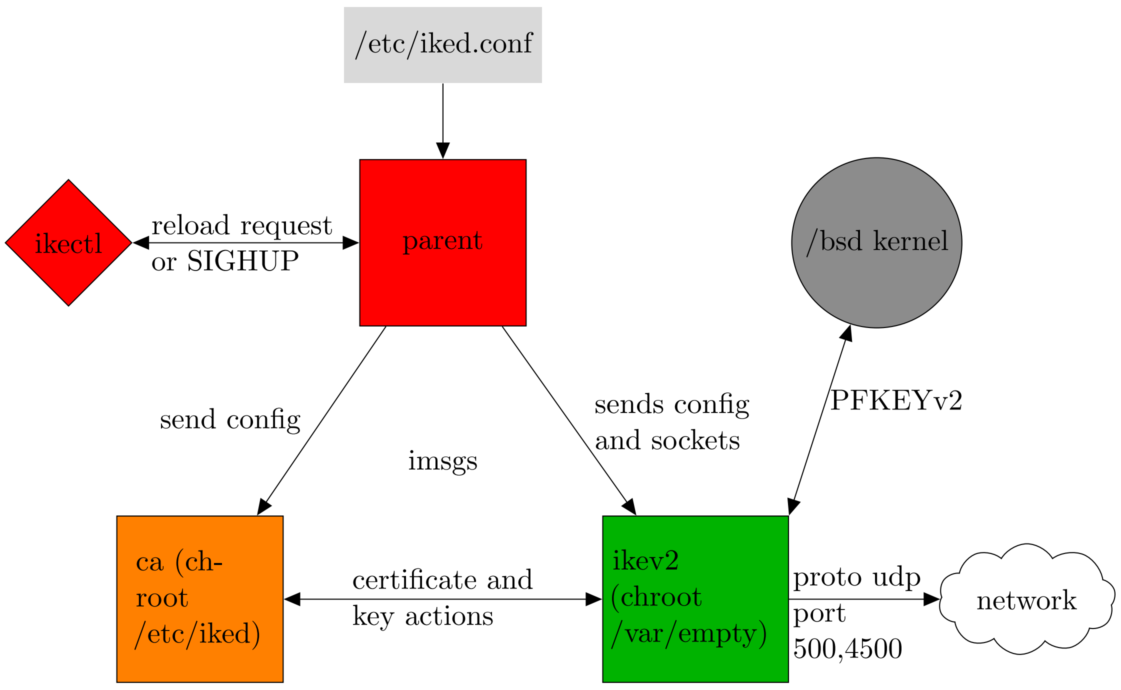

Figure 5.1: OpenBSD iked privilege separation model

```
ikev2 active esp from 192.168.0.0/24 to 192.168.1.0/24 \
        peer 192.168.1.1 \
        psk supersecretpassword
```

Figure 5.2: Exemplary iked.conf Configuration File

```
-----BEGIN SISIG PRIVATE KEY-----
d617605485a29c20fa1565c42db93ecc9bcb26191c724b47
7ac2f9a4b9baf08dcf50e257f37b75ed8ed7b1b990f08501
-----END SISIG PRIVATE KEY-----
```

Figure 5.3: Exemplary encoding of Isogeny-based Secret Key

```
struct imsg {
    struct imsg_hdr  hdr;
    int              fd;
    void            *data;
};
```

Figure 5.4: Imsg Data Structure

```
    uint32_t        type;
    uint16_t        len;
    uint16_t        flags;
    uint32_t        peerid;
    uint32_t        pid;
};
```

Figure 5.5: Imsg Header Data Structure

5.2.2 Integration of a new Signature Method

Iked implements Digital Signature Authentication as specified in RFC 7427 [KS15] allowing a straightforward integration of the further signature schemes. The challenge in integrating a new signature scheme lies in parsing the public and secret keys as the iked implementation uses the crypto library [liba] provided by LibreSSL [libb]. New signature schemes that are not supported by the crypto library need to deal with the encoding and parsing of the keys on their own. A simple solution might be to represent the key data in hexadecimal format, two chars per byte, and indicate the key type in meta data provided with the key data. Figure 5.3 shows a potential representation of a secret key of the isogeny-based signature scheme. Iked could then parse the first line and find out the actual used signature scheme.

To actually incorporate the keygeneration, signing and verifying algorithms in iked, the cleanest approach is to provide a library with a header file defining the API to the signature scheme. This allows iked to call a function provided by the API while maintaining a separation between iked and the signature scheme.

5.2.3 Dealing with imsgs bigger 16384 bytes

As mentioned in Section 5.2.1 iked uses imsgs to exchange messages between local processes using UNIX domain sockets. The current implementation of the imsg functionality allows the sending of messages of up to 16384 bytes. This limit is not sufficient in the context of quantum-resistant algorithms. Most of the quantum-safe algorithms either have public keys or signature data, that exceeds this internal limit. As this data needs to be passed between the ca and the ikev2 process, we must think of a way to get around this 16kiB limit. The proposed solution is to send more than one imsg which needs adaptions at both processes. The sending process needs to split the buffer containing either the public key or the signature data up into chunks smaller than 16kiB. Additionally, the sending process needs to indicate to the receiving process, that it will receive more than one chunk, which needs to be reassembled. This can be done with the imsg `type` which is specified in the `imsg_header` struct, which is part of the `imsg` struct. Those data structures are shown in Figure 5.4 and 5.5.

```
struct ibuf_truncated {
    uint16_t curr_no;
    uint16_t total;
    uint8_t *data;
};
```

Figure 5.6: Truncated Ibuf Structure

To enable the receiving peer to reassemble fragmented imsgs, the sender provides metadata together with the chunk of data itself, which is shown in Figure 5.6. This metadata consists of the sequence number of the current chunk and is represented by the `curr_no` variable. The second part of the metadata is the total number of chunks the big data has been split into, represented by `total`. This enables the receiver to reassemble the chunks in the correct order and gives information on the number of outstanding imsgs.

5.2.4 Incorporating multiple IKE_AUTH Messages

To enable the iked implementation to incorporate more than one IKE_AUTH message, it first needs to indicate the capability of multiple authentication to the other peer. Therefore, as described in Section 3.1.4, the initiator sends a notify payload in the first IKE_AUTH request, whereas the responder already sends the notify payload in the IKE_SA_INIT response. To add the notify payload we need to modify two functions, as the exchanges differ depending on a peer acting as initiator or as responder. The initiator needs to add the `MULTIPLE_AUTH_SUPPORTED` payload in the `ikev2_init_ike_auth()` function executed by the ikev2 process. The responder uses the `ikev2_resp_ike_sa_init()` function to add the notify payload indicating support for multiple authentications.

After the peers have mutually indicated the capability to authenticate more than once, each peer can add the `ANOTHER_AUTH_FOLLOWS` notify payload in their IKE_AUTH message to express the wish to authenticate one more time. This payload needs to be added in the function `ikev2_init_ike_auth()` if the peer is the initiator, otherwise in the `ikev2_resp_ike_auth()` function.

The peer receiving an IKE_AUTH message recognizes the presence of an `ANOTHER_AUTH_FOLLOWS` notify payload in the `ikev2_pld_payloads()` function and needs to behave different from the case where this payload is not present. Most notably, the peer must not change the state of the SA to "authenticated" after having verified the authentication payload, but must wait for at least one more IKE_AUTH exchange. A peer can only assume the SA to be authenticated successfully after having received and verified an IKE_AUTH payload *without* the `ANOTHER_AUTH_FOLLOWS` notify payload, as this is how the sending peer indicates that it does not wish to authenticate with another method.

5.2.5 Dealing with Payloads bigger 64kiB

In the context of quantum-safe authentication incorporating the protocol specified in Chapter 4 the only payload that might exceed the 64kiB limit is the `AUTH` payload containing the signature data. The `ikev2_next_payload()` function handles the adding of another payload. The first

```
ikev2 active esp from 192.168.0.0/24 to 192.168.1.0/24 \
        peer 192.168.1.1 \
        rfc7427 rsa sisig
```

Figure 5.7: Examplary iked.conf Configuration File

step in this function is to check if the size of the payload exceeds 64kiB. A mechanism needs to be implemented so that the function will, in case of a payload bigger than 64kiB, not return an error, but split the payload into smaller chunks as described in Section 4.3.

If the "Bulk Transfer and Confirmation" approach is chosen, it is sufficient to just split a large payload into smaller payloads and put all of them into one IKEv2 message. In a scenario where both peers use signature schemes with signature data exceeding the limit, this approach requires one more IKEv2 message which is the confirmation of the IKE_AUTH response. The initiators IKE_AUTH request is implicitly confirmed by the responder sending the IKE_AUTH response.

Using the "Incremental Transfer and Confirmation" approach, the process gets more complicated. Every smaller chunk of data needs to be transferred in an separate IKEv2 message and has to be confirmed separately by the receiver before the usual protocol procedure can continue. This is more intrusive to the protocol and requires more changes to the current implementation.

The same holds true on the receiving peer where the bulk transfer approach only requires the parsing of more than one `AUTH` payload and reassembling the data. Additionally the initiator peer has to acknowledge the receipt of the full IKE_AUTH message from the responder to ensure correct transfer of the data. The function `ikev2_pld_payloads` detects that the `AUTH` payload is split up into smaller chunks and is responsible for reassembling the payload.

In contrast, the incremental transfer approach requires the receiving end to parse each IKEv2 message, store the content temporarily, acknowledge each received message and wait for the next message to arrive. In this case also the `ikev2_pld_payloads()` function would detect that the `AUTH` payload is split over more than one IKEv2 message. But the behaviour of the function needs to differ from the bulk transfer case so that the confirmation message gets prepared and sent to the other peer.

5.2.6 Adapting the Configuration and the Configuration-Parser

The last thing that needs to be taken into account when integrating quantum-safe signature schemes into IKEv2 is adapting the configuration. This mainly happens in the file `parse.y`, which is responsible for reading in the configuration file. The new signature algorithm needs to be configurable in the configuration file. Also, with hybrid authentication more than one signature scheme has to be configurable. A potential configuration file might look like the following Figure 5.7 establishing an ESP tunnel between the two networks 192.168.0.0/24 and 192.168.1.0/24 where the other peer has the IP-address 192.168.1.1 using two Digital Signature (RFC 7427) authentication methods being RSA and the isogeny-based SISig signature scheme.

The `ikectl(8)` [ikea] utility also needs adaptions which enable it to generate secret and public keys for the added signature schemes.

6 Evaluation

After having discussed the proof-of-concept implementation of the isogeny-based signature scheme and the IKEv2 protocol extension, we evaluate both. In Section 6.1 we show the performance of the implemented signature scheme, followed by explaining the security of the scheme in Section 6.2. Finally we also present security considerations of the IKEv2 protocol extension in Section 6.3.

6.1 Performance Analysis of the isogeny-based Signature Scheme

We evaluate the isogeny-based signature scheme proposed in Chapter 3 and implemented as described in Chapter 5 with regards to performance in a real world scenario. Therefore we run experiments on three different machines with different operating systems. We use two notebooks, one with a 5.9 Linux kernel and an Intel Core i5-6300 processor and one with OpenBSD and an Intel Core i7-4600 CPU. The third machine is a workstation running Linux 5.8 and an Intel Core i7-8700 processor.

Figure 6.1 shows the execution time for key generating, signing and verifying algorithms for each of these machines (average over 100 measurements). The time (in milliseconds) which the algorithm needs to finish is shown on the y-axis. It is logarithmic to base 10. We can see from the figure, that the key generation algorithm takes significant less time ($\sim$ 10ms) than the signing and verifying algorithm ($\sim$ 4000ms$-$8000ms). This is expected, since the key generation algorithm does not perform multiple rounds of the incorporated zero-knowledge proof as described in Section 3.2. Calculating isogenies is the most time consuming operation. Table 6.1 shows the computations, each algorithm executes. The key generation algorithm chooses a random point $\mathcal{S}$ and generates one isogeny $\phi(\mathcal{S})$. The signing algorithm runs a determined number of rounds and within each round it computes two isogenies as part of the commitment, as described in Section 3.2.4. The verifying algorithm also runs the same number of rounds and either calculates two isogenies or calculates one isogeny and triples one point to check its order, depending on the challenge bit. In Table 6.1, computational expensive operations are highlighted with red background color. Overall, a complete execution of the isogeny-based signature scheme takes up to 13 seconds on the two slower dual core processors and around seven seconds on the faster hexa-core processor. To put these time measurements into context, the RSA signature scheme using SHA256 (executed on the i5-6300) needs approximately 100ms for the process of generating a key-pair, signing a message with the secret key and verifying the message with the public key.

The measurements present the result of the signature scheme running in one thread. To improve performance, we let the signature scheme run on the various machines using all threads the machine can work on concurrently. Although in theory all of the three CPUs support hyper-threading, i.e. being able to execute four threads concurrently, only two

Table 6.1: Calculation steps done by Key-generation, Signing and Verifying algorithm

Algorithm	**Input**	**Computation**	
Key-generation	E, S	$\phi : E \to E/\langle S\rangle$	executed once
Signing	$E, E/\langle S\rangle, R$	$\psi : E \to E/\langle R\rangle$ $\phi' : E/\langle R\rangle \to E/\langle R, S\rangle$ $\phi(R), \quad \psi(S)$	executed over multiple rounds
Verifying	$E, E/\langle S\rangle,$ $E/\langle R\rangle, E/\langle R, S\rangle$ **case1:** $R, \phi(R)$ **case2:** $\psi(S)$	**case1:** check order of R and $\phi(R)$ $\psi : E \to E/\langle R\rangle$ $\psi' : E/\langle S\rangle \to E/\langle R, S\rangle$ **case2:** check order of $\psi(S)$ $\phi' : E/\langle R\rangle \to E/\langle R, S\rangle$	

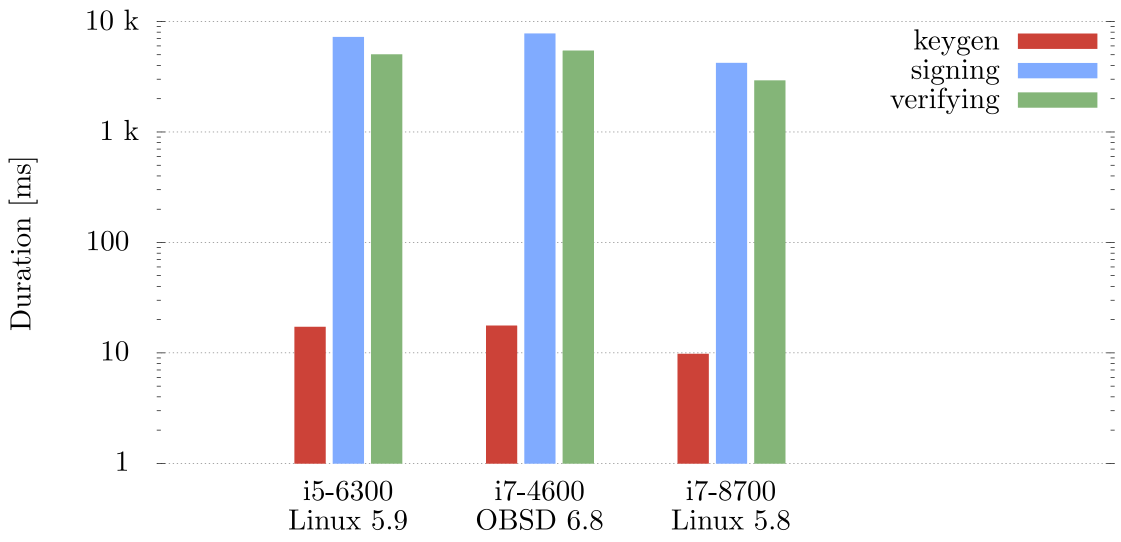

Figure 6.1: Single-threaded Runtime of Key generation, Signing and Verifying Algorithm

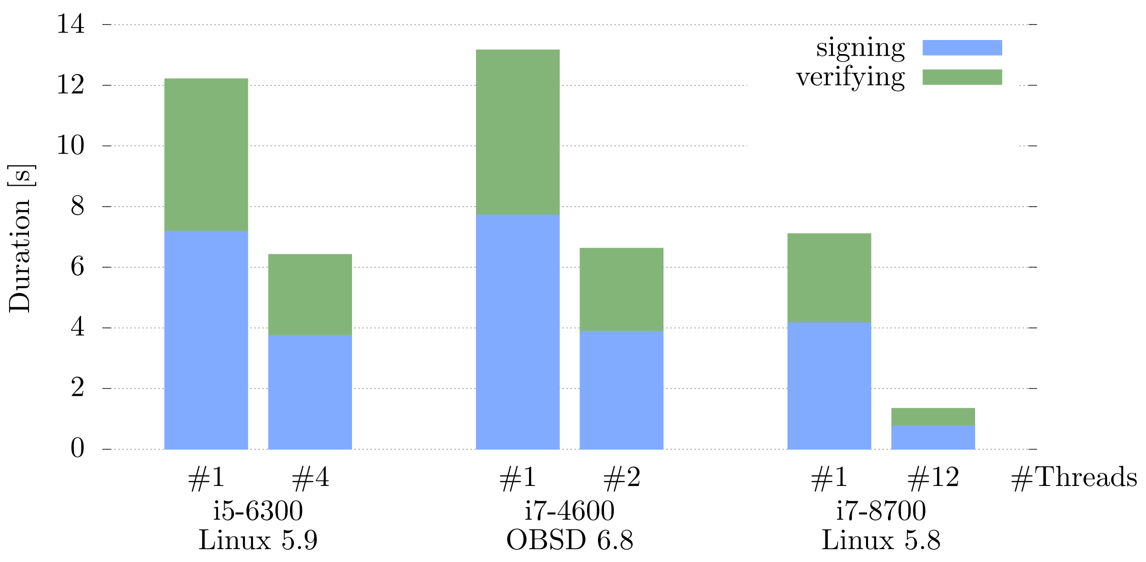

Figure 6.2: Performance Improvements achieved by Parallelization

threads have been spawned on the i7-4600, since OpenBSD disabled hyper-threading for security reasons. On the i5-6300, 4 threads have been spawned, and we utilized even 12 threads on the i7-8700. For spawning new threads, we used the `pthreads`-API, defined in the POSIX.1 standard (see [SR13, §11-12]). The execution of the rounds in the signing and in the verifying algorithm can be easily parallelized, as computational steps of these rounds do not correlate.

Figure 6.2 only shows the signing algorithm and the verifying algorithm, as this allows to have a linear y-axis, again showing the runtime of each algorithms. The runtime of the key generation is negligible and has therefore not been included on the revised stacked bar chart. The figure shows the performance improvement on each CPU when using multiple threads. In the group of two bars, the left bar shows the runtime of the algorithms computed on one thread (indicated with #1), whereas the right bar shows the runtime utilizing more threads (indicated with #2, #4 and #12). The performance improves approximately by a factor of two on the dual core processors and by a factor of 5 on the i7-8700. Furthermore, on the most powerful CPU, the i7-8700, both the signing and verifying algorithm finish in under one second, improving the practicability of the signature scheme.

As mentioned in Section 5.1.1 the signature scheme does not use a current implementation of the PQCrypto-SIDH library. Current implementations of this library provide an optimization for processors which already have the ADX extension. This extension allows the use of further assembly instructions, which speed up the field arithmetic computations notably. Migrating the signature scheme to this newer library would result in a performance gain on processors of the Broadwell generation or newer.

Summarizing the runtime evaluation, we see that the performance of the isogeny-based signature scheme is far from optimal taking more than 10 seconds on conventional dual core processors. However, we have seen that considerable improvements can be achieved by tweaking the algorithms to run CPU-intensive tasks concurrently. The existence of an improved PQCrypto-SIDH library and ongoing research in post quantum algorithms means we can expect further improvements on the runtime of isogeny-based cryptography in general.

6.2 Security Considerations of the isogeny-based Signature Scheme

This section discusses the underlying security of the isogeny-based signature scheme from a theoretical perspective in Section 6.2.1. Afterwards, in Section 6.2.2 the security of the proof-of-concept implementation is taken into consideration.

6.2.1 Theoretical Foundation

De Feo et al. define five problems in their work towards quantum-resistant isogeny-based cryptographic systems, the following two being relevant to their zero-knowledge proof [DFJP14, §5].

Problem 6.1 (Computational Supersingular Isogeny (CSSI) problem)**.** Let $\phi_A : E_0 \to E_A$ be an isogeny whose kernel is $\langle [m_A]P_A + [n_A]Q_A \rangle$, where m_A and n_A are chosen at random from $\mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ and not both divisible by ℓ_A. Given E_A and the values $\phi_a(P_B), \phi_A(Q_B)$, find a generator R_A of $\langle [m_A]P_A + [n_A]Q_A \rangle$.

Problem 6.2 (Decisional Supersingular Product (DSSP) problem)**.** Given an isogeny $\phi : E_0 \to E_3$ of degree $\ell_A^{e_A}$ and a tuple sampled with probability 0.5 from one of the following two distributions:

- (E_1, E_2, ϕ'), where the product $E_1 \times E_2$ is chosen at random among those $\ell_B^{e_B}$-isogenous to $E_0 \times E_3$, and where $\phi' : E_1 \to E_2$ is an isogeny of degree $\ell_A^{e_A}$, and
- (E_1, E_2, ϕ'), where E_1 is chosen at random among the curves having the same cardinality as E_0, and $\phi' : E_1 \to E_2$ is a random isogeny of degree $\ell_A^{e_A}$,

determine from which distribution the tuple is sampled.

These problems are believed to be hard to solve for both, a classic computer as well as a quantum computer.

Further De Feo et al. prove that under the assumptions of CSSI and DSSP their proposed identification scheme is zero-knowledge and therefore meeting the requirements of a sigma protocol shown in Section 3.2.2[DFJP14, §6.2].

Applying Unruh's transformation to this proof of identity results in an NIZK proof that is simulation-sound and online extractable in the random oracle model as discussed in Section 3.2.3. According to Unruh [Unr15, Therorem 18] we get a strongly unforgeable signature scheme in the random oracle model from this NIZK.

6.2.2 Security of the Implementation

To evaluate the security of the isogeny-based signature scheme implementation, this section discusses security relevant topics. We take a look at the PQCrypto-SIDH library provided by

Microsoft, discuss the used random number generators and hash functions before the memory management is examined from a security perspective. These topics are particular important for the implementation as they are the cryptographic foundation, the signature scheme is based on.

PQCrypto-SIDH Library The implementation as described in Section 5.1 incorporates the PQCrypto-SIDH library from Microsoft [pqc]. This implementation claims to protect against timing and cache-timing attacks by using regular, constant time operations on all secret key material. As mentioned in Section 5.1.1 the signature scheme incorporates an outdated SIDH library, but there is no information on security issues of the older version which would reduce the security of the signature scheme.

Random Number Generator The implementation uses the `rand()` and `srand()` calls from the C standard library. The usual procedure is to initialize the `srand()` function with a seed that is used by the `rand` function, to generate pseudo random numbers between 0 and `RAND_MAX`. The call to `srand()` is deterministic, meaning, if one calls the function with the same seed twice, the same sequence of random numbers is returned by the `rand()` function. This enables an attacker who either knows or guesses the right seed to generate the same pseudo random numbers as the actual program and thus to compute the same values. For example an attacker might be able to compute the same secret key as its victim empowering the attacker to authenticate as the victim.

The OpenBSD operating system replaced the standard `rand()/srand()` implementations with the one from `arc4random(3)` [arc]. This eliminates the problem described above by ignoring the call to `srand()` and instead requesting pseudo random numbers from the kernel incorporating different mechanisms to ensure randomness. However, when using this signature scheme on systems that do not replace the standard random mechanism with `arc4random`, this is a security threat and must be taken into account when using the scheme for more than just a proof of concept.

Hash Function The isogeny-based signature scheme implementation utilizes a minimal SHA-3 implementation. This implementation has not been analyzed with regards to security as this is not within the scope of this work. Using the signature scheme in scenarios beyond a proof of concept, one should consider replacing the prototypical implementation with either the reference implementation or another more mature implementation of SHA-3.

Memory Management Section 5.1.3 has already discussed the measures taken to eliminate bogus memory access and memory leaks. It is important to ensure correct memory access to prevent the signature scheme to crash or, in the worst case, allow an attacker to execute malicious code by illegal memory access. The implementation has been tested with Valgrind [val] and OpenBSD's internal `malloc`-option to dump statistics [mal]. Both tools confirmed that the program is free from memory leaks and illegal memory access. Figure 6.3 shows the Heap and Error summary of Valgrind on the implemented signature scheme.

```
==90062== HEAP SUMMARY:
==90062==     in use at exit: 0 bytes in 0 blocks
==90062==   total heap usage: 4,037 allocs, 4,037 frees, 1,309,878 bytes
    allocated
==90062==
==90062== All heap blocks were freed -- no leaks are possible
==90062==
==90062== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figure 6.3: Valgrind Heap and Error summary of improved Memory Management

6.3 Security Considerations of the IKEv2 Protocol Extension

The security considerations for plain IKEv2 are discussed in [KHN+14, §5]. This work adds extensions to IKEv2 interfering with the protocol procedure. As a result it is necessary to take a look at the implications for the security of the protocol.

Signature Authentication (RFC 7427) [KS15] The use of the Digital Signature authentication method as shown in Section 4.1.2 allows a flexible combination of hash algorithms and signature schemes. This enables the user to choose from any available hash algorithm and signature scheme, of which some are more secure than others. The user might choose a secure signature scheme in combination with a less secure hash function or the other way round. It is important to understand that the achieved security is just as high as the weakest algorithm and might result in users thinking their authentication is harder to break than it actually is.

One might think that negotiation of the hash algorithm in the IKE_SA_INIT exchanges might be prone to downgrading attacks. A man-in-the-middle might change the `SIGNARURE_HASH_ALGORITHM` notify payload of either the initiator or the responder, forwarding a slightly changed payload indicating support for only one unsecure hash function. The other peer thinks its communication partner only supports this kind of hash algorithm and makes use of it. Note that the man-in-the-middle would be detected in the IKE_AUTH message, where the peer whose `SIGNARURE_HASH_ALGORITHM` has been altered, composes its `AUTH` payload signing, inter alia, this payload. The man-in-the-middle could exploit the unsecure hash algorithm if he can generate the same hash from the altered payload as the originating peer from the actual payload. As hash algorithms always map any data to a fixed size digest, one can not eliminate the case, where two inputs result in the same digest and so there is the theoretical possibility for this to happen. Nevertheless, this is something we have to accept dealing with hash algorithms and common hash algorithms that can be used in the `SIGNARURE_HASH_ALGORITHM` (see "IKEv2 Notify Message Types - Status Types" by [IAN]) can reduce the probability of a hash collision to a point where it is negligible.

Downgrade attacks modifying the `AUTH` payload of an IKE_AUTH message are of less concern as the peer receiving the message would fail to verify the `AUTH` payload with the public key of the sending peer.

Multiple Authentication Exchanges (RFC 4739) As proposed by RFC 7296 [KHN+14], in IKEv2 the responder discloses his identity only after the initiator has been successfully authenticated. This changes with the initiator using multiple exchanges to authenticate. The initiator is only considered authenticated by the responder once he has reveived a message of type IKE_AUTH which does not contain an `ANOTHER_AUTH_FOLLOWS` notify payload. If the initiator sends an IKE_AUTH message to the responder with the `ANOTHER_AUTH_FOLLOWS` notify payload, the responder does not yet consider the initiator as authenticated, but in his response he already reveals his identity in the `IDr` payload which is part of the IKE_AUTH response. An attacker might exploit this by breaking a weak authentication method of an initiator, authenticating to the responder as the initiator with this broken authentication mechanism and promising a more secure authentication to follow. As soon as the attacker has seemingly authenticated as the actual initiator, the responder reveals its identity and the attacker can gather information related to its identity. A SA is never established as the attacker can not complete its own authentication at the peer.

Beyond 64kiB Limit of IKEv2 Paylod The ability of a peer to accept IKEv2 messages containing payloads bigger than 64kiB makes the peer vulnerable to (distributed) Denial of Service (DOS) attacks where an attacker initiates an IKE_AUTH exchange sending huge payloads in its IKE_AUTH message. The responder has to parse these payloads and perform computations on the received data using resources like memory and CPU time. If the attacker sends a big enough amount of IKE_AUTH messages, this can exceed the responder's capability to handle all those requests. As a result, the victim may be unable to process any requests from non-malicious peers. For a peer to protect itself from one single rogue peer, RFC 7296 proposes to use Cookies as described in [KHN$^+$14, §2.6], though this method does not provide protection from a botnet e.g. where the attacker uses more than one IP address. See [NS16] for measurements against (D)DOS attacks.

7 Conclusion and Future Work

No one can safely predict the progress of research in the field of quantum computing. We need to expect a breakthrough in the development of a large-scale quantum computer. Thus we need to prepare for a scenario where today's cryptography is rendered unsecure. While there exists a protocol extension to IKEv2 realizing a quantum safe key exchange, this approach does not consider authenticity, arguing that it would require a quantum computer in real-time to exploit authenticity. Although this argument is true, with the arrival of a quantum computer with enough computing power, forging a wrong identity becomes a real threat.

This work aims to tackle the lack of quantum-safe authentication in IKEv2. We can achieve our goal by incorporating a quantum-safe signature scheme in the initial exchanges of IKEv2 by adapting the IKE_SA_INIT and IKE_AUTH exchanges. This work integrates an isogeny-based signature scheme in IKEv2, which is assumed to hold against attacks utilizing a quantum computer. We provide an extensive description of isogeny-based cryptography in general and of an isogeny-based signature scheme in particular. Moreover an overview of the IKEv2 protocol is given.

In the next step of this work, an extensive analysis of the IKEv2 environment and post-quantum cryptography results in requirements for a successful integration of a quantum-safe authentication. One important finding of the requirements analysis is to not develop a protocol extension requiring one specific signature scheme, but to ensure *crypto-agility*. This enables the drop-in replacements of various authentication mechanisms, which is vital in a scenario where one authentication mechanism is broken and needs to be replaced. Another finding is the need for hybrid authentication, which allows to authenticate with a classic, more mature algorithm and additionally with a new quantum-resistant algorithm. In a case, where the new quantum-resistant algorithm is broken, we can still rely on the security of the classic algorithm protecting against a classic attacker. But if the quantum-resistant algorithm holds, we additionally get protection from an attacker utilizing a quantum computer. By hybrid authentication, we *prevent a reduction of security.*

Finally we design a protocol extension allowing quantum-resistant authentication in IKEv2. Combining the flexible *Digital Signature* authentication method with *multiple authentication exchanges* allows hybrid authentication where the selected signature schemes can easily be exchanged. It is even possible to authenticate using a classic authentication mechanism followed by several quantum-safe mechanisms. Furthermore, since signature data potentially exceeds the IKEv2-internal limit of 64kiB, we need the protocol extension to *fragment payloads >64kiB* to avoid this limit.

A proof-of-concept implementation is provided, to see how the signature scheme performs in a real-world scenario. We start using an existing implementation of an isogeny-based NIZK that has major flaws and turn it into an isogeny-based signature scheme by eliminating the flaws and adding functionality. Further we discuss in detail an implementation of the elaborated protocol extension of IKEv2 and point out challenges that need to be considered when implementing the protocol.

Focusing on performance and security, we give an evaluation of the isogeny-based signature scheme. We observe high runtimes of the signing and the verification algorithms, but the improvement by parallelization and current research give hope that the overall performance will improve significantly. The extended IKEv2 protocol is also evaluated from a security perspective, by discussing the introduced approaches necessary for a quantum-resistant authentication. We show that the protocol extension is not prone to downgrading attacks and how to mitigate the risk of man-in-the-middle and DOS attacks.

This work shows the feasibility of quantum-resistant authentication in IKEv2 fulfilling the functional requirements of *quantum-security*, *compliance with the protection goals integrity, authenticity and non-repudiation*, *prevention of security reduction*, *crypto-agility* and *support for payloads bigger than 64kiB*. The introduction of multiple authentication exchanges as well as the interchangeability of signature mechanisms and the payload fragmentation all add additional complexity to the otherwise simple IKEv2 protocol. This is why we could not accomplish compliance with the requirement of being *minimalistic*. Maybe in future, the need for hybrid authentication becomes unnecessary, as soon as we have gained enough confidence to trust in new quantum-resistant authentication mechanisms. This would again reduce the complexity of the protocol. However, we will not be able to reduce the complexity introduced by the interchangeability of the signature mechanisms, as this would lower the crypto-agility that has shown to be vital for cryptography. The payload fragmentation has to be introduced because of an internal shortcoming of IKEv2 specifying the payload size in a 16bit wide field. It appears unlikely that this will ever be subject to change, as this would mean changes in the core of IKEv2 affecting backwards compatibility and interoperability of the protocol significantly. *Practicability* can only be proven by a full implementation of the provided protocol and evaluating the implementation in a real-world scenario which is left to future work.

The performance analysis of the isogeny-based signature scheme confirmed the well known fact that isogeny-based cryptography suffers from suboptimal runtime. It is subject to research to further improve the performance of isogeny-based cryptography and a lot of improvement has already been achieved on this topic. As mentioned in Chapter 5, the implementation of the isogeny-based signature scheme provided in this work does not use a current implementation of the PQCrypto-SIDH library. We can expect a performance improvement already by porting the signature scheme to an up-to-date implementation of the PQCrypto-SIDH library, which is also left for future work. The isogeny-based signature scheme may be extended by adding alternative security parameters. The provided implementation only operates on the field arithmetic over the prime P751. While this offers the highest security at the expense of computing time, the signature scheme might benefit from an implementation utilizing other security parameters providing less security but improving the performance.

Finally, combining this protocol with the aforementioned quantum-resistant key exchange that has already been developed for IKEv2 enables us for the first time to establish an IPSec communication, securing both confidentiality and authenticity against an attacker in possession of a large-scale quantum computer and is also subject to future work.

This work including the source code of the proof-of-concept implementations is freely available, hoping for and enabling further research on this topic.

Glossary

API Application Programming Interface. 67, 84

CA Certificate Authority. 9, 44, 84

DH Diffie-Hellman. 1, 9, 16, 65, 84

DLP Discrete Logarithm Problem. 1, 16, 84

DOS Denial of Service. 77, 80, 84

DSA Digital Signature Algorithm. 1, 7, 14, 16, 26, 39, 84

DSS Digital Signature Standard. 7, 84

EAP Extensible Authentication Protocol. 39, 44, 46, 84

ECC Elliptic Curve Cryptography. 14–17, 84

ECDH Elliptic Curve Diffie-Hellman. 1, 14, 16, 84

ECDLP Elliptic Curve Discrete Logarithm Problem. 15, 16, 84

ECDSA Elliptic Curve Digital Signature Algorithm. 1, 7, 14, 15, 25, 39, 41, 84

EdDSA Edward-curves Digital Signature Algorithm. 7, 84

ESP Encapsulating Security Payload. 69, 84

HTTP Hypertext Transfer Protocol. 27, 49, 84

HVZK Honest Verifier Zero-Knowledge. 31, 84

IANA Internet Assigned Numbers Authority. 26, 41, 44, 48, 84

IETF Internet Engineering Task Force. 36, 57, 84

IKEv2 Internet Key Exchange Protocol version 2. vii, 2, 3, 5, 7, 9, 12, 13, 20, 22, 23, 25–30, 33, 35–37, 39, 41, 43, 46–50, 52–54, 57, 59, 65, 69, 71, 76, 77, 79, 80, 84

IP Internet Protocol. 69, 84

IPSec Internet Protocl Security. vii, 2, 7, 12, 65, 80, 84

KEM Key Encapsulation Mechanism. 23, 27, 84

MTU Maximum Transmission Unit. 26, 84

NAT Netword Address Translation. 49, 53, 84

NIST National Institute of Standards and Technology. 7, 15–17, 36, 84

NIZK Non-interactive Zero-knowledge Proof. 28, 30, 31, 60, 61, 63, 74, 79, 84

PKCS Public Key Cryptographic System. 1, 2, 84

PRF Pseudo Random Function. 9, 84

RFC Request for Comments. 84

RSA Rives, Shamir and Adleman. 1, 7, 14, 16, 26, 27, 36, 39, 46, 69, 84

SA Security Association. 7, 9, 12, 13, 23, 25, 68, 84

SHA-1 Secure Hash Algorithm 1. 25, 41, 84

SHA-2 Secure Hash Algorithm 2. 84

SHA-3 Secure Hash Algorithm 3. 42, 75, 84

SIDH Supersingular Isogeny Diffie-Hellman. 20, 84

SPD Security Policy Database. 84

SPI Security Parameter Index. 9, 13, 84

TCP Transport Control Protocol. 84

TLS Transport Layer Security. 84

UDP User Datagram Protocol. 26, 84

List of Figures

Bibliography

[arc] *arc4random(3).* : *arc4random(3)*, `https://man.openbsd.org/arc4random.3`. – Last checked: 2021-01-17

[Bar13] Barker, Elaine B.: Digital Dignature Standard (DSS) / National Institute of Standards and Technology. 2013 (186-4). – Federal Inf. Process. Stds. (NIST FIPS). – Last checked: 2021-01-17, Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[Bar20] Barker, Elaine B.: Recommendation for Key Management: Part 1 - General / National Institute of Standards and Technology. 2020 (800-57 Pt1 Rev 5). – Special Publication (NIST SP). – Last checked: 2021-01-17, Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[BCR+18] Barker, Elaine B. ; Chen, Lidong ; Roginsky, Allen L. ; Vassilev, Apostol T. ; Davis, Richard: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography / National Institute of Standards and Technology. 2018 (800-56Ar3). – Special Publication (NIST SP). – Last checked: 2021-01-17, Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

[bsda] *OpenBSD Make Utility.* : *OpenBSD Make Utility*, `https://man.openbsd.org/make`. – Last checked: 2021-01-17

[bsdb] *OpenBSD Security.* : *OpenBSD Security*, `https://www.openbsd.org/security.html`. – Last checked: 2021-01-17

[cas] *The Case for Elliptic Curve Cryptography.* : *The Case for Elliptic Curve Cryptography*, `https://web.archive.org/web/20090117023500/http://www.nsa.gov/business/programs/elliptic_curve.shtml`. – Last checked: 2021-01-17

[CCJ+16] Chen, Lily ; Chen, Lily ; Jordan, Stephen ; Liu, Yi-Kai ; Moody, Dustin ; Peralta, Rene ; Perlner, Ray ; Smith-Tone, Daniel: *Report on post-quantum cryptography.* Bd. 12. US Department of Commerce, National Institute of Standards and Technology, 2016

[Che17] Chen, L.: Cryptography Standards in Quantum Time: New Wine in an Old Wineskin? In: *IEEE Security Privacy* 15 (2017), Nr. 4, S. 51–57. `http://dx.doi.org/10.1109/MSP.2017.3151339`. – DOI 10.1109/MSP.2017.3151339

[cla] *Clang: a C language family frontend for LLVM.* : *Clang: a C language family frontend for LLVM*, `https://clang.llvm.org`. – Last checked: 2021-01-17

[CMS+15] Córcoles, Antonio D. ; Magesan, Easwar ; Srinivasan, Srikanth J. ; Cross, Andrew W. ; Steffen, Matthias ; Gambetta, Jay M. ; Chow, Jerry M.: Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. In: *Nature communications* 6 (2015), Nr. 1, S. 1–10

[DFJP14] De Feo, Luca ; Jao, David ; Plût, Jérôme: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: *Journal of Mathematical Cryptology* 8 (2014), Nr. 3, S. 209–247

[DH76] Diffie, Whitfield ; Hellman, Martin: New directions in cryptography. In: *IEEE transactions on Information Theory* 22 (1976), Nr. 6, S. 644–654

[DH17] Deering, S. ; Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200 (Internet Standard). Version: Juli 2017. `http://dx.doi.org/10.17487/RFC8200` (Internet Request for Comments)

[DS13] Devoret, Michel H. ; Schoelkopf, Robert J.: Superconducting circuits for quantum information: an outlook. In: *Science* 339 (2013), Nr. 6124, S. 1169–1174

[EK06] Eronen, P. ; Korhonen, J.: *Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol.* RFC 4739 (Experimental). Version: November 2006. `http://dx.doi.org/10.17487/RFC4739` (Internet Request for Comments)

[Fis05] Fischlin, Marc: Communication-efficient non-interactive proofs of knowledge with online extractors. In: *Annual International Cryptology Conference* Springer, 2005, S. 152–168

[Flo] Floeter, Reyk: *IKEv2 VPN with OpenBSD iked(8)*, `https://www.openbsd.org/papers/eurobsdcon2010-iked.pdf`. – Last checked: 2021-01-17

[FS86] Fiat, Amos ; Shamir, Adi: How to prove yourself: Practical solutions to identification and signature problems. In: *Conference on the theory and application of cryptographic techniques* Springer, 1986, S. 186–194

[FS07] Fu, D. ; Solinas, J.: *IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA).* RFC 4754 (Proposed Standard). Version: Januar 2007. `http://dx.doi.org/10.17487/RFC4754` (Internet Request for Comments)

[gcc] *GCC, the GNU Compiler Collection.* : *GCC, the GNU Compiler Collection*, `https://gcc.gnu.org/`. – Last checked: 2021-01-17

[gnu] *GNU Make Utility.* : *GNU Make Utility*, `https://www.gnu.org/software/make/manual/`. – Last checked: 2021-01-17

[HC98] Harkins, D. ; Carrel, D.: *The Internet Key Exchange (IKE).* RFC 2409 (Proposed Standard). Version: November 1998. `http://dx.doi.org/10.17487/RFC2409` (Internet Request for Comments). Obsoleted by RFC 4306, updated by RFC 4109

[Hei19] Heider, Tobias: *Towards a Verifiably Secure Quantum-Resistant Key Exchange in IKEv2*, Ludwig-Maximilians Universität München, Diplomarbeit, 2019

[IAN] IANA: *Internet Key Exchange Version 2 (IKEv2) Parameters.* `https://www.iana.org/assignments/ikev2-parameters`. – Last checked: 2021-01-17

[ikea] *ikectl(8).* : *ikectl(8)*, `https://man.openbsd.org/man8/ikectl.8`. – Last checked: 2021-01-17

[ikeb] *OpenIKED.* : *OpenIKED*, `https://www.openiked.org`. – Last checked: 2021-01-17

[ims] *IMSG_INIT(3).* : *IMSG_INIT(3)*, `https://man.openbsd.org/imsg_init.3`. – Last checked: 2021-01-17

[JDF11] Jao, David ; De Feo, Luca: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: *International Workshop on Post-Quantum Cryptography* Springer, 2011, S. 19–34

[Kau05] Kaufman (Ed.), C.: *Internet Key Exchange (IKEv2) Protocol.* RFC 4306 (Proposed Standard). Version: Dezember 2005. `http://dx.doi.org/10.17487/RFC4306` (Internet Request for Comments). Obsoleted by RFC 5996, updated by RFC 5282

[KBF+15] Kelly, Julian ; Barends, Rami ; Fowler, Austin G. ; Megrant, Anthony ; Jeffrey, Evan ; White, Theodore C. ; Sank, Daniel ; Mutus, Josh Y. ; Campbell, Brooks ; Chen, Yu u. a.: State preservation by repetitive error detection in a superconducting quantum circuit. In: *Nature* 519 (2015), Nr. 7541, S. 66–69

[KHN+14] Kaufman, C. ; Hoffman, P. ; Nir, Y. ; Eronen, P. ; Kivinen, T.: *Internet Key Exchange Protocol Version 2 (IKEv2).* RFC 7296 (Internet Standard). Version: Oktober 2014. `http://dx.doi.org/10.17487/RFC7296` (Internet Request for Comments). Updated by RFCs 7427, 7670, 8247

[KHNE10] Kaufman, C. ; Hoffman, P. ; Nir, Y. ; Eronen, P.: *Internet Key Exchange Protocol Version 2 (IKEv2).* RFC 5996 (Proposed Standard). Version: September 2010. `http://dx.doi.org/10.17487/RFC5996` (Internet Request for Comments). Obsoleted by RFC 7296, updated by RFCs 5998, 6989

[Kiv11] Kivinen, T.: *Secure Password Framework for Internet Key Exchange Version 2 (IKEv2).* RFC 6467 (Informational). Version: Dezember 2011. `http://dx.doi.org/10.17487/RFC6467` (Internet Request for Comments)

[KK10] Karpfinger, Christian ; Kiechle, Hubert: *Kryptologie - Algebraische Methoden und Algorithmen.* Springer, 2010

[Kob87] Koblitz, Neal: Elliptic curve cryptosystems. In: *Mathematics of computation* 48 (1987), Nr. 177, S. 203–209

[KS05] Kent, S. ; Seo, K.: *Security Architecture for the Internet Protocol.* RFC 4301 (Proposed Standard). Version: Dezember 2005. `http://dx.doi.org/10.17487/RFC4301` (Internet Request for Comments). Updated by RFCs 6040, 7619

[KS15] Kivinen, T. ; Snyder, J.: *Signature Authentication in the Internet Key Exchange Version 2 (IKEv2).* RFC 7427 (Proposed Standard). Version: Januar 2015. `http://dx.doi.org/10.17487/RFC7427` (Internet Request for Comments)

[KWT16] Kivinen, T. ; Wouters, P. ; Tschofenig, H.: *Generic Raw Public-Key*

Support for IKEv2. RFC 7670 (Proposed Standard). Version: Januar 2016. `http://dx.doi.org/10.17487/RFC7670` (Internet Request for Comments)

[liba] *crypto(3)*. : *crypto(3)*, `https://man.openbsd.org/man3/crypto.3`. – Last checked: 2021-01-17

[libb] *LibreSSL*. : *LibreSSL*, `https://www.libressl.org`. – Last checked: 2021-01-17

[mal] *malloc(3)*. : *malloc(3)*, `https://man.openbsd.org/malloc.3`. – Last checked: 2021-01-17

[Mil85] Miller, Victor S.: Use of elliptic curves in cryptography. In: *Conference on the theory and application of cryptographic techniques* Springer, 1985, S. 417–426

[MKJR16] Moriarty (Ed.), K. ; Kaliski, B. ; Jonsson, J. ; Rusch, A.: *PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017 (Informational). Version: November 2016. `http://dx.doi.org/10.17487/RFC8017` (Internet Request for Comments)

[MMP98] McDonald, D. ; Metz, C. ; Phan, B.: *PF_KEY Key Management API, Version 2*. RFC 2367 (Informational). Version: Juli 1998. `http://dx.doi.org/10.17487/RFC2367` (Internet Request for Comments)

[Mos18] Mosca, Michele: Cybersecurity in an era with quantum computers: will we be ready? In: *IEEE Security & Privacy* 16 (2018), Nr. 5, S. 38–41

[MP19] Mosca, Michele ; Piani, Marco: *Quantum Threat Timeline Report*. https://globalriskinstitute.org/publications/quantum-threat-timeline/, 2019. – https://globalriskinstitute.org/download/quantum-threat-timeline-full-report-2/

[NIS16] NIST: Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process / National Institute of Standards and Technology. 2016. – Forschungsbericht

[NS16] Nir, Y. ; Smyslov, V.: *Protecting Internet Key Exchange Protocol Version 2 (IKEv2) Implementations from Distributed Denial-of-Service Attacks*. RFC 8019 (Proposed Standard). Version: November 2016. `http://dx.doi.org/10.17487/RFC8019` (Internet Request for Comments)

[ope] *OpenBSD*. : *OpenBSD*, `https://www.openbsd.org`. – Last checked: 2021-01-17

[Piz90] Pizer, Arnold K.: Ramanujan graphs and Hecke operators. In: *Bulletin of the American Mathematical Society* 23 (1990), Nr. 1, S. 127–137

[Piz98] Pizer, Arnold K.: Ramanujan graphs. In: *Computational perspectives on number theory (Chicago, IL, 1995)* Bd. 7. Providence, RI : Amer. Math. Soc., 1998

[Pos80] Postel, J.: *User Datagram Protocol*. RFC 768 (Internet Standard). Version: August 1980. `http://dx.doi.org/10.17487/RFC0768` (Internet Request for Comments)

[Pos81] Postel, J.: *Internet Protocol*. RFC 791 (Internet Standard). Version: September 1981. `http://dx.doi.org/10.17487/RFC0791` (Internet Request for Comments). Updated by RFCs 1349, 2474, 6864

[pqc] *SIDH Library.* : *SIDH Library*, `https://www.microsoft.com/en-us/research/project/sidh-library/`. – Last checked: 2021-01-17

[PST20] Paquin, Christian ; Stebila, Douglas ; Tamvada, Goutam: Benchmarking post-quantum cryptography in tls. In: *International Conference on Post-Quantum Cryptography* Springer, 2020, S. 72–91

[Raa] Raadt, Theo de: *Mitigations and other real security features*, `https://www.openbsd.org/papers/bsdtw.pdf`. – Last checked: 2021-01-17

[RPH$^+$15] Riste, Diego ; Poletto, Stefano ; Huang, M-Z ; Bruno, Alessandro ; Vesterinen, Visa ; Saira, O-P ; DiCarlo, Leonardo: Detecting bit-flip errors in a logical qubit using stabilizer measurements. In: *Nature communications* 6 (2015), Nr. 1, S. 1–6

[Sch07] Schneier, Bruce: *Applied cryptography: protocols, algorithms, and source code in C.* john wiley & sons, 2007

[SFG20] Steblia, Douglas ; Fluhrer, Scott ; Gueron, Shay: Hybrid key exchange in TLS 1.3 / Internet Engineering Task Force. Version: Oktober 2020. `https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-01`. Internet Engineering Task Force, Oktober 2020 (draft-ietf-tls-hybrid-design-01). – Internet-Draft. – Work in Progress

[Sho94] Shor, Peter W.: Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings 35th annual symposium on foundations of computer science* Ieee, 1994, S. 124–134

[sik] *SIKE – Supersingular Isogeny Key Encapsulation.* : *SIKE – Supersingular Isogeny Key Encapsulation*, `https://sike.org`. – Last checked: 2021-01-17

[Sil09] Silverman, Joseph H.: *The arithmetic of elliptic curves.* Bd. 106. Springer Science & Business Media, 2009

[SKP15] Stevens, Marc ; Karpman, Pierre ; Peyrin, Thomas: *Freestart collision for full SHA-1.* Cryptology ePrint Archive, Report 2015/967, 2015. – `https://eprint.iacr.org/2015/967`

[Smy14] Smyslov, V.: *Internet Key Exchange Protocol Version 2 (IKEv2) Message Fragmentation.* RFC 7383 (Proposed Standard). Version: November 2014. `http://dx.doi.org/10.17487/RFC7383` (Internet Request for Comments)

[Smy20] Smyslov, Valery: Intermediate Exchange in the IKEv2 Protocol / Internet Engineering Task Force. Version: September 2020. `https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-intermediate-05`. Internet Engineering Task Force, September 2020 (draft-ietf-ipsecme-ikev2-intermediate-05). – Internet-Draft. – Work in Progress

[SR13] Stevens, W. R. ; Rago, Stephen A.: *Advanced Programming in the UNIX Environment.* 2013. – ISBN 9780321637734

[SW15] Smyslov, V. ; Wouters, P.: *The NULL Authentication Method in the Internet*

Key Exchange Protocol Version 2 (IKEv2). RFC 7619 (Proposed Standard). Version: August 2015. `http://dx.doi.org/10.17487/RFC7619` (Internet Request for Comments)

[THS20] Tjhai, C. ; Heider, Tobias ; Smyslov, Valery: Beyond 64KB Limit of IKEv2 Payload / Internet Engineering Task Force. Version: Oktober 2020. `https://datatracker.ietf.org/doc/html/draft-tjhai-ikev2-beyond-64k-limit-00`. Internet Engineering Task Force, Oktober 2020 (draft-tjhai-ikev2-beyond-64k-limit-00). – Internet-Draft. – Work in Progress

[TTB$^+$20] Tjhai, C. ; Tomlinson, M. ; Bartlett, G. ; Fluhrer, Scott ; Geest, Daniel V. ; Garcia-Morchon, Oscar ; Smyslov, Valery: Multiple Key Exchanges in IKEv2 / Internet Engineering Task Force. Version: Juli 2020. `https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-multiple-ke-01`. Internet Engineering Task Force, Juli 2020 (draft-ietf-ipsecme-ikev2-multiple-ke-01). – Internet-Draft. – Work in Progress

[Unr15] Unruh, Dominique: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques* Springer, 2015, S. 755–784

[val] *Valgrind.* : *Valgrind*, `https://valgrind.org`. – Last checked: 2021-01-17

[Vél71] Vélu, Jacques: Isogénies entre courbes elliptiques. In: *CR Acad. Sci. Paris, Séries A* 273 (1971), S. 305–347

[YAJ$^+$17] Yoo, Youngho ; Azarderakhsh, Reza ; Jalali, Amir ; Jao, David ; Soukharev, Vladimir: A post-quantum digital signature scheme based on supersingular isogenies. In: *International Conference on Financial Cryptography and Data Security* Springer, 2017, S. 163–181