KU LEUVEN

FACULTY OF SCIENCE

New signature schemes based on UOV

with smaller public keys

Ward BEULLENS

Supervisor: Prof. B. Preneel
Co-supervisor: *Alan Szepieniec*

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Mathematics

Academic year 2016-2017

Preface

This thesis is submitted for the degree of Master of Science in Mathematics. The research described in this thesis was conducted during the academic year 2016-2017 at the COSIC research group at the department of electrical engineering (ESAT) of the KU Leuven. Conducting the research and compiling the results into this thesis has been a thrilling but arduous experience. Luckily, I could count on the help of a number of people whenever help was needed.

I would like to thank my supervisor Prof. B. Preneel and the researchers at the COSIC group who helped me, including W. Castryck and F. Vercauteren. I am particularly grateful to Alan Szepieniec, who co-supervised this thesis, for his help throughout the year and the many fruitful conversations we had that where indispensable for my research. Thanks also to my friends and family for their encouragement and support.

Part of the contents of this thesis was presented in the following publication:

- Alan Szepieniec, Ward Beullens, Bart Preneel. MQ signatures for PKI. In *Eighth International Conference on Post-Quantum Cryptography*. Springer, 2017.

A different paper was submitted to the Selected Areas in Cryptography (SAC) conference and is currently being reviewed:

- Ward Beullens, Bart Preneel. Field lifting for smaller UOV public keys. Submitted to *Selected Areas in Cryptography - SAC 2017*. Springer, 2017.

Abstract

Shor's quantum algorithm threatens to break nearly all public-key cryptography deployed today. Multivariate cryptography is one of the major candidates for providing fast and secure public-key cryptography in a post-quantum world. However, multivariate cryptosystems have prohibitively large public keys. This thesis addresses this issue by proposing two new variants of the Unbalanced Oil and Vinegar signature scheme. The first scheme realizes dramatically smaller public keys (15 times smaller) by lifting the UOV map to a much larger extension field, where system solving is harder. The second scheme uses techniques from hash based cryptography in order to obtain tiny public keys, at the cost of larger signatures. The security of the second scheme reduces to the security of UOV in the Quantum Random Oracle Model. Software implementations of both signature schemes are made in ANSI C. The aim is to submit these signature schemes to the NIST Post-Quantum Crypto project.

Contents

Chapter 1

Introduction

1.1 What are signature schemes?

In today's world the Digital Revolution has enabled billions of people to communicate with each other across the globe at near-light speed. This development has increased quality of life, created millions of jobs and massive amounts of wealth. However, the new technologies also introduce new vulnerabilities. Criminals trick people into revealing their passwords or into installing malicious software in order to plunder their bank accounts or monitor their communications. How to know who you can trust on the world wide web? Digital signatures can provide a way to be certain that whoever you are talking to is really who they claim to be.

To understand a digital signature it is useful to make the comparison with its ancestor: the wax seal. When a medieval lord or lady received a letter from the king, he or she could check the wax seal on the envelope to be sure that the letter was indeed signed with the royal stamp. Under the assumption that it is impossible to replicate a stamp and that the royal stamp was still in the possession of the king this would mean that the letter was really sent by the king. Moreover, if the seal was not broken he or she would be sure that the letter had not been opened before, so the contents of the letter has not been changed. Modern digital signature schemes operate on the same principles. Anyone who wants to sign a document needs a digital "stamp" which he has to keep secret, this "stamp" is called the secret key. In medieval times, if someone wanted to check the validity of a seal he or she has to know what the seal is supposed to look like. In the modern version a verifier also needs a piece of information in order to be able to verify a signature, this piece of information is called the public key since it is public knowledge.

A digital signature scheme consists of three algorithms.

- The *key generation algorithm* randomly generates a secret key sk and a corresponding public key pk.
- The *signature generation algorithm* uses the secret key to produce a valid signature s for any document d.
- Given d,s and pk the *validation algorithm* checks if s is a valid signature for the document d.

A good signature scheme guarantees the following properties for signatures it decides are valid:

- Authenticity : The document was signed by the user who knows the secret key.
- Integrity : The document has not been tampered with by anyone after it was signed.
- Non-repudiation of origin : Since the signer of the document is the only one in possession of the secret key he can not deny having signed the document at some later time.

Today, signature schemes are used for a wide variety of applications. For example, if you download or update an app via the Google Play Store or the Apple app store the downloaded data is verified with a digital signature. This way you can be sure that the app you download is really in the same state as the version of the app that was signed by the developer. This means no malware was added to the app after the app was signed. However it is still possible that it was already there when the signing happened of course. (Maybe even intentionally put there by the designer of the app.) Most software updates done today are verified with a digital signature in this way. Moreover, the security of all secure communication protocols rely on some kind of authentication. After all, it is useless to be able to communicate securely if you can't be certain who you are communicating with. This authentication can be provided with a digital signature. Also, in some countries, including the EU countries, legal contracts can be signed with a digital signature instead of a physical signature, which can be very convenient when doing business with partners across the globe.

1.2 Why do we need new signature schemes?

Since digital signature schemes are so widely used there is a constant search for digital signature schemes that require less memory and fewer computational resources. This is particularly important in applications that run on smart cards or applications such as pacemakers where the energy supply is very limited but where the stakes are very high. Another challenge for cryptographers comes from the advances in quantum computing.

The idea of a quantum computer, a computer that relies on quantum mechanical phenomena to perform its calculations, is often attributed to the famous Nobel prize-winning physicist Richard Feynman. He noticed that some quantum mechanical systems appear to be extremely difficult to simulate on a classical computer, so he proposed that it might be possible to simulate these systems with another quantum mechanical system, a quantum simulator [20]. Later, people realized that it might be possible to use this theoretical device to calculate other things that are hard to compute for a classical computer. The idea of the quantum computer was born. Since then a lot of effort has been put in developing this idea and many breakthroughs have been made. On a theoretical level quantum algorithms have been found that are exponentially faster than the best known classical algorithm that solve the same problem. The most famous example of this exponential speed-up is Shor's algorithm [31], which is able to efficiently factor large integers and compute discrete logarithms in any group. On a practical level physicists and engineers

have succeeded in building small quantum computers and run small algorithms on them. It remains to be seen if it is possible to scale up those quantum computers to useful sizes and if so, if this will be possible within a decade or within centuries.

The security of most digital signature schemes in use today rely on the hardness of computing discrete logarithms, factorizing large semiprimes and related problems. These are all problems that can be solved efficiently once large enough quantum computers are built. Therefore, the advances in quantum computing pose a serious threat to the majority of the digital signature schemes in use today as well as many other cryptographic systems such as public key encryption, key agreement protocols and zero-knowledge identification schemes. This is why cryptographers are looking for quantum resistant algorithms. There are a number of promising branches of so-called post-quantum cryptography, one of which is based on the hardness of finding solutions to systems of multivariate quadratic equations over a finite field $\mathbb{F}_q$. This is known as the MQ-problem. A short analysis of the hardness of this problem will be given in Sect.3.2.1. It is this branch, known as MQ-crypto that this thesis will focus on.

Very recently NIST, the American National Institute for Standards and Technology, has started a search for post-quantum cryptographic algorithms with the goal to standardize a selection of them [1]. The institute has issued a call for proposals. Submissions will be analyzed for security and efficiency. Hopefully, at the end of this process, which is projected to last six to eight years, some post-quantum algorithms will be standardized. Previous NIST standardization projects have resulted in the Advanced Encryption Standard AES and the SHA-3 hashing algorithm. These algorithms are now used extensively by the US government and extremely many other users. It is noteworthy that both the AES and the SHA-3 competitions were won by Belgian submissions.

1.3 Contributions of this thesis

In this thesis we develop two new signature schemes with the goal of submitting them to the NIST post-quantum crypto standardization project. In the first three chapters we introduce the reader to some preliminaries and we give an introduction to MQ signature schemes, and the UOV signatures scheme in particular. The new signature schemes we develop are based on the well studied signature scheme UOV, which has withstood attacks since its formulation in 1999 with some minor parameter changes. The largest problem with UOV schemes is that they have very large public and secret keys. An important step in mitigating this problem was taken by Petzoldt et al. [28]. They reduced the public key size by a factor 8 without affecting the security of the scheme. We propose two different signature schemes that reduce the key sizes even further, each in its own way. Both signature schemes work over large finite fields.

In chapter 4 we develop a first new signature scheme that dramatically reduces the public key sizes by using public keys and secret keys over $\mathbb{F}_2$. The key idea is to lift the keys to a large extension field $\mathbb{F}_{2^r}$ where solving polynomial systems is more difficult. The resulting signature scheme is very competitive with the best post quantum signature schemes in terms of signature size, key sizes and efficiency.

In chapter 5 we develop a second signature scheme which is a modified version of UOV which relies on Merkle trees, a technique used in hash-based cryptography. With this modification we obtain tiny public keys at the cost of larger signatures. The combined size of the public key and the signature is roughly a factor 2.5 smaller than the best MQ signature schemes (UOVrand and RainbowLRS2 [28]). We are able to prove rigorously that this scheme is at least as secure as the original UOV scheme in the quantum random oracle model. Also, it is possible to apply the modification we used on the UOV scheme on other MQ signature schemes to significantly reduce the combined size of a signature and a public key.

In chapter 6 we adapt the signature scheme of chapter 5 to reduce the size of the signatures even further. The adapted signature scheme is no longer provably as secure as the original UOV signature scheme. Instead, its security depends on the hardness of a new problem, the Approximate MQ problem, which is a natural generalization of the MQ problem. This adapted version of the idea of chapter 5 also be applies to other MQ signature schemes.

Chapter 2

Preliminaries

2.1 Cryptographic hash functions

The cryptographic hash function is an extremely important tool for many applications in cryptography. A hash function is a function that assigns to any message of arbitrary length a hash value (also referred to as a message digest) of fixed length. A hash function $\mathcal{H} : X^* \to X^n$ is called cryptographically secure if the following properties are satisfied:

- **One-wayness** (also known as preimage resistance) : Given a hash value $h \in X^n$ it is hard to find a message $m \in X^*$ such that $\mathcal{H}(m) = h$.
- **Second preimage resistance** : Given a message m it is hard to find a different message m' such that $\mathcal{H}(m) = \mathcal{H}(m')$.
- **Collision resistance** : It is hard to find two messages $m \neq m'$ such that $\mathcal{H}(m) = \mathcal{H}(m')$.

These rather informal properties can be formalized using games. A game is played by an adversary. We say that a hash function satisfies one of the properties if no adversary can win the associated game with a non-negligible probability. For example, the game associated to the second preimage resistance is defined in Alg. 2.1. A hash function $\mathcal{H}$ (or rather a family of hash functions indexed by a security parameter n) is second preimage resistant if for any efficient adversary A (i.e. the running time of A is polynomial in n), the probability that A wins the SPR game is negligible, meaning that this probability decreases faster than the inverse of any polynomial. More precisely, for any polynomial $p(n) \in \mathbb{R}[n]$ there exists an N such that for all $n > N$

$$Pr[A \text{ wins SPR}(\mathcal{H},\text{A})] < 1/p(n)\,.$$

Note that the hash function $\mathcal{H}$ is indexed by a security parameter n, although in practice this is almost never mentioned explicitly.

It should be understood that the hardness of a problem depends on the model of computation that is being used. For example, some problems (or rather families of problems parametrized by a security parameter), such as integer factorization, are believed to be hard for classical computers, but can be efficiently solved by a quantum computer. Therefore it is possible that a certain hash function is secure against classical adversaries, but not against quantum adversaries.

Algorithm SPR

input: $\mathcal{H} : D \to R$ — A hash function from domain D to range R.
A — An adversary

output: **Win** / **Lose** — Whether A wins or loses the game

1: $m \xleftarrow{\$} D$ ▷ choose m uniformly random from D
2: $h \leftarrow \mathcal{H}(m)$
3: $m' \leftarrow A(m, h)$
4: **if** $m' \neq m$ and $\mathcal{H}(m') = h$ **then**
5: | **return Win**
6: **else**
7: | **return Lose**
8: **end if**

Alg. 2.1: The game associated the the second preimage resistance property.

Currently, there are no functions which are proven to be cryptographically secure hash functions. It is not even known whether one-way functions exist. The existence of a one-way function would prove that P≠NP, thereby solving the famous problem in Computer Science, which is also a Clay Institute Millennium Prize Problem [13]. However, there are many practical functions for which no known efficient algorithms find (second) preimages or collisions. One such function is SHA-3 (secure hashing algorithm 3). This algorithm is was standardized by NIST, the US National Institute of Standards and Technology, after it won the NIST Hash function competition.

2.1.1 Random Oracle Model

A function $\mathcal{H} : D \to R$ which is drawn uniformly from the set of all functions from D to R would be an ideal hash function. Unfortunately such a function is impossible to implement. Nevertheless, we use the concept of a random oracle that 'computes' a randomly chosen function. An algorithm can make use of a random oracle by sending it inputs and receiving the results. A random oracle is one-way, second preimage resistant and collision resist, since the best way to find a preimage (or second preimage or collision) is just to query the oracle at a large number of points and hope that the oracle returns the right value. This has a very small success probability if the range of the random oracle is large. Random oracles even have much stronger properties, that are of practical use, and that hash functions don't necessarily have. For example, evaluations of a random oracle at different values are independent of each other. Even though random oracles cannot be implemented they are frequently used in the design of cryptographic schemes. It is often possible to prove that some cryptographic scheme that uses a hash function is secure if the hash function that is used were replaced by a random oracle. If this is the case the scheme is called secure in the random oracle model. Strictly speaking, this does not prove anything about the security of the actual scheme and indeed, there are schemes that are proven secure in the random oracle model, but are insecure when a real hash function is used instead of a random oracle [11]. Still, if a scheme is proven secure in the random oracle model this is considered to be evidence for the security of the scheme.

2.1.2 Quantum Random Oracle Model

A random oracle can only be evaluated at one input at once. That is, a user queries the oracle at an input $M \in D$, and the oracle returns $f(M) \in R$, where f is a fixed randomly chosen function $D \to R$. In the context of quantum computation this is not a realistic idealization of a hash function anymore because on a quantum computer a hash function $\mathcal{H}$ can be evaluated in superposition. This means that given a superposition $\sum_i c_i|m_i\rangle$ the evaluation of the hash function at this superposition $\sum_i c_i|m_i\rangle|\mathcal{H}(m_i)\rangle$ can be calculated. Like the classical random oracle, a quantum random oracle is an oracle that computes a randomly chosen function. The difference is that the quantum oracle can answers queries that are superpositions. That is, when queried at a superposition $\sum_i c_i|m_i\rangle$ the quantum random oracle returns $\sum_i c_i|m_i\rangle|f(m_i)\rangle$, with f a fixed randomly chosen function. If a cryptographic scheme is proven to be secure when the hash functions are replaced by quantum random oracles the scheme is called secure in the quantum random oracle model.

Since a quantum random oracle is more powerful than a classical random oracle it could be that a cryptographic scheme that is secure in the random oracle model is not secure in the quantum random oracle model. Boneh et al. show that this indeed occurs and that security in the quantum random oracle model is strictly stronger than security in the random oracle model [8].

2.2 Trapdoor functions

A trapdoor function is a collection of one-way functions $f_k : D_k \to R_k$ parametrized by $k \in \mathcal{K}$, where $\mathcal{K}$ is the space of public keys. For every public key k, there is a corresponding trapdoor t_k that allows the function f_k to be inverted efficiently. More precisely there exists the following efficient algorithms :

- **Key generation algorithm** : Produces a random key k and corresponding trapdoor t_k.
- **Evaluation algorithm** : Given a key k and a value x it computes the evaluation $f_k(x)$.
- **Inversion algorithm** : Given a key k, the trapdoor t_k and an evaluation e for which there exists a value x such that $f_k(x) = e$, the algorithm computes a value y such that $f_k(y) = e$.

Example. *An important example of a trapdoor function is the RSA-function. Given a public key of the form $k = (e, n)$, where n is the product of two large primes and e coprime to $\phi(n)$, the function is defined as*

$$f_k(x) = x^e \mod n\,.$$

The trapdoor is given by a number d such that $de \equiv 1 \mod \phi(n)$. The inverse is then

$$f_k^{-1}(x) = x^d \mod n\,.$$

Note that similar to the concept of hash functions, the concept of trapdoor functions depends of the model of computation that is being used. It is shown by Peter Shor that there are efficient quantum algorithms that invert the RSA-function, so it can no longer be called a trapdoor function in the context of quantum computers.

2.3 Hash-and-Sign signature schemes

A hash function and a trapdoor function can be combined into a signature scheme. Suppose we have a hash function $\mathcal{H} : \{0,1\}^* \to R$ which maps bitstrings to a range R and a trapdoor function from a signature space S to the same range $f_k : S \to R$ for all keys $k \in \mathcal{K}$. Then we can formulate the following algorithms.

- **Key generation algorithm** : Use the key generation algorithm for the trapdoor function to get (k, t_k). The public key of the signature scheme is k, the private key is t_k.
- **Signature generation algorithms** : Given a message $M \in \{0,1\}^*$ calculate $h = \mathcal{H}(M)$. Use the trapdoor t_k to calculate the signature $s = f_k^{-1}(h)$.
- **Signature verification algorithm** : Given a message M and a signature s, calculate $h = \mathcal{H}(M)$ and $h' = f_k(s)$. The signature is valid if and only if $h = h'$.

This is a general framework for constructing signature schemes known as the hash-and-sign paradigm that is used by many signature schemes, including the RSA signature scheme and the UOV signature scheme.

The reader might wonder why the hashing is required. It seems easier to use the trapdoor function to sign the message directly. That is, a valid signature for a message M would be an s such that $f_k(s) = M$. This has two problems, which are solved using the hash function. The first problem is that with this approach we can only sign messages which lie in the range of the trapdoor function, whereas in practice we want to be able to sign any message. The hash function solves this problem by converting bitstrings to a hash value of fixed length. A second problem is that it is trivial to produce valid message-signature pairs by choosing a random signature s and evaluating $M = f_k(s)$ to find a corresponding message. In the hash-and-sign paradigm this is not possible. If an attacker takes a random s, he can calculate $h = f_k(s)$, but in order to make this into a valid message-signature pair the attacker should find a message M such that $\mathcal{H}(M) = h$, which is hard for hash functions.

Nearly all digital signature schemes follow the hash-and-sign paradigm, including the popular DSA (Digital Signature Algorithm) algorithm, most MQ signature schemes and the first of the new signature schemes that are proposed in this thesis.

2.4 Digital signature forgery attacks

In this section we explain different types of attacks against digital signature schemes. Each type of attack is formalized by a game G, played by an adversary A. We say that a signature scheme is secure against attacks of some type X, if for every efficient adversary A, the probability that A wins the game associated to X is negligible. The different types of attacks specify a goal of the attack and the attack model.

Different goals of an attack:

1. Universal Forgery : Given an arbitrary message M, the goal of the attack is to produce a valid signature for M.

2. Selective Forgery : The adversary himself chooses a message M before the start of the attack, then the goal of the attack is to forge a valid signature for M

3. Existential Forgery : The goal of the attack is to produce any valid message-signature pair.

The difference between a selective and an existential forgery is a bit subtle. In a selective forgery the adversary has to select a message M before the start of his attack. An existential forgery is potentially easier, because the adversary can decide which message to forge a signature for during the execution of the attack. These goals are ordered from harder to easier. It is clear that if an attacker can preform a universal forgery, he can also do selective forgery, and if an attacker can do a selective forgery, he can also do an existential forgery.

Different attack models :

1. Key-only attack : The adversary only knows the public key.

2. Known message attack : The adversary knows the public key and a list of valid signature-message pairs. (The message the adversary needs to forge a signature for is not in the list)

3. Chosen message attack : The adversary knows the public key and he can ask for signatures for any message he chooses. (Except for the message the adversary needs to forge a signature for.)

The attack models are ordered from more difficult to easier. If an adversary can do a Key-only attack he can also do known message attacks. If an adversary can do a known message attack, he can also do chosen message attacks.

The strongest form of security is security against the easiest goal (Existential forgery) with the easiest attack model (Chosen message attack). This is known as Existential Unforgeability under Chosen Message Attack (EUF-CMA). Other types of attacks have similar abbreviations, e.g. UUF-KOA. We describe the game which formalizes the EUF-CMA attack in Alg. 2.2.

Remark. *In a chosen message attack, a quantum adversary is not allowed to ask for messages in superposition to be signed. This is so because the CMA attack models a situation where an attacker is able to trick a victim into signing some documents. It is not realistic that an attacker can trick a victim into signing a document in superposition.*

Algorithm EUF-CMA Game

input: (GenerateKeys,Sign,Verify) — A signature scheme
A — An adversary

output: **Win** / **Lose** — Whether A wins or loses the game

```
(pk, sk) ← GenerateKeys()
Communicate pk to A
Messages ← empty list
while A queries a signature for the message M do
    Messages.append(M)
    s ←Sign(M, sk)
    communicate s to A
end while
A returns a signature message pair (s, M)
if M ∈ Messages then
    return Lose
else if Verify(s, M) then
    return Win
else
    return Lose
end if
```

Alg. 2.2: The game associated to the EUF-CMA attack

2.5 Insecurity functions

The insecurity function $\text{InSec}^p(s; t, q)$ denotes the maximum probability with which an adversary that runs in time t and that makes at most q queries wins the game associated to property p of the primitive s. For example, $\text{InSec}^{\text{EUF-CMA}}(\text{OUV}, t, q)$ is the maximal probability with which any adversary that makes q queries and that runs in time t can win the EUF-CMA game against the Unbalanced Oil and vinegar signature scheme. A different example is $\text{InSec}^{SPR}(\mathcal{H}; t, q)$ which is the maximal probability with which any adversary which runs in time t and makes q queries to $\mathcal{H}$ can forge a second preimage. If we omit the time variable t from the function, we mean the maximal probability with which an adversary with an unbounded amount of time can win the game associated to property p.

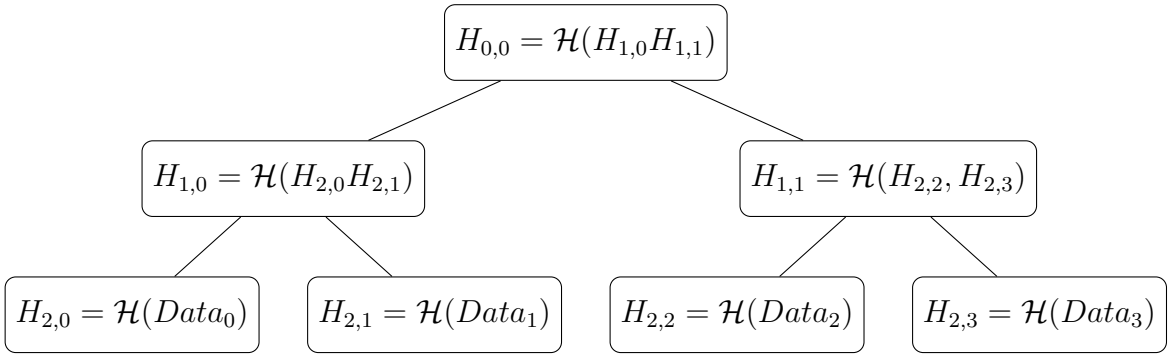

Figure 2.1: Binary Merkle tree of height 2.

2.6 Merkle trees

A Merkle tree or hash tree is a tree structure where each internal node (i.e. non-leaf node) contains the hash value of its children and each leaf node is the hash of some piece of data. This way, the root of the Merkle tree can authenticate a large number of data values.

One use of Merkle trees is the authentication of large amounts of data. For example, consider the situation where a movie or some piece of software is downloaded via a peer-to-peer network such as BitTorrent. In such a case the data is divided in small packages that are downloaded separately, and each package can be provided by a different computer in the network. Not all computers in the network can be trusted, some may try to send corrupted data. Therefore each package has to be verified. To do this, a user first acquires a Merkle root from a trusted party, in the BitTorrent case the Merkle tree root is included in the torrent file, which you get from a "trusted" party such as PirateBay. Then each downloaded package contains a path of the Merkle tree that validate the data in the package. For example, consider the simple case where the data is divided in four packages, labeled from $Data_1$ up to $Data_4$ and a binary tree of height 2 is used. The tree is displayed in Fig. 2.1. Suppose we download the second package, the download includes $Data_2$ as well as the values $H_{2,0}$ and $H_{1,1}$. To validate $Data_2$ we consecutively calculate $H_{2,1} = \mathcal{H}(Data_1), H_{1,0} = \mathcal{H}(H_{2,0}H_{2,1})$ and $H_{0,0} = \mathcal{H}(H_{1,0}H_{1,1})$. Then the root $H_{0,0}$ is compared to the root value we have acquired from the trusted party. If the two roots match we can be sure that the $Data_1$ has not been tampered with, because due to the nature of hash functions, it is very hard for an opponent to produce a Merkle path that connects the corrupted data with the true Merkle root.

In general, a binary Merkle of height h can be used to validate up to 2^h pieces of data. To validate a data piece h extra hash values have to be included with the data. Now we describe the algorithms associated to Merkle trees more formally. To this end, we denote the hash value of the i-th node (starting from 0) at depth d by $H_{d,i}$. The children of the i-th node at depth d are the $2i$-th and $2i+1$-th nodes at depth $d+1$, so with this labeling we have

$$H_{d,i} = \begin{cases} \mathcal{H}(H_{d+1,2i}H_{d+1,2i+1}) & \text{if } d < h \\ \mathcal{H}(Data_i) & \text{if } d = h \end{cases}.$$

This relation translates readily into a recursive algorithm for calculating the hash value of an arbitrary node of the Merkle tree. This algorithm is presented in Alg. 2.3. In order to verify a piece of data we have to include a list of hash values that allow a verifier to verify a path from the piece of data to the root of the Merkle tree. Starting from the bottom of the tree, if we are in a left child of the a node we have to include the hash value of the right node and vice versa. This means that if we want to verify $H_{d,i}$ we need to include $H_{d,i\oplus 1}$, where $\oplus$ stands for the bitwise XOR operation. Then, in order to verify the next value $H_{d-1,\lfloor i/2 \rfloor}$ we need to include $H_{d-1,\lfloor i/2 \rfloor \oplus 1}$. Continuing like this we travel up the tree until we have a complete list of hash values. This algorithm is described more formally in Alg. 2.5. To verify the i-th piece of data we need to calculate all the values $H_{d,i}, H_{d-1,\lfloor i/2 \rfloor}, \ldots$ all the way up to $H_{0,0}$. And check whether this last value is equal to root of the Merkle tree. This is described in Alg. 2.6

Algorithm CalculateNode

input: $\text{Data}_0, \cdots, \text{Data}_{2^h-1}$ — The data values of the Merkle tree
d — The depth of the node
i — The node whose hash value to calculate
output: $H_{d,i}$ — The hash value of the i-th node at depth d

1: **if** $d = h$ **then**
2: | **return** $\mathcal{H}(\text{Data}_i)$
3: **else**
4: | $H_{d+1,2i} \leftarrow \text{CalculateNode}(\text{Data}_0, \cdots, \text{Data}_{2^h-1}, d+1, 2i)$
5: | $H_{d+1,2i+1} \leftarrow \text{CalculateNode}(\text{Data}_0, \cdots, \text{Data}_{2^h-1}, d+1, 2i+1)$
6: | **return** $\mathcal{H}(H_{d+1,2i} H_{d+1,2i+1})$
7: **end if**

Alg. 2.3: Algorithm that calculates the hash value of a node of a Merkle Tree.

Algorithm CalculateMerkleRoot

input: $\text{Data}_0, \cdots, \text{Data}_{2^h-1}$ — The data values of the Merkle tree

output: The root of the Merkle tree

1: **return** $\text{CalculateNode}(\text{Data}_0, \cdots, \text{Data}_{2^h-1}, 0, 0)$

Alg. 2.4: Algorithm that calculates the root of a Merkle tree

Algorithm OpenMerklePath

input: $\text{Data}_0, \cdots, \text{Data}_{2^h-1}$ — The data values of the Merkle tree
i — The index of the leaf of the Merkle tree to open

output: A list of Hash values required to validate a leaf of the Merkle tree

```
1: List ← empty list of length h
2: for depth from h to 1 do
3: |  i ← i ⊕ 1                                   ▷ Toggles last bit of i
4: |  List[depth] ← CalculateNode(Data_0, ··· , Data_{2^h−1},depth , node)
5: |  i ← ⌊i/2⌋
6: end for
7: return List
```

Alg. 2.5: Algorithm for calculating the list of hash values required to validate a leaf.

Algorithm VerifyMerklePath

input: i — The index of the leaf to verify a path for
Data_i — The leaf to verify a path for
List — A list of Hash Values
Root — Root of the Merkle tree

output: **True** if the Merkle path is valid, **False** otherwise

```
1: CurrentHash ← H(Data_i)
2: for depth from h to 1 do
3: |  if i is even then
4: |  |  CurrentHash ← H(CurrentHash List[depth])
5: |  else
6: |  |  CurrentHash ← H(List[depth] CurrentHash)
7: |  end if
8: |  i ← ⌊i/2⌋
9: end for
10: return CurrentHash = Root
```

Alg. 2.6: Algorithm that verifies the validity of a Merkle path

Chapter 3

The UOV signature scheme

The UOV or Unbalanced Oil and Vinegar digital signature scheme is a modified version of the original (balanced) oil and vinegar signature scheme that was proposed by Patarin in 1997 [27]. UOV belongs to the family of MQ-digital signature schemes, this means that the security of UOV relies on the hardness of finding solutions to certain quadratic systems. Since the first proposal of the oil and vinegar scheme in 1997 much more has been learned about the complexity of solving quadratic systems. These new insights have revealed that some of the parameter sets that were proposed originally are not secure. However, with the right set of parameters UOV remains one of the most promising MQ signature schemes. In this chapter we will first explain how MQ signature schemes work in general. Then we will explain the specifics of the UOV signature scheme and the known attacks against it.

3.1 MQ signature schemes

Roughly speaking, MQ signatures are an instantiation of the hash-and-sign paradigm which use a trapdoor functions which are multivariate quadratic polynomial maps. At the heart of any MQ signature scheme there are two quadratic polynomial maps $\mathcal{P}, \mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ that are isomorphic in the sense that there exist linear isomorphisms $\mathcal{S} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ and $\mathcal{T} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ such that

$$\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} .$$

The polynomial map $\mathcal{F}$ is chosen in such a way that it is possible to find a solution s to the system $F(s) = x$ for any $x \in \mathbb{F}_q^m$ in a reasonable amount of time. During the key generation phase the polynomial maps $\mathcal{P}, \mathcal{F}$ and the linear maps $\mathcal{T}, \mathcal{S}$ are generated. The polynomial map $\mathcal{P}$ is the public key. The factorization $\mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ is the secret key. Signing a message M happens as follows: first a message digest $\mathcal{H}(M) = h \in \mathbb{F}_q^m$ is calculated using some hash function $\mathcal{H}$. Then we solve the system $\mathcal{P}(s) = h$ to find a valid signature s. This can be done using the factorization $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ since the linear maps $\mathcal{T}$ and $\mathcal{S}$ are easily inverted and finding an inverse of $\mathcal{F}$ is easy by assumption. In order for the scheme to be secure it should be the case that only someone who knows the factorization of $\mathcal{P}$, can invert $\mathcal{P}$. Moreover it should be so that given $\mathcal{P}$ it is unfeasible to find the factorization $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$. (Or any other factorization that can be inverted easily.) If that is the case then signing a message is only possible if the secret key is known.

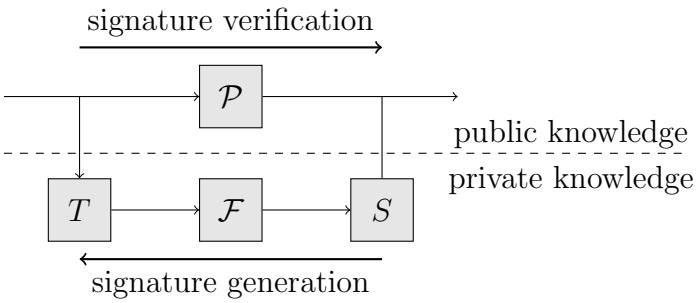

Figure 3.1: Schematic representation of multivariate quadratic cryptosystems.

Now we describe a MQ signature scheme more formally. Like any digital signature scheme an MQ signature scheme consists of 3 algorithms. The key generation algorithm, signature generation and signature validation algorithms are shown in Alg. 3.1, Alg. 3.2 and Alg. 3.3 respectively.

Algorithm MQGenerateKeys

input: Random bits to choose $\mathcal{F}, \mathcal{S}$ and $\mathcal{T}$

output: A key pair $(\mathcal{P}, (\mathcal{S}, \mathcal{F}, \mathcal{T}))$

1: $\mathcal{F} \leftarrow$ chosen randomly from $\mathfrak{F}$, a family of polynomial quadratic functions that can be inverted easily.
2: $\mathcal{S} \leftarrow$ A random invertible affine map ${\mathbb{F}_q}^m \rightarrow {\mathbb{F}_q}^m$
3: $\mathcal{T} \leftarrow$ A random invertible affine map ${\mathbb{F}_q}^n \rightarrow {\mathbb{F}_q}^n$
4: $\mathcal{P} \leftarrow \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$
5: **return** $(\mathcal{P}, (\mathcal{S}, \mathcal{F}, \mathcal{T}))$

Alg. 3.1: The generic MQ Signature generation Algorithm

Algorithm MQSign

input: $(\mathcal{S}, \mathcal{T}, \mathcal{F})$ — The public key
M — A message to sign

output: $s \in \mathbb{F}_q^n$ — A signature for the message M

1: $h \leftarrow \mathcal{H}(M)$
2: $h' \leftarrow \mathcal{S}^{-1}(h)$
3: $s' \leftarrow$ a solution for $\mathcal{F}(s') = h'$
4: $s \leftarrow \mathcal{T}^{-1}(s')$
5: **return** s

Alg. 3.2: The generic MQ signature generation algorithm

Algorithm MQVerify

input: $\mathcal{P}$ — A public Key
M — A message
s — A candidate-signature

output: **True** if s is a valid signature for M, **False** otherwise

1: $h \leftarrow \mathcal{H}(M)$
2: $h' \leftarrow \mathcal{P}(s)$
3: **if** $h = h'$ **then**
4: | **return True**
5: **else**
6: | **return False**
7: **end if**

Alg. 3.3: The generic MQ signature verification algorithm

3.2 MQ-Problem and IP-Problem

The security of an MQ signature scheme relies on the hardness of two problems, the MQ-problem and the IP-problem. We give a brief discussion of these problems here.

3.2.1 MQ-Problem

It should be difficult for an attacker to solve the system $\mathcal{P}(s) = h$ for any given h, otherwise they would be able to sign any message. This problem is known as the MQ-problem, which stands for Multivariate Quadratic.

MQ Problem. Given a quadratic polynomial map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ over a finite field $\mathbb{F}_q$, find $\mathbf{x} \in \mathbb{F}_q^n$ that satisfies $\mathcal{P}(\mathbf{x}) = \mathbf{0}$.

It is known that the MQ problem is NP-hard [26]. Therefore it is unlikely that there are polynomial time algorithm that solve the hardest instances of the MQ problem. The problem is also believed to be hard on average in the case $n \approx m$. Only exponential time algorithms are known to solve random instances of the problem for these parameters.

Systems with $n = m$ are called determined systems; these are the most difficult systems to solve. When $n < m$ a system is called overdetermined, and when $n > m$ the system is called underdetermined. Finding a solution for an underdetermined system with $n = \alpha m$ can be reduced to finding a solution of a determined system with only $m+1-\lfloor\alpha\rfloor$ equations [34]. This means that as a system becomes more underdetermined it becomes easier to solve. This fact will become important in the security analysis of UOV.

3.2.2 Classical algorithms

The best known classical algorithms to solve the MQ-problem for generic determined systems over finite fields use the hybrid approach [6, 7], which combines an exhaustive search with Gröbner basis computations. In this approach k variables are fixed to random values and the remaining $n-k$ variables are found with a Gröbner basis algorithm such as F_4, F_5 or XL. If no assignment to the remaining $n-k$ variables exists that solves the system, the procedure starts again with a different guess for the first k variables. On average, we require roughly q^k Gröbner basis computations until a solution is found. As a result, the optimal value of k decreases as q increases. To estimate the complexity of Gröbner basis algorithms we need the concept of degree of regularity. Several non-equivalent definitions of the degree of regularity are in circulation. We will use the definition introduced by Bardet in her PhD thesis [2].

Definition. *For a polynomial system $f_1, \cdots, f_m$ with finitely many solutions the* **degree of regularity** *d_{reg} with respect to some monomial order $\prec$ is defined to be the smallest integer such that all monomials of total degree d_{reg} are the leading monomial of some polynomial in $\langle f_1, \cdots, f_m \rangle$.*

The complexity of the F_5 algorithm is given by

$$C_{F_5}(n, d_{reg}) = O\left(\binom{n + d_{reg}}{d_{reg}}^{\omega}\right),$$

where $2 \leq \omega < 3$ is the constant in the complexity of matrix multiplication. Therefore the complexity of the hybrid approach is

$$C_{\text{Hybrid}F_5(n,d_{reg},k)} = O\left(q^k \binom{n - k + d_{reg}(k)}{d_{reg}(k)}^{\omega}\right), \tag{3.1}$$

where $d_{reg}(k)$ stand for the degree of regularity of the system after fixing the values of k variables.
Determining the degree of regularity for a specific polynomial system is difficult, but for a certain class of systems, called semi-regular systems, it was shown by Bardet that the degree of regularity can be deduced from the number m of equations and the number n of variables [2]. For semi-regular systems the degree of regularity is the degree of the first term in the power series of

$$S_{m,n}(x) = \frac{(1 - x^2)^m}{(1 - x)^n}$$

with a non-positive coefficient. This gives a practical method to calculate the degree of regularity of any semi-regular system. Empirically, polynomial systems that are randomly chosen have a very large probability of being semi-regular and it is conjectured that most systems are semi-regular systems. For the definition and the theory of semi-regular systems we refer to chapter 3 of the PhD thesis of Bardet [2].

For completeness, we mention that for small dense polynomial systems over $\mathbb{F}_2$ the most efficient way of finding solutions is by an exhaustive search [9]. Using Grey codes it is possible to do an exhaustive search for the solutions of a polynomial system of m equations in n variables with approximately $\log_2(n)2^{n+2}$ bit operations.

3.2.3 Quantum algorithms

Currently, there are no specialized quantum algorithms that solve polynomial system over finite fields. Grover search can be used in a brute force search for solutions. This would yield a solution in $O(q^{m/2})$, which is slower than classical algorithms that use the hybrid approach. (An exception being the case $q = 2$, where the best classical algorithms have a complexity $O(2^{0.79n})$ [3].) Applying Grover search naively to the MQ problem is not that successful, but Grover's algorithm can be used to speed up the brute force part of the hybrid approach, giving a quadratic speedup for this part of the attack. The new complexity would be

$$C_{\text{Hybrid}F_5(n,d_{reg},k)} = O\left(q^{k/2} \binom{n - k + d_{reg}(k)}{d_{reg}(k)}^{\omega}\right), \tag{3.2}$$

where the difference with (3.1) is that we have the factor $q^{k/2}$ instead of q^k. However it should be noted that this approach requires running $q^{k/2}$ Gröbner basis computations sequentially on a quantum computer. This would be an incredible feat because even for

moderately sized polynomial systems this would require gigabytes worth of qubits and days of computation. Also, note that the gains of parallelizing Grover search grow only with the square root of the number of independent computers used, instead of a linear growth for the classical brute force search [36]. Nevertheless, in the security analysis of the signature schemes proposed in this thesis we will be cautious and assume that these kinds of attacks on the MQ problem are possible and we will make our parameter choices accordingly. This has the additional benefit of providing a thick safety margin against classical attacks.

3.2.4 IP-Problem

In order for an MQ signature scheme to be secure it should be infeasible for an attacker to factorize $\mathcal{P}$ into $\mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$. If an attacker is able to do so he has recovered the secret key and he or she can then sign any message in exactly the same way a legitimate owner of the secret key would. For some MQ cryptographic schemes the polynomial map $\mathcal{F}$ is the same for every key pair and hence publicly known. These systems include Square [12] and ℓIC [15]. For these systems the problem is known as the IP-problem.

IP-Problem. (for Isomorphism of Polynomials) Given two polynomial systems $\mathcal{P}$ and $\mathcal{F}$ that are guaranteed to be isomorphic, find 2 affine transformations $\mathcal{S}, \mathcal{T}$ such that $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$.

A graph-theoretic algorithm is known that solves this problem over finite fields of q elements in $O(q^{n/2})$ where n is the number of variables [10]. However for specific (non-random) choices of $\mathcal{F}$ the problem can be much easier.

Not all MQ cryptosystems uniquely specify the system $\mathcal{F}$. For example, during the key generation phase of the UOV signature scheme, which we will describe later in this chapter, $\mathcal{F}$ is chosen randomly from some large family of polynomial systems that are easily inverted. For those kinds of MQ schemes we need a more general form of the IP-Problem.

EIP-Problem. (for Extended Isomorphism of Polynomials) Given a polynomial system $\mathcal{P}$ that is guaranteed to be isomorphic to a polynomial system in some class of polynomial systems $\mathfrak{F}$, find 2 affine transformations $\mathcal{S}, \mathcal{T}$ and a polynomial system $\mathcal{F} \in \mathfrak{F}$ such that $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$.

Not much can be said about the EIP-Problem in general. It turned out to be very easy in the case of the original oil and vinegar scheme [24], however after changing the parameters it is believed that the scheme is now safe against key recovery attacks. The modified scheme is known as Unbalanced Oil and Vinegar or UOV.

3.3 Description of the UOV signature scheme

In the UOV signature scheme the polynomial map $\mathcal{F}$ consists of m randomly chosen UOV polynomials. The variables are partitioned into two sets, the first $v = n - m$

variables $x_1, \cdots, x_v$ are called vinegar variables, the last m variables $x_{v+1}, \cdots, x_n$ are the oil variables. Given this partition a UOV polynomial is a polynomial of the form

$$f(\mathbf{x}) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j} x_i x_j + \sum_{i=1}^{n} \beta_i x_i + \gamma \,.$$

In other words, a UOV polynomial is a quadratic polynomial in which no quadratic terms appear with two oil variables. The oil variables and the vinegar variables are not fully mixed, which is where the names come from. However it is not really a good name because in reality oil mixes with oil and vinegar mixes with vinegar but no mixing happens between oil and vinegar, and this is not what happens in UOV polynomials. A better name would have been hen variables and rooster variables because hens can get along with hens and roosters, but two roosters start a fight when they appear in the same term. Moreover, we will see later that it is essential for the security of UOV that there are more hen variables than rooster variables, which is natural for a flock of chickens.

A UOV system is a polynomial system that consists of UOV polynomials. UOV systems can be solved very easily. Suppose we want to find $s \in \mathbb{F}_q^n$ such that $\mathcal{F}(s) = x$ for some $x \in \mathbb{F}_q^m$. One simply chooses v random values for the vinegar variables and substitutes them into the equations. Since the system contains no quadratic terms with two oil variables the remaining system contains only linear equations. We get a system of m linear equations in the m oil variables, so we can solve this system rapidly to find the values of the oil variables. It might happen that the linear system has no solutions, in that case we can simply try again with different values for the vinegar variables.

When using the UOV scheme a number of parameters has to be chosen, these are

- q, the size of the finite field that is used.
- m, the number of equations in the polynomial systems and also the number of oil variables
- n, the total number of variables. The value of n and the value of m determine the value of $v = n - m$, the number of vinegar variables.
- $\mathcal{H}$, a hash function $\mathbb{F}_2^* \to \mathbb{F}_q^m$.

The key generation, signature generation and signature verification algorithms are displayed in Algs. 3.4,3.5 and 3.6

Remark. *In contrast to the general framework of MQ signature schemes we have not chosen an affine map $\mathcal{S}$ that scrambles the different polynomials of the central map $\mathcal{F}$. Having an $\mathcal{S}$ would not affect the distribution of the public keys because any affine bijection $\mathcal{S}$ acts as a permutation on the set of UOV systems. Therefore we can always choose $\mathcal{S} = I_m$ without influencing the security of the scheme. Also, it is easy to see that a translation $x \mapsto x + c$ for come constant vector $c \in \mathbb{F}_q^m$ acts as a permutation on the set of UOV systems, so without losing any security we can pick $\mathcal{T}$ to be a linear transformation instead of an affine transformation.*

Algorithm UOVGenerateKeys

input: Random bits to generate $\mathcal{F}$ and $\mathcal{T}$

output: $\mathcal{P}$ — A public key
$(\mathcal{F}, \mathcal{T})$ — A corresponding secret key

1: $\mathcal{F} \leftarrow$ a randomly chosen UOV system
2: $\mathcal{T} \leftarrow$ a randomly chosen linear map $\mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$
3: $\mathcal{P} \leftarrow \mathcal{F} \circ \mathcal{T}$
4: **return** $\mathcal{P}$ and $(\mathcal{F}, \mathcal{T})$

Alg. 3.4: The UOV key pair generation algorithm

Algorithm UOVSign

input: $(\mathcal{F}, \mathcal{T})$ — A public key
M — A message to sign

output: $\mathbf{s}$ — A signature for the message M

1: $\mathbf{h} \leftarrow \mathcal{H}(M)$
2: **while** No solution $\mathbf{s}'$ to the system $\mathcal{F}(\mathbf{s}') = \mathbf{h}$ is found **do**
3: Assign random values to the first v entries of $\mathbf{s}'$
4: Substitute these values into $\mathcal{F}(\mathbf{s}') = \mathbf{h}$ to get a linear system $L(\mathbf{o}) = \mathbf{h}$.
5: **if** $L(\mathbf{o}) = h$ has solutions **then**
6: Calculate an assignment $\mathbf{o}$ to the oil variables such that $L(\mathbf{o}) = \mathbf{h}$
7: Assign the entries of $\mathbf{o}$ to the last m entries of $\mathbf{s}'$
8: **end if**
9: **end while**
10: $\mathbf{s} \leftarrow \mathcal{T}^{-1}(\mathbf{s}')$
11: **return** $\mathbf{s}$

Alg. 3.5: The UOV signature generation algorithm

Algorithm UOVVerify

input: $\mathcal{P}$ — A public key
M — A message
$\mathbf{s}$ — A candidate–signature

output: **True** if $\mathbf{s}$ is a valid signature for M, **False** otherwise

1: $\mathbf{h} \leftarrow \mathcal{H}(M)$
2: $\mathbf{h}' \leftarrow \mathcal{P}(\mathbf{s})$
3: **if** $\mathbf{h} = \mathbf{h}'$ **then**
4: **return True**
5: **else**
6: **return False**
7: **end if**

Alg. 3.6: The UOV signature verification algorithm

3.3.1 Key and signature sizes of the UOV scheme

The public key is a quadratic map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$, so it consists of m quadratic polynomials in n variables. Each quadratic polynomial contains $n(n+1)/2$ quadratic terms, n linear terms and one constant term. Each coefficient requires $\lceil \log_2(q) \rceil$ bits to represent, so the total number of bits required to represent a public key is

$$m \left(\frac{n(n+1)}{2} + n + 1 \right) \lceil \log_2(q) \rceil .$$

A private key consists of the UOV map $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ and a linear map $\mathcal{T} \in GL(q, n)$. Each polynomial in the UOV map has $v(v+1)/2$ quadratic terms that contain two vinegar variables and vm quadratic terms that contain one vinegar variable and one oil variable. Therefore the size of a public key is equal to

$$\left(m \left(\frac{v(v+1)}{2} + vm + n + 1 \right) + n^2 \right) \lceil \log_2(q) \rceil \text{ bits.}$$

A signature s is a vector of n elements, so the size of a signature is $n\lceil \log_2(q) \rceil$ bits. For secure parameter choices the public and private keys are very large, ranging from several hundreds of kilobytes to megabytes. We refer to Table 3.1 for some secure parameter choices and corresponding public key sizes.

Remark. *When working over a field of characteristic 2 terms of the form cx_i^2 are linear, so they need not be represented as quadratic terms in the keys. This reduces the size of the public key by mn coefficients, and the private key by mv coefficients.*

3.4 Equivalent secret keys

For any UOV public key $\mathcal{P}$ there is a multitude of UOV systems $\mathcal{F}$ that are equal to $\mathcal{P}$ up to some change of coordinates $\mathcal{T}$. In other words, for any public key there are many secret keys. This fact can be useful for the designer of a UOV-like scheme, because some private keys could be stored more efficiently and be faster to calculate with than other keys. This idea is explained in Sect. 3.8. The notion of equivalent keys is also important for attackers of a UOV scheme because it suffices to find only one of the equivalent keys to be able to forge signatures. Therefore it is important to have a good understanding of how many equivalent keys there are and how the different keys are related. This is worked out by Wolf and Preneel in [35]. We will present a reformulation of their results here, but with different proofs based on the arguments in the thesis of Petzoldt [28].

Definition. *For a homogeneous quadratic polynomial f in n variables over a field k. If M_f is a $n \times n$ matrix such that $f(x) = x^T M_f x$ for all $x \in k^n$, we call this matrix a matrix representation of f.*

Remark. *It is easy to see that every homogeneous quadratic polynomial has a matrix representation. It is not unique, but it is unique up to addition of a skew symmetric matrix, i.e. a matrix A such that $A^\top = -A$.*

With this representation we can restate the definition of a UOV polynomial system. A quadratic polynomial f is a UOV polynomial if and only if there is a matrix representation of the quadratic part of f of the form

$$\mathbf{M}_f = \begin{pmatrix} \mathbf{M}_1 & \mathbf{M}_2 \\ \mathbf{M}_3 & \mathbf{0}_{m\times m} \end{pmatrix}.$$

Property. *Let f be a quadratic polynomial in n variables over a field k and let $\mathcal{T}$ be a linear transformation $k^n \to k^n$. Let $\boldsymbol{M}_f$ be a matrix representation for the quadratic part of f and let $\boldsymbol{M}_\mathcal{T}$ be the matrix representation of $\mathcal{T}$. Then we have that*

$$\boldsymbol{M}_{f\circ\mathcal{T}} = \boldsymbol{M}_\mathcal{T}^\top \boldsymbol{M}_f \boldsymbol{M}_\mathcal{T}$$

is a matrix representations of the quadratic parts of $f \circ \mathcal{T}$.

Theorem 1. *Let $\mathcal{F}$ be a UOV system and let $\mathcal{E}$ be a linear transformation with matrix representation*

$$\boldsymbol{M}_\mathcal{E} = \begin{pmatrix} \boldsymbol{A}_{v\times v} & \boldsymbol{0}_{v\times m} \\ \boldsymbol{B}_{m\times v} & \boldsymbol{C}_{m\times m} \end{pmatrix} \tag{3.3}$$

with invertible submatrices $\boldsymbol{A}$ and $\boldsymbol{C}$. Then we have that $\mathcal{F}\circ\mathcal{E}$ is a UOV system. Therefore, if $\mathcal{E}$ is invertible and $\mathcal{P}$ and $(\mathcal{F},\mathcal{T})$ form a UOV key pair then $(\mathcal{F}\circ\mathcal{E}, \mathcal{E}^{-1}\circ\mathcal{T})$ is an equivalent private key for $\mathcal{P}$.

Proof. The fact $\mathcal{P} = \mathcal{F}\circ\mathcal{E}\circ\mathcal{E}^{-1}\circ\mathcal{T}$ is trivial. We only have to show that $\mathcal{F}\circ\mathcal{E}$ is a UOV system. Let Q be the quadratic part of a UOV polynomial. Then, as explained above, Q has a matrix representation of the form

$$\mathbf{M}_Q = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{Q}_3 & \mathbf{0}_{m\times m} \end{pmatrix}.$$

We have a matrix representation for $Q\circ\mathcal{E}$ which is given by

$$\begin{aligned} \mathbf{M}_\mathcal{E}^\top \mathbf{M}_q \mathbf{M}_\mathcal{E} &= \begin{pmatrix} \mathbf{A}^\top & \mathbf{B}^\top \\ \mathbf{0}_{m\times v} & \mathbf{C}^\top \end{pmatrix} \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{Q}_3 & \mathbf{0}_{m\times m} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{0}_{v\times m} \\ \mathbf{B} & \mathbf{C} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{A}^\top\mathbf{Q}_1\mathbf{A} + \mathbf{A}^\top\mathbf{Q}_2\mathbf{B} + \mathbf{B}^\top\mathbf{Q}_3\mathbf{A} & \mathbf{A}^\top\mathbf{Q}_2\mathbf{C} \\ \mathbf{C}^\top\mathbf{Q}_3\mathbf{A} & \mathbf{0}_{m\times m} \end{pmatrix}. \end{aligned}$$

Therefore composing a UOV system with $\mathcal{E}$ gives another UOV system. □

Corollary 1. *For any public UOV key there are at least $q^{vm}\prod_{i=0}^{v-1}(q^v - q^i)\prod_{i=0}^{m-1}(q^m - q^i)$ corresponding secret keys.*

Proof. When choosing $\mathcal{E}$ in the previous theorem there are $|GL(q,v)|$ choices for $\mathbf{A}$, $|GL(q,m)|$ choices for $\mathbf{C}$ and q^{mv} choices for $\mathbf{B}$. Using the fact that

$$|GL(q,n)| = \prod_{i=0}^{n-1}(q^n - q^i)$$

we obtain the above formula. □

Theorem 2. *If $(\mathcal{P}, (\mathcal{F}, \mathcal{T}))$ is a UOV key pair such that the upper left $v \times v$ matrix of the matrix representation of $\mathcal{T}$ is invertible, then there is an equivalent secret key $(\mathcal{F}', \mathcal{T}')$ such that $\mathcal{T}'$ has the matrix representation*

$$\boldsymbol{M}_{\mathcal{T}} = \begin{pmatrix} \boldsymbol{I}_v & \boldsymbol{T}' \\ \boldsymbol{0}_{m\times v} & \boldsymbol{I}_m \end{pmatrix},$$

with some $v \times m$ matrix $\boldsymbol{T}'$.

Proof. Let $\mathcal{P}$ and $(\mathcal{F}, \mathcal{T})$ be such a UOV key pair. We write the matrix representation of $\mathcal{T}$ as

$$M_{\mathcal{T}} = \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_2 \\ \mathbf{T}_3 & \mathbf{T}_4 \end{pmatrix},$$

where $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3$ and $\mathbf{T}_4$ are $v \times v, v \times m, m \times v$ and $m \times m$-matrices respectively. By assumption we have that $\mathbf{T}_1$ is invertible. If this is the case we can consider the linear map $\mathcal{E}_1$ with matrix representation

$$\mathbf{M}_{\mathcal{E}_1} = \begin{pmatrix} \mathbf{I}_v & \mathbf{0}_{v\times m} \\ \mathbf{T}_3\mathbf{T}_1^{-1} & \mathbf{I}_m \end{pmatrix}.$$

According to the previous theorem we know that $(\mathcal{F} \circ \mathcal{E}_1, \mathcal{E}_1^{-1} \circ \mathcal{T})$ is an equivalent key for $\mathcal{P}$. The new linear transformation $\mathcal{E}^{-1} \circ \mathcal{T}$ has the matrix representation

$$\begin{pmatrix} \mathbf{I}_v & \mathbf{0} \\ -\mathbf{T}_3\mathbf{T}_1^{-1} & \mathbf{I}_m \end{pmatrix} \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_2 \\ \mathbf{T}_3 & \mathbf{T}_4 \end{pmatrix} = \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_2 \\ \mathbf{0} & \mathbf{T}_4 - \mathbf{T}_3\mathbf{T}_1^{-1}\mathbf{T}_2 \end{pmatrix}.$$

This shows that $\mathbf{T}_4 - \mathbf{T}_3\mathbf{T}_1^{-1}\mathbf{T}_2$ is invertible. So we can define a second linear transformation $\mathcal{E}_2$ with matrix representation

$$\begin{pmatrix} \mathbf{T}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_4 - \mathbf{T}_3\mathbf{T}_1^{-1}\mathbf{T}_2 \end{pmatrix}.$$

According to the previous theorem we know know that $(\mathcal{F} \circ \mathcal{E}_1 \circ \mathcal{E}_2, \mathcal{E}_2^{-1} \circ \mathcal{E}_1^{-1} \circ \mathcal{T})$ is an equivalent private key. This private key has a linear transformation with matrix representation

$$\begin{pmatrix} \mathbf{T}_1^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{T}_4 - \mathbf{T}_3\mathbf{T}_1^{-1}\mathbf{T}_2)^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{T}_1 & \mathbf{T}_2 \\ \mathbf{0} & \mathbf{T}_4 - \mathbf{T}_3\mathbf{T}_1^{-1}\mathbf{T}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{I}_v & \mathbf{T}_1^{-1}\mathbf{T}_2 \\ \mathbf{0} & \mathbf{I}_m \end{pmatrix},$$

which finishes the proof. □

Corollary 2. *Usually a UOV key pair is generated with the invertible map $\mathcal{T} : {\mathbb{F}_q}^n \to {\mathbb{F}_q}^n$ chosen randomly. Then we have that the upper $v \times v$ matrix is invertible with a high probability, so with a large probability there is an equivalent secret key with matrix representation of $\mathcal{T}$ of the form*

$$\begin{pmatrix} \boldsymbol{I}_v & \boldsymbol{T} \\ \boldsymbol{0}_{m\times v} & \boldsymbol{I}_m \end{pmatrix}.$$

Remark. *Contrary to what is mentioned in [35] this theorem does not hold for all UOV keys. A counterexample in the case $m = v$ is given by a public key $\mathcal{P}$ consisting of m quadratic equations that only contain the oil variables. These equations have matrix representations of the form*

$$\begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix}.$$

A secret key for this public key is given by $(\mathcal{F}, \mathcal{T})$ with $\mathcal{F}$ a UOV system that consists of m equations that only contain the vinegar variables and the linear transformation that swaps the oil variables and the vinegar variables. However, this UOV key pair has no equivalent key of the form of the previous theorem because for any choice of $\boldsymbol{T}$ we have

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{T}^\top & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{T} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix}.$$

This cannot be the matrix representation of a UOV polynomial since the lower right $m \times m$ matrix is not skew symmetric.

3.5 Classical Attacks against UOV

3.5.1 Direct attack

In a direct attack an attacker tries to forge a signature s for a message M by solving the polynomial system $\mathcal{P}(s) = \mathcal{H}(M)$. An attacker can use the trick of Thomae and Wolf [34] to reduce this problem to finding a solution of a polynomial system with $m + 1 - \lfloor n/m \rfloor$ equations. The best known algorithm to solve this is the hybrid approach [6] which was briefly described in Sect. 3.2.1. Empirically, the systems that have to be solved behave like semi-regular systems [19], therefore we calculate the degree of regularity and use this to estimate of the complexity of the hybrid approach. This is essentially the same method as the method used by Petzoldt [28] to estimate the complexity of a direct attack against UOV, the only difference being that we have used an updated estimate of the complexity of F_5. In Petzoldt's thesis it was shown that the estimated complexity of a direct attack agrees very well with the measured complexity of a direct attack against small instances of UOV.

Example. *We will estimate the complexity of a direct attack against UOV with the parameter set $(q = 31, m = 52, v = 104)$; this set is proposed in [28] as a set that achieves 128-bit security. Using the trick of Thomae we can reduce finding a solutions to finding a solution of a determined system with $52+1-\lfloor(52+104)/52\rfloor = 50$ equations. We assume this system to be semi-regular. If we fix k extra variables the degree of regularity is then the degree of the first term in the power series of*

$$S_{50,50-k}(x) = \frac{(1-x^2)^{50}}{(1-x)^{50-k}}$$

which has a non-positive coefficient. For $k = 0$ this is we have $S_{50,50}(x) = (1+x)^{50}$, so the degree of regularity is 51. For $k = 1$ we have

$$S_{50,49}(x) = 1 + 49x + 1175x^3 + \cdots + 4861946401452x^{25} - 4861946401452x^{26} + O(x^{27})$$

where all the omitted terms have positive coefficients, so the degree of regularity is 26*. For* $k = 2$ *and* $k = 3$ *we can do the same calculation to get that the degree of regularity is* 23 *and* 21 *respectively. We can now use (3.1) to estimate the complexity of the hybrid approach. We prefer to error on the side of caution, so we have chosen* $\omega = 2$ *for the value of the linear algebra constant. For* k *equal to* $0, 1, 2$ *and 3 this is equal to*

$$\begin{aligned}
\binom{50+51}{51}^2 &\approx 2^{194.7}, \\
31\binom{50-1+26}{26}^2 &\approx 2^{137.8}, \\
31^2\binom{50-2+23}{23}^2 &\approx 2^{132.3}, \\
31^3\binom{50-3+21}{21}^2 &\approx 2^{129.6}
\end{aligned}$$

respectively. Continuing this for even higher values of k *we eventually see that the optimal value of* k *is 6, the corresponding degree of regularity is* 16 *and the complexity of the direct attack is* $2^{123.9}$*.*

In the example we concluded that the complexity of the attack is less than 2^{128} which was supposed to be the security level of the parameter set $(q = 31, m = 52, v = 104)$ according to [28]. Even though we have used roughly the same method of estimating the complexity as the method used in in [28] we arrive at a slightly different value because we have used a tighter bound on the complexity of F_5 coming from an improved analysis of the hybrid approach [7].

With this method we can calculate the minimal number of equations that is needed in a determined semi-regular system in order to guarantee that the complexity of finding a solution is larger than a targeted security level. For quantum attackers, we can follow the same method with (3.2) instead of (3.1) for estimating the complexity of the hybrid approach. The result of these calculations for the security levels of 2^{128} and 2^{256} for different finite fields of size up to $q = 2^{100}$ are plotted in Fig. 3.2.

3.5.2 UOV attack

In the original version of the Oil and Vinegar scheme proposed by Patarin it was suggested to use the scheme with $v = m$, that is, with the same number of vinegar variables as oil variables. It turns out that this choice is not safe. This version of the scheme was broken by Kipnis and Shamir [24]. Roughly, they showed that it is possible for an attacker to find the inverse image of the oil variables under the map $\mathcal{T}$. This is enough information to find an equivalent secret key, so this breaks the scheme. This approach generalizes for the case $v > m$, but it gets exponentially harder as $v - m$ grows larger, it has a complexity of $O(q^{v-m}m^4)$ [23]. Typically one chooses $v = 2m$ or $v = 3m$, then the UOV attack is not feasible anymore.

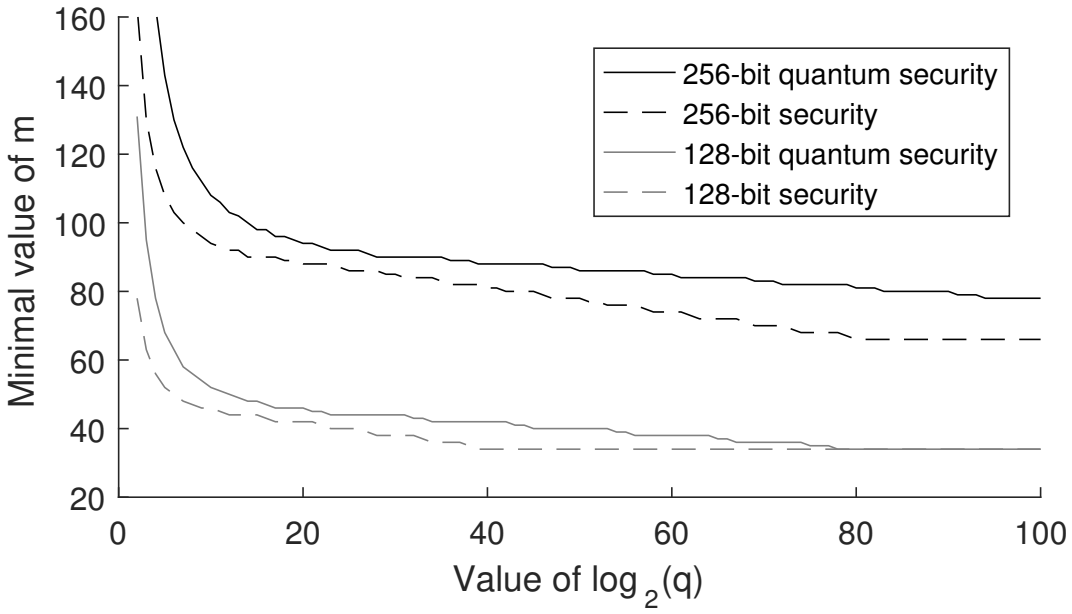

Figure 3.2: The minimal size of a determined semi-regular system to reach 128-bit of security and 256-bit security for different finite fields.

We will briefly describe the original attack, which works in the case $v = m$. For a more detailed version we refer to [24] and for the case $v > m$ we refer to [23]. We consider two subspaces of $\mathbb{F}_q^{2m}$.

Definition. *The* **vinegar subspace***, denoted by V is the subspace of $\mathbb{F}_q^{2m}$ that is spanned by the first m unit vectors. Equivalently the vinegar subspace is the subspace of all vectors in $\mathbb{F}_q^{2m}$ of which the last m entries are all zero.*

Definition. *The* **oil subspace***, denoted by O is the subspace of $\mathbb{F}_q^{2m}$ that is spanned by the last m unit vectors. Equivalently the oil subspace is the subspace of all vectors in $\mathbb{F}_q^{2m}$ of which the first m entries are all zero.*

The goal of the attack is to find the inverse image of O under $\mathcal{T}$. If an attacker can do this he can efficiently sign any message. Indeed, the attacker can simply pick any linear transformation $\mathcal{T}'$ that maps O to $\mathcal{T}^{-1}(O)$. Then we have that $\mathcal{T} \circ \mathcal{T}'$ maps the oil subspace onto itself, so the matrix representation of $\mathcal{T} \circ \mathcal{T}'$ is of the form (3.3). We have

$$\mathcal{P} = \mathcal{F} \circ (\mathcal{T} \circ \mathcal{T}') \circ \mathcal{T}'^{-1}. \tag{3.4}$$

According to Theorem 1 we have that $\mathcal{F} \circ (\mathcal{T} \circ \mathcal{T}')$ is a UOV system, so $(\mathcal{P} \circ \mathcal{T}', \mathcal{T}'^{-1})$ is an equivalent secret key which can be used to sing any message.

Now we explain how an attacker can find $\mathcal{T}'^{-1}(O)$ efficiently. At our disposal we have the polynomials f_i of the system $\mathcal{P}$, so we have m matrices $\mathbf{P}_i$ that represent the quadratic part of p_i respectively. We know that there exist some unknown private key $(\mathcal{F}, \mathcal{T})$. Therefore there is a matrix $\mathbf{T}$ and matrices $\mathbf{F}_i$ that represent the quadratic part of UOV polynomials such that, $\mathbf{P}_i = \mathbf{T}^\top \mathbf{F}_i \mathbf{T}$. We will use the following lemma.

Lemma 1. *Let $\boldsymbol{F}$ be the matrix representation of the quadratic part of a UOV polynomial. Then, the linear map with matrix representation $\boldsymbol{F}$ sends the oil subspace into the vinegar subspace. If $\boldsymbol{F}$ is an invertible matrix, then $\boldsymbol{F}^{-1}$ sends the vinegar subspace to the oil subspace.*

Proof. We can assume that $\mathbf{F}$ has the form

$$\begin{pmatrix} \mathbf{F}_1 & \mathbf{F}_2 \\ \mathbf{F}_3 & \mathbf{0}_{m\times m} \end{pmatrix},$$

with $\mathbf{F}_1, \mathbf{F}_2$ and $\mathbf{F}_3$ matrices of size $m \times m$. It is trivial to check that a vector of the oil subspace is mapped into the vinegar subspace. Since the oil subspace and the vinegar subspace have the same dimension the inverse map $\mathbf{F}^{-1}$ sends the vinegar space to the oil space if $\mathbf{F}$ is invertible. □

The crucial observation is that if a matrix $\mathbf{P}_i$ and $\mathbf{P}_j$ are invertible then $\mathbf{P}_i^{-1}\mathbf{P}_j$ sends $\mathcal{T}^{-1}(O)$ to itself. (We say that $\mathcal{T}^{-1}(O)$ is an eigenspace of $\mathbf{P}_i^{-1}\mathbf{P}_j$) Indeed, it is easily checked that

$$\begin{aligned} \mathbf{P}_i^{-1}\mathbf{P}_j(\mathcal{T}^{-1}(O)) &= \mathbf{T}^{-1}\mathbf{F}_i^{-1}\mathbf{T}^{\top-1}\mathbf{T}^{\top}\mathbf{F}_j\mathbf{T})(\mathcal{T}^{-1}(O)) \\ &= \mathbf{T}^{-1}\mathbf{F}_i^{-1}\mathbf{F}_j(O) \\ &= \mathbf{T}^{-1}\mathbf{F}_i^{-1}(V) \\ &= \mathcal{T}^{-1}(O)\,. \end{aligned}$$

Therefore $\mathcal{T}^{-1}(O)$ is the common eigenspace of all matrices of the form $\mathbf{P}_i^{-1}\mathbf{P}_j$ with $\mathbf{P}_i$ and $\mathbf{P}_j$ invertible matrices. The common eigenspace of a large set of matrices can be computed efficiently, so this completes the attack.

3.5.3 UOV reconciliation attack

Similar to the UOV attack, the UOV reconciliation attack tries to find an equivalent secret key. The attack was first described in [16]. Here we give a very short summary of the attack. The attack is based on Corollary 2, which says that for a public key $\mathcal{P}$ there is with a very high probability a private key $(\mathcal{F}, \mathcal{T})$ such that the matrix representation of $\mathcal{T}$ is of the form

$$\mathbf{M}_{\mathcal{T}} = \begin{pmatrix} \mathbf{I}_v & \mathbf{T} \\ 0 & \mathbf{I}_m \end{pmatrix}.$$

This means that an attacker only has to find the $v \times m$ matrix $\mathbf{T}$ to get an equivalent key. For any polynomial p_i in the public polynomial system $\mathcal{P}$ we consider a matrix representation $\mathbf{M}_i$ for the quadratic part of p_i. If $(\mathcal{F}, \mathcal{T})$ is a valid secret key, we have that $p_i \circ \mathcal{T}^{-1} = f_i$ is a UOV polynomial with a matrix representation

$$\begin{pmatrix} \mathbf{F}_{i1} & \mathbf{F}_{i2} \\ \mathbf{F}_{i3} & \mathbf{F}_{i4} \end{pmatrix} := \begin{pmatrix} \mathbf{I}_v & 0 \\ -\mathbf{T}^{\top} & \mathbf{I}_m \end{pmatrix} \mathbf{M}_i \begin{pmatrix} \mathbf{I}_v & -\mathbf{T} \\ 0 & \mathbf{I}_m \end{pmatrix}.$$

Since f_i is a UOV polynomial we know that the $m \times m$ matrix $\mathbf{F}_{i4}$ has to be skew symmetric. Therefore for each $i \in \{1, \ldots, m\}$ we have $m(m-1)/2$ equations of the form

$(\mathbf{F}_{i4})_{i,j} = -(\mathbf{F}_{i4})_{j,i}$. The entries of $\mathbf{F}_{i4}$ are quadratic functions of the entries of $\mathbf{T}$, so in total we have $m^2(m-1)/2$ quadratic equations in the vm entries of $\mathbf{T}$, so we can solve this system to find $\mathbf{T}$.

The resulting system of equations is far from random and it is much easier to solve than a random system of equations. Ding et al. argue that the complexity of this attack for UOV variants with $v \leq m$ (such as Rainbow and TTS) is the same as the complexity of solving a random system of m equations in v variables [16]. For the case $v > m$ we can use this to derive a lower bound on the complexity of the reconciliation attack. Adding equations to an overdetermined system only makes solving it easier, so the UOV reconciliation attack on a UOV scheme with m equations with $v > m$ vinegar variables is at least as difficult as a UOV reconciliation attack on a system with v equations and v vinegar variables. This complexity is the same as the complexity of solving a random system of v equations in v variables.

We conclude that an UOV reconciliation attack on a UOV system with m equations and $v > m$ vinegar variables is at least as difficult as solving a random system of v variables in v equations. (But it is in general probably much more difficult.) So, for the typical choice of $v = 2m$ or $v = 3m$ the UOV reconciliation attack is much more difficult than attacking the system $\mathcal{P}(s) = \mathcal{H}(M)$ directly.

3.5.4 Hash collision attack

If an attacker knows a hash collision, i.e. two messages m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$ he could possibly convince the holder of the secret key to produce a signature s form m_1, but s would also be a valid signature for m_2. So, the signature is not secure in the EUF-CMA model. This attack could also lead to real world security threats. For example, the message m_1 could be "I donate 5 dollars to X", while the second message could be "I donate 1000 dollars to Y". A cunning adversary could trick someone into donating five dollars to a charity, and make him unknowingly donate a large sum to some evil organization.

This is a general attack which works against any signature which follows the hash-and-sign paradigm (see Sect. 2.3). To protect against this attack it should be impossible to find hash collisions. In the random oracle model this is achieved if the output of the hash function has at least twice the amount of bits as the desired security level.

3.6 Quantum attacks against UOV

3.6.1 Direct attack and Reconciliation attack

The most expensive parts of the direct attack and reconciliation attack are solving systems of polynomials over finite fields. In Sect. 3.2.3 we discussed that Grover search could be used to speed up the algorithms that solve the MQ problem. Therefore, if we want the scheme to be secure against quantum attackers we will need to increase the size of the public map $\mathcal{P}$ and the number of vinegar variables.

3.6.2 UOV attack

The UOV attack tries to reconstruct the oil subspace. It does this by repeatedly picking a random linear combination of the polynomials in the public system and doing some cheap linear algebra operations which yield a nonzero vector in the oil subspace with probability q^{v-m}. Quantum computers could speed up this attack. A Grover search can be used to find the right linear combinations, thereby reducing the complexity of the attack from $q^{v-m}m^4$ to $q^{(v-m)/2}m^4$.

3.6.3 Hash Collision attacks

There is a popular myth that the complexity of the Brassard-Høyer-Tapp quantum algorithm for collision finding in a κ-bit hash is $O(2^{\kappa/3})$, which would be less than the $O(2^{\kappa/2})$ which is the complexity for classical computers. This myth is debunked by Bernstien [4] who argues that the complexity the Brassard-Høyer-Tapp quantum algorithm is also $O(2^{\kappa/2})$, and that the quantum algorithms benefit less from parallelization than classical algorithms employing the rho method. Therefore, perhaps surprisingly, quantum computers are not cost-effective for finding hash collisions, so quantum computers will not help an adversary to preform a hash collision attack.

3.7 Reducing public key size by generating part of it with a PRNG

In this section we present a method by Petzoldt that drastically reduces the size of the public keys of the UOV signature scheme [28]. The idea is to generate a large part of the coefficients of the public system $\mathcal{P}$ with a pseudo-random number generator (PRNG). Then we can include the seed for the random number generator in the public key instead of all the generated coefficients, reducing the size of the public key. It is clear that we cannot hope to generate the entire public map $\mathcal{P}$ with a PRNG, because a randomly chosen map has a very low probability of being a UOV public key. However, it is possible to generate a part of $\mathcal{P}$ with a PRNG and calculate the rest of $\mathcal{P}$ such that it is a valid public key.

Usually we first choose $\mathcal{F}$ and $\mathcal{T}$ and compute $\mathcal{P} = \mathcal{F} \circ \mathcal{P}$, but this way we cannot control the coefficients of $\mathcal{P}$. To make the idea work Petzoldt proposed another approach: First we choose $\mathcal{T}$ and a large part of the coefficients of $\mathcal{P}$. Then we solve the system $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ to find the coefficients of $\mathcal{F}$, and the remaining coefficients of $\mathcal{P}$.

3.7.1 The modified key generation algorithm

Let $D = n(n+1)/2$ denote the total number of coefficients for a homogeneous quadratic polynomial in $n = m + v$ variables. We can partition the coefficients in a set of $D_1 = v(v+1)/2+mv$ coefficients corresponding to monomials which contain at least one vinegar variable, and a set of $D_2 = m(m+1)/2$ coefficients, which correspond to monomials in the oil variables. Let $\mathbf{P}, \mathbf{F} \in \mathbb{F}_q^{m \times D}$ be the Macaulay matrices of $\mathcal{P}$ and $\mathcal{F}$ respectively, and let $\mathbf{T}$ be the matrix representation of $\mathcal{T}$, then the equation $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ corresponds

to the matrix equation

$$\mathbf{P} = \mathbf{F}\mathbf{A}\,,$$

where $\mathbf{A}$ is the $D \times D$-matrix whose entries are given by

$$\mathbf{A}_{ij,rs} = \begin{cases} \mathbf{T}_{r,i} \cdot \mathbf{T}_{s,i} & \text{if } i = j \\ \mathbf{T}_{r,i} \cdot \mathbf{T}_{s,j} + \mathbf{T}_{r,j} \cdot \mathbf{T}_{s,i} & \text{if } i \neq j \end{cases}.$$

Note that we index the rows and columns of A by pairs ij, where the ij-th row/column is the row/column corresponding to the monomial $x_i x_j$. We use a lexicographic ordering of the pairs to order the columns and rows. Then we have that the first D_1 rows/columns of the matrix correspond to the monomials that contain at least one vinegar variable, and the last D_2 correspond to monomials in the oil variables. We split up the matrices $\mathbf{P}, \mathbf{F}$ and $\mathbf{A}$ in parts. Let $\mathbf{P}_1, \mathbf{F}_1 \in \mathbb{F}_q^{m \times D_1}$ and $\mathbf{P}_2, \mathbf{F}_2 \in \mathbb{F}_q^{m \times D_2}$ consist of the first D_1 and last D_2 columns of $\mathbf{P}$ and $\mathbf{F}$ respectively. Since $\mathcal{F}$ is a UOV map we know that $\mathbf{F}_2 = 0$. Let $\mathbf{A}_{11}$ be the upper left $D_1 \times D_1$-submatrix of $\mathbf{A}$, $\mathbf{A}_{12}$ the upper right $D_1 \times D_2$-submatrix of $\mathbf{A}$, $\mathbf{A}_{21}$ the lower left $D_2 \times D_1$-submatrix of $\mathbf{A}$ and $\mathbf{A}_{22}$ the lower right $D_2 \times D_2$-submatrix of $\mathbf{A}$. The we have the equation

$$\begin{pmatrix} \mathbf{P}_1 & \mathbf{P}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{F}_1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}.$$

Petzoldt showed empirically that for a random choice of $\mathcal{T}$ the square matrix $\mathbf{A}_{11}$ has a very large probability of being invertible. This means that with high probability we can solve $\mathbf{P}_1 = \mathbf{F}_1\mathbf{A}_{11}$ for $\mathbf{F}_1$. Now we can describe the key generation algorithm.

Key generation algorithm:

Algorithm PetzoldtKeyGen

input: Random bits to choose s and $\mathcal{T}$

output: $(s, \mathbf{P}_2)$ — A public key
$(\mathbf{F}_1, \mathcal{T})$ — A corresponding private key

1: Randomly choose a seed s for a prng
2: Generate $\mathbf{P}_1$ using s
3: **while** $\mathbf{F}_1$ is not found **do**
4: | Randomly choose $\mathcal{T}$
5: | Calculate $\mathbf{A}_{11}$
6: | Try to solve $\mathbf{P}_1 = \mathbf{F}_1\mathbf{A}_{11}$ for $\mathbf{F}_1$
7: **end while**
8: Calculate $\mathbf{A}_{12}$
9: Calculate $\mathbf{P_2} = \mathbf{F}_1\mathbf{A}_{12}$
10: **return** $(s, \mathbf{P}_2)$ and $(\mathbf{F}_1, \mathcal{T})$

Alg. 3.7: An improved Key generation algorithm

3.7.2 Results

With this approach the public key size is decreased by $m(v(v+1)/2+ov)$ field elements, at the negligible cost of including the seed for the random number generator. The public key size is now $mo(o+1)/2\log_2(q)+|\text{seed}|$. Table 3.1 shows that this method drastically reduces the size of the public key. A disadvantage of the method is that during the key-pair generation phase we have to solve a very large system of linear equations, which makes the key generation algorithm quite slow. A solution to this problem is given in the next section.

Table 3.1: The effect of Petzoldt's method on the public key size

security level	q	(o,v)	public key (kB)	public key with Petzoldt's method (kB)
100 bits	2^8	(36,72)	207	23
128 bits	2^8	(47,94)	460	52
192 bits	2^8	(72,144)	1648	185
256 bits	2^8	(98,196)	4150	464

Instead of inserting a pseudo-randomly generated submatrix into the Macaulay matrix of the public key it is also possible to use this approach to insert a more structured submatrix. Such an approach can still reduce the size of the public key and have the additional benefit of speeding up the verification process [30].

3.8 Speeding up key-pair generation

As explained in Sect. 3.4 there is for each public key a large number of possible secret keys. In this section we show that it is possible to pick a particularly nice secret key such that the size of the secret key is slightly smaller, and the key-pair generation and signature generation algorithms are much faster. The idea was introduced by Czypek [14], but here we develop the idea independently and we describe it in more detail.

The idea is to take $\mathcal{T}$ of the form of Theorem 2, that is we take a $\mathcal{T}$ with matrix representation of the form

$$\begin{pmatrix} \mathbf{I}_v & \mathbf{T} \\ 0 & \mathbf{I}_m \end{pmatrix},$$

where $\mathbf{T}$ is a randomly chosen $v \times m$-matrix. This approach of choosing $\mathcal{T}$ instead of taking $\mathcal{T}$ uniformly at random does not affect the security of the scheme. This is so because Theorem 2 says that for a random public key there is a large probability that there exists a private key with $\mathcal{T}$ of this form. That means that if there is an attack against the modified signature scheme, that attack would work on nearly all public keys of the original UOV scheme, so the modified scheme is at least as secure as the original scheme.

A first advantage of this approach is that we only have to store a $v \times m$-matrix instead of a $n \times n$-matrix, so the size of the private key gets smaller. However, the size of $\mathcal{T}$ is often negligible in comparison with the size of $\mathcal{F}$, so this does not have a very important impact.

More importantly, this choice of $\mathcal{T}$ makes $\mathbf{A}_{11}$ an upper unitriangular matrix, which means that we can solve the linear system $\mathbf{F}_1 \cdot \mathbf{A}_{11} = \mathbf{P}_1$ for $\mathbf{F}_1$ very fast by using forward substitution. More precisely we have $\mathbf{A}_{11} = \mathbf{I}_{D_1} + \mathbf{B} + \mathbf{C}$, where $\mathbf{B}, \mathbf{C} \in \mathbb{F}_q^{D_1 \times D_1}$ are sparse strictly upper triangular matrices defined as

$$\mathbf{B}_{ij,rs} = \begin{cases} \mathbf{T}_{s,j-v} & \text{if } s \leq v < j \text{ and } r = i \\ 0 & \text{otherwise} \end{cases},$$

$$\mathbf{C}_{ij,rs} = \begin{cases} \mathbf{T}_{r,j-v} & \text{if } v < j \text{ and } s = i \\ 0 & \text{otherwise} \end{cases}.$$

This results in Alg. 3.8 for solving $\mathbf{F}_1 \cdot \mathbf{A}_{11} = \mathbf{P}_1$ for $\mathbf{F}_1$.

Algorithm FindF1

input: $\mathbf{P}_1$ — Part of Macaulay matrix of $\mathcal{P}$
$\mathbf{T}$ — A $v \times m$ matrix
output: $\mathbf{F}_1$ — The Macaulay matrix for $\mathcal{F}$

```
F_1 ← P_1
for i from 1 to v do
    for j from 1 to o do
        for s from i to v do
            Subtract T_{s,j} times the is-th col of F_1 from the ij-th col of F_1
        end for
        for r from 1 to i do
            Subtract T_{r,j} times the ri-th col of F_1 from the ij-th col of F_1
        end for
    end for
end for
return P_1
```

Alg. 3.8: Algorithm for solving $\mathbf{F}_1 \cdot \mathbf{A}_{11} = \mathbf{P}_1$ for $\mathbf{F}_1$.

The complexity of this algorithms is $O(v^2 m^2)$ (that is $v^2 m$ for doing the for loops times m for subtracting the columns). When v is proportional to m this makes the complexity of the algorithm $O(m^4)$. In contrast, the previous algorithm required solving m linear systems of D_1 equations in D_1 variables, this has complexity $O(mD_1^3) = O(m^7)$. In practice the new algorithm is several orders of magnitude faster.

Chapter 4

Reducing key sizes by lifting the UOV map to a large extension field

In this section we will work with UOV over a finite field $\mathbb{F}_{2^r}$ of characteristic 2. The parameter r is quite important for the security of the scheme, the signature size and key sizes. Figure 3.2 shows that by choosing a larger value of r we can put a smaller number of equations in the system and still reach the same level of security. Since the number of field elements in the public key and secret key is $O(m^3)$ it is very desirable to have a small value of m. However, since it costs r bits to store a field element r should not be too big either. We must make a trade off between large r and large m. However, in this section we propose a scheme that gets some security benefits of a high value of r, but has a public and private key with coefficients in $\mathbb{F}_2$, greatly reducing the key sizes.

A paper presenting the contents of this chapter was submitted to the Selected Areas of Cryptography (SAC) conference and is currently being reviewed.

4.1 Description of the new scheme

As usual, the public key of the scheme represents a quadratic system over $\mathbb{F}_{2^r}$, given by

$$\mathcal{P} = \mathcal{F} \circ \mathcal{T}\,.$$

When we want to sign a message m we use a hash function to generate a digest of mr bits which represents a vector x of m elements of $\mathbb{F}_{2^r}$. Then we use the knowledge of the private key to solve the system $\mathcal{P}(s) = x$ to get a valid signature s. However, the difference with standard UOV is that we now choose all the coefficients of $\mathcal{F}, \mathcal{P}$ and all entries of T in $\mathbb{F}_2$. Therefore the key generation process is identical to the key generation process of a regular UOV scheme over $\mathbb{F}_2$. In particular we can use the approach of Petzoldt [28] as explained in Sect. 3.7 to generate the first $\frac{v(v-1)}{2} + ov$ columns of $\mathbf{B}$, the Macaulay matrix of $\mathcal{P}$, using a random number generator and use $\mathbf{T}$ to deduce $\mathcal{F}$ and the remaining $\frac{o(o-1)}{2}$ columns of $\mathbf{B}$. Contrary to the key generation, the signature generation and verification still happen over entire field $\mathbb{F}_{2^r}$ as usual.

This way the public key only consists of a seed for the random number generator and $m\frac{o(o+1)}{2}$ bits, representing the last part of $\mathbf{B}$. The public key is approximately a factor r smaller than if we were to use the regular UOV scheme over $\mathbb{F}_{2^r}$ since we only use one bit to represent each field element instead of r bits. Furthermore, we can choose r to be much larger than what would otherwise be the optimal value of r. This in turn allows for a smaller value of m (See Fig. 3.2), reducing the public key size even more.

Remark. *Though we have presented this scheme with a finite field of characteristic 2 and with the subfield $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$, it is easy to see that we can use this scheme with any field extension of finite fields $K \subset K'$. In such a scenario we generate a key pair with coefficients in the small field K, and the signing and verifying is done with elements of the big field K'.*

4.2 Security analysis of the new scheme

4.2.1 Direct attack

In a Direct attack we try to forge a signature for a certain message M by trying to find a solution $s \in \mathbb{F}_{2^r}^n$ for the system $\mathcal{F}(s) = \mathcal{H}(M)$. The best known method for this is the hybrid approach as described in Sect. 3.2.1.

When we do a direct attack against the new scheme the polynomials of the system that needs to be solved has all coefficients in $\mathbb{F}_2$ except those of the constant monomials, because those coefficients come from the message digest. We claim that this does not significantly increase the efficiency of a hybrid approach. It has been noticed by Faugère and Perret [19] that the polynomials systems that result from fixing $\approx v$ variables in a UOV system tend to behave like semi-regular systems. The degree of regularity of a quadratic semi-regular system is given by the degree of the first term in the power series of

$$\frac{(1-x^2)^m}{(1-x)^n}$$

with a non-positive coefficient. In particular the degree of regularity does not depend on q for semi-regular systems. Therefore we expect the degree of regularity for a direct attack against the modified UOV scheme to be identical to the degree of regularity of an attack against the regular UOV scheme. Therefore it seems likely that a Gröbner basis computation against the modified scheme is not significantly more efficient than a Gröbner basis computation against regular UOV with the same parameters. This heuristic argument is confirmed by the experimental data in Table 4.1. There we see that a direct Gröbner basis attack is slightly faster against the modified scheme than against the original UOV scheme, but only by a constant factor. Even though the Gröbner basis calculation is done over $\mathbb{F}_{2^r}$, the largest part of the arithmetic only involves the field elements 0 and 1, so the arithmetic is faster than with generic elements of $\mathbb{F}_{2^r}$. We believe that this is where the difference observed in Table 4.1 comes from. If we do the same experiment with a smaller extension field such as $\mathbb{F}_{2^8}$ there is no observed difference between the running time of a direct attack against a regular UOV scheme and our modified scheme.

Remark. *In a direct attack one usually fixes $\approx v$ variables randomly to make the system a slightly overdetermined system. In our experiments we have fixed these variables to values in $\mathbb{F}_2$ to make sure that we do not introduce linear terms with coefficients in $\mathbb{F}_{2^r}$ instead of $\mathbb{F}_2$ in the case of the modified UOV scheme.*

Table 4.1: Running time of a direct attack against the regular UOV scheme over $\mathbb{F}_{2^{64}}$ and the modified UOV scheme, with the MAGMA implementation of the F4 algorithm.

(m,v)	regular UOV scheme	modified UOV scheme	difference
(7,14)	0.23 s	0.13 s	-43 %
(8,16)	1.18 s	0.60 s	-49 %
(9,18)	6.36 s	3.15 s	-50 %
(10,20)	44.3 s	22.2 s	-50 %
(11,22)	286.8 s	147.3 s	-48 %

4.2.2 Key recovery attack

In contrast to a direct attack, the modified scheme is significantly more vulnerable to a key recovery attack. In a key recovery attack we usually try to find a linear map $\mathcal{T} \in \mathbb{F}_{2^r}^{n\times n}$ such that $\mathcal{T}$ transforms the public key system into a UOV system. Given such a $\mathcal{T}$ an attacker can sign any document in the same way a legitimate user would. It is clear that in our modified scheme the attacker can restrict his search to $\mathcal{T} \in \mathbb{F}_2^{n\times n}$. Therefore a key recovery attack on our modified UOV scheme is equivalent to a key recovery attack on a regular UOV scheme over $\mathbb{F}_2$. In particular, the complexity of key recovery attacks is independent of r. We will investigate how this affects the feasibility of the UOV attack and the UOV Reconciliation attack.

4.2.3 UOV attack

The UOV attack attempts to recover an equivalent private key by searching for the oil subspace (see Sect. 3.5.2). This attack has a complexity of $q^{v-o-1} \cdot n^4$. Since a UOV attack on our scheme is equivalent to a UOV attack over $\mathbb{F}_2$, we have that the complexity of a UOV attack against our scheme is $2^{v-o-1} \cdot n^4$.

4.2.4 UOV Reconciliation attack

The UOV reconciliation attack attempts to recover a private key $(\mathcal{F}, \mathcal{T})$ with the matrix representation of $\mathcal{T}$ of the form

$$\begin{pmatrix} \mathbf{I}_{v\times v} & \mathbf{T}' \\ \mathbf{0}_{o\times v} & \mathbf{I}_{o\times o} \end{pmatrix}$$

by solving an overdetermined quadratic system. (see Sect. 3.5.3)

A lower bound to the complexity of the UOV reconciliation attack is given by the complexity of solving a random quadratic system of v variables and v equations over $\mathbb{F}_2$. When solving systems over $\mathbb{F}_2$ it is more efficient to do a smart exhaustive search instead of a Gröbner basis approach [9]. It is possible to do an exhaustive search for the

solutions of a system of quadratic equations in n variables with approximately $log_2(n)2^{n+2}$ bit operations (for a proof see [9]). This is independent of the number of equations. Using this approach a lower bound to the complexity of a UOV reconciliation attack is $\log_2(v)2^{v+2}$ bit operations.

4.3 Choice of parameters

For convenience and efficiency we will work with binary finite fields whose elements are represented by a number of bits that is a multiple of 16, that is the finite fields we want to use are $\mathbb{F}_{2^{16}}, \mathbb{F}_{2^{32}}, \mathbb{F}_{2^{48}}, \cdots$

When designing a signature scheme of security level ℓ, we choose a finite field that is large enough such that the minimal number of equations in a determined regular system that is needed to reach the security level ℓ is minimized. Figure 3.2 shows that for 128-bit and 256-bit security the chosen fields are $\mathbb{F}_{2^{48}}$ and $\mathbb{F}_{2^{80}}$ respectively, and the minimal number of equations is 34 and 66 respectively or 40 and 81 when considering quantum attacks. For 100-bit and 192-bit security the chosen fields are $\mathbb{F}_{2^{32}}$ and $\mathbb{F}_{2^{64}}$, and the minimal number of equations is 27 and 50 for classical attackers or 33 and 60 for quantum attackers.

We now consider the constraints on the parameters due to the different attacks against our scheme. In order to be safe against a direct attack we must have that

$$m - \lfloor v/m \rfloor \geq m_{min}\,,$$

with m_{min} equal to $27, 34, 50$ or 66 if the desired security level is 100 bits, 128 bits, 192 bits or 256 bits respectively. For quantum attackers m_{min} is equal to $33, 40, 60$ and 81 respectively. In order to be safe against the UOV attack we must have that

$$2^{v-o-1}n^4 > 2^\ell \quad \text{or} \quad 2^{(v-o-1)/2}n^4 > 2^\ell\,,$$

depending on whether we want ℓ bits of security against classical, or quantum adversaries. To be secure against the UOV reconciliation attack it suffices that an attacker cannot solve a determined system with v equations over $\mathbb{F}_2$. Therefore it suffices to have

$$\min(\log_2(v)2^{v+2}, 2^{52+v0.79}) > 2^\ell \quad \text{or} \quad 2^{v/2} > 2^\ell\,,$$

for classical and quantum attackers respectively. The parameter sets displayed in Table 4.2 satisfy all the constraints for the targeted security level and minimize the size of the public key, i.e. they minimize m.

4.3.1 Trade-off

In comparison to regular UOV, Lifted UOV has much smaller public keys, but also larger signatures. In the discussion above, we have chosen the parameter r very large in order to minimize the size of the public key, without regard for the size of the signatures. It is possible to make a trade-off between the size of the public key and the size of the signature by choosing a smaller value of r. Having a smaller value of r requires a larger value of m to reach the same security level, resulting in a larger public key, but since

Table 4.2: Parameter choices and corresponding public key and signature sizes for different security levels

security level		(r, m, v)	|pk| (kB)	|sig| (kB)	classical security
100 bits	classical	(32,30,105)	1.7	0.5	
	quantum	(32,37,200)	3.2	0.9	117-bit
128 bits	classical	(48,37,137)	3.2	1.0	
	quantum	(48,45,256)	5.7	1.8	153-bit
192 bits	classical	(64,54,215)	9.8	2.1	
	quantum	(64,65,384)	17.0	3.5	235-bit
256 bits	classical	(80,70,293)	21.2	3.5	
	quantum	(80,87,526)	40.7	6.0	312-bit

the signature consist of n elements of $\mathbb{F}_{2^r}$ it also leads to smaller signatures. Figure 7.1 compares public key sizes and signature sizes of the Lifted UOV scheme with different values of the parameter r with some other MQ signature schemes [28], the lattice-based signature scheme BLISS-II [17] and SPHINCS, a hash-based signature scheme [5].

Example. *For some application on a low-cost device it might be desirable to have a signature scheme that provides 128 bits of post-quantum security with minimal signature sizes subject to the condition that the public key is smaller than, say, 10 kB. If we choose the parameters as in the discussion above, we would have a public key of 5.7 kB and signatures of 1.8 kB. However, we can do better by choosing* $r = 12$. *The lowest values of* m *and* v *providing 128 bits of security are then* $m = 54$ *and* $v = 256$. *This leads to a public key of 9.8 kB (*< 10*kB) and a signature of 0.45 kB.*

4.4 Implementation and results

For implementing the arithmetic in large binary fields we have used an approach of [25]. All our fields are considered to be field extensions of $\mathbb{F}_{2^{16}}$ of low degree, so the elements are represented by low degree polynomials modulo some irreducible polynomial $f \in \mathbb{F}_{2^{16}}[x]$. Adding two field elements is just adding the coefficients of the polynomials, which is done with a cheap xor operation per coefficient. In order to multiply two field elements we have to multiply the polynomials and reduce modulo f, so we need to do a small number of additions and multiplications over $\mathbb{F}_{2^{16}}$. For multiplications over $\mathbb{F}_{2^{16}}$ we use a table with the logarithms of the field elements with respect to some generator, and a table that goes in the other direction. Multiplying $a, b \in \mathbb{F}_{2^{16}}$ is then calculated as $\log^{-1}(\log(a) + \log(b))$ with three table lookups and one addition modulo $2^{16} - 1$.

During the key generation algorithm we only work with elements of $\mathbb{F}_2$, so we can use bit slicing to do many field operations at the same time. This makes the key generation algorithm much more efficient. In fact, key generation is so efficient that we have chosen not to store the entire private key $(\mathcal{F}, \mathcal{T})$, which is typically quite big. Instead, we only store the seed that was used to generate the key pair. A part of the key generation algorithm is invoked at the beginning of the signature generation algorithm to generate the private key from this seed. (Only a part because we only need to generate the secret key, not the public key.) This approach gives roughly a factor 2 overhead for the signing algorithm, but it has the benefit that we do not have to store the large private key.

Table 4.3: Comparison of key and signature sizes of some signature schemes

security level		pk (B)	sk (B)	signature (B)	Quantum resistant
100 bits	**Lifted UOV**	1748	36	540	Yes
	lattisigns512 [21]	1536	256	1184	Yes
	UOVrand [28]	25702	187597	105	Yes
	RainbowLRS2 [28]	20685	46080	59	Yes
	RSA1536	192	1536	192	No
	ECDSA	56	84	56	No
128 bits	**Lifted UOV**	3256	36	1044	Yes
	BLISS-II [17]	7168	2048	5120	Yes
	SPHINCS [5]	1056	1088	41000	Yes
	UOVrand [28]	52531	390963	135	Yes
	RainbowLRS [28]	45568	104960	79	Yes
	RSA3072	384	3072	384	No
	ECDSA	64	96	64	No

The key and signature sizes of our signature scheme are compared to other signature schemes in Table 4.3. We see that the public key size and the signature size is significantly larger than the most widely used signatures (RSA and ECDSA), however these signature schemes are known to be vulnerable to attacks on quantum computers. In comparison to other post-quantum signature schemes we do very good in terms of key sizes and signature sizes. In comparison with other MQ signatures the public keys of the lifted UOV scheme are much smaller, but the signature sizes are larger.

The signature scheme was implemented in ANSI C. The running times of the different algorithms are reported in Table 4.4. Our signature scheme is a bit slower than other MQ signature schemes. One reason is that we work with a larger field, so the arithmetic takes more work. However, the main reason that our code is not as fast as other MQ signature schemes is probably that the code is poorly optimized. Apart from optimizing the code there is a number of things that can be done to speed up the algorithms. We have used naive implementations of matrix multiplication and field arithmetic, more advanced methods such as Karatsuba's algorithm could speed up the code. Large parts of the code can be parallelized without overhead. Newer CPUs support the CLMUL instruction set which could be used to perform the field arithmetic efficiently without table lookups. Moreover, it is possible to use a method of Petzoldt to structure part of the public key in such a way that the verification algorithm is faster [30]. In order to avoid storing the large private key, part of the key generation algorithm is run each time a signature is generated to generate the private key. If a batch of messages is signed together this step only has to happen once. Alternatively, if storing the private key is not an issue, this part can be omitted altogether to speed up the signing algorithm significantly.

Table 4.4: Running times for the key generation, signing and verification algorithms on a single thread on an Intel®Core™ i7-4710MQ CPU at 2.5 GHz

security level		key gen (ms)	sig gen (ms)	verification (ms)
100 bits	classical	3	4	2
	quantum	10	13	6
128 bits	classical	6	10	5
	quantum	22	30	14
192 bits	classical	22	35	16
	quantum	153	166	57
256 bits	classical	92	119	44
	quantum	395	443	156

4.5 Application to other MQ signature schemes

The idea of lifting keys to a large extension field can be applied to any MQ signature scheme, but it might not always we useful to produce smaller public keys. We believe that the idea could be used to reduce the size of the public keys of the Rainbow signature scheme, which is very similar to the UOV scheme. But for schemes such as HFE and C^* this seems unlikely. The reason is that the public keys of these signature schemes are not semi-regular maps [18] and have a much smaller degree of regularity than random maps of the same dimensions. This means that guessing a few variables does not necessarily reduce the degree of regularity significantly, unlike in the case of semi-regular systems were guessing even one variable can halve the degree of regularity. This makes the hybrid approach unsuitable for attacking these systems, since solving the system with one big Gröbner basis computation is likely to be more efficient. Therefore there is no use for lifting the system to a larger field, because the complexity of a Gröbner basis computation is largely independent of the size of the finite field.

Chapter 5

Reducing key sizes using Merkle trees

In this chapter we present a new signature scheme based on UOV that achieves tiny public keys at a cost of larger signatures. The combined sizes of signature and public key is reduced significantly. Moreover, we show that the new scheme is as secure as the original OUV scheme. We rely on techniques from hash based cryptography, most notably the concept of a Merkle Tree, which is explained in Sect. 2.6.

The contents of this chapter and the next is based on joint work with Alan Szepieniec. Our paper will be presented at the PQCrypto 2017 conference [33].

5.1 Description of the new scheme

The main idea behind the new algorithms is that it is not necessary to communicate the entire public map $\mathcal{P}$ to the verifier. Instead, it suffices to communicate $\mathcal{S} \circ \mathcal{P}$, where $\mathcal{S}$ is a randomly chosen linear map from $\mathbb{F}_q^m$ to $\mathbb{F}_q^\alpha$ with α much smaller than m. This is an improvement because the size of $\mathcal{S} \circ \mathcal{P}$ is smaller than $\mathcal{P}$ by a factor of $\frac{\alpha}{m}$. In the usual UOV scheme the verifier needs $\mathcal{P}$ in order to check if $\mathcal{P}(s) = \mathcal{H}(d)$. However $\mathcal{P}$ is not needed because it suffices to check if $\mathcal{S} \circ \mathcal{P}(s) = \mathcal{S} \circ \mathcal{H}(d)$. It is obvious that if the first equation is satisfied, then so is the second. Conversely, for a randomly chosen $\mathcal{S}$, if the first equality does not hold, then the probability that the second equality holds, is very small. The linear map $\mathcal{S}$ is chosen by casting the result of a hash function $\mathcal{H}_2$ evaluated at s to a α-by-m matrix. This makes it impossible to solve the system $\mathcal{S} \circ \mathcal{P}(s) = \mathcal{S} \circ \mathcal{H}(s)$ for s because $\mathcal{S}$ is only known after s is determined.

The probability that $\mathcal{S} \circ \mathcal{P}(s) = \mathcal{S} \circ \mathcal{H}(d)$ for an invalid signature (i.e. $\mathcal{P}(s) \neq \mathcal{H}(d)$) is $q^{-\alpha}$. Therefore we should take $\alpha = \lceil \log_q(2^\ell) \rceil$, such that $q^{-\alpha} < 2^{-\ell}$, where ℓ is the targeted security level. Each of the α components of $\mathcal{S} \circ \mathcal{P}$ takes $n(n+1)/2$ coefficients, so the total number of bits it takes to communicate $\mathcal{S} \circ \mathcal{P}$ is

$$\frac{n(n+1)}{2} \lceil \log_q(2^\ell) \rceil \log_2(q) \approx \frac{n(n+1)}{2} \ell \,.$$

This is independent of q, therefore we can choose the size of $\mathbb{F}_q$ really large. This has the benefit that with a smaller number of variables n we can reach the same security level (see Fig. 3.2). This decreases the size of the signatures even more.

There is a flaw in the scheme as we have described it until now. If the map $\mathcal{S} \circ \mathcal{P}$ is included in the signature, then nothing prevents an attacker from including some other map $\mathcal{R}$ in the signature such that $\mathcal{R}(s) = \mathcal{S} \circ \mathcal{H}(d)$. The verifier needs a way to check if the map $\mathcal{R}$ which is included in the signature is equal to $\mathcal{S} \circ \mathcal{P}$. The verifier can calculate $\mathcal{S}$, because it is derived deterministically from the document d and the signature s. However, the full system $\mathcal{P}$ is usually not known to the verifier, exactly because it is too expensive too communicate.

This problem is solved by interpreting the coefficients of the i-th component of $\mathcal{P}$ as the coefficients of a univariate polynomial $\hat{p}_i$ of degree $\frac{n(n+1)}{2}$. Similarly, we interpret the coefficients of i-th component of $\mathcal{R}$ as coefficients of the univariate polynomial $\hat{r}_i$ of degree $\frac{n(n+1)}{2}$. So we have the polynomials $\hat{p}_1, \cdots, \hat{p}_m, \hat{r}_1, \cdots, \hat{r}_\alpha \in \mathbb{F}_q[x]$. We can bundle these polynomials in two polynomial maps $\hat{\mathcal{P}} : \mathbb{F}_q \to \mathbb{F}_q^n$ and $\hat{\mathcal{R}} : \mathbb{F}_q \to \mathbb{F}_q^\alpha$. Obviously, we have $\mathcal{R} = \mathcal{S} \circ \mathcal{P}$ if and only if $\hat{\mathcal{R}} = \mathcal{S} \circ \hat{\mathcal{P}}$. The verifier can check if $\hat{\mathcal{R}} = \mathcal{S} \circ \hat{\mathcal{P}}$ by evaluating both sides at random values in $\mathbb{F}_q$ and checking equality. To make this possible we include a number of evaluations $v_1 = \hat{\mathcal{P}}(x_1), \cdots, v_\vartheta = \hat{\mathcal{P}}(x_\vartheta)$ in the signature, where $x_1, \cdots, x_\vartheta$ are deduced deterministically from d, s and $\mathcal{R}$. Then the verifier checks if $\hat{\mathcal{R}}(x_i)$ is equal to $\mathcal{S}(v_i)$ for i from 1 to ϑ.

So far we have merely shifted the problem, now we can be sure that the $\mathcal{R}$ which is included in the signature is really equal to $\mathcal{S} \circ \mathcal{P}$, but only if the $v_i \in \mathbb{F}_q^m$ which are included in the signature are equal to $\hat{\mathcal{P}}(x_i)$. So cheating is still possible. To prevent this the x_i are drawn from a reasonably large subset of $\mathbb{F}_q$ and for each possible x_i we validate the value $\hat{\mathcal{P}}(x_i)$ using a Merkle tree, the root of which is included in the public key.

5.1.1 Pseudocode

Now we describe the algorithm more formally. The parameters for the algorithm are

- q , the size of the finite field that is used
- m, the number of components of the UOV system
- $n = m + v$, the number of variables of the UOV system
- α, the number of components of $\mathcal{S}$
- κ, the length of the hash values in the Merkle tree
- τ, the number of leaves in the Merkle tree
- ϑ, the number of x_i's that is used to check if $\mathcal{R} = \mathcal{S} \circ \mathcal{P}$
- $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{F}_q^m$, the hash function used to calculate the digest of a message
- $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{F}_q^{\alpha \times m}$, the hash function used to deduce $\mathcal{S}$
- $\mathcal{H}_3 : \{0,1\}^* \to \{1, \cdots, \tau\}^\vartheta$, the hash used to determine which x_i are chosen.
- $\mathcal{H}_4 : \{0,1\}^* \to \{0,1\}^\kappa$, the hash function used in the Merkle tree.

Algorithm UOVHashGenerateKeys

input: Random bits to generate a UOV key pair

output: MerkleRoot — A public key
$(\mathcal{P}, (\mathcal{F}, \mathcal{T}))$ — A corresponding secret key

1: $(\mathcal{P}, (\mathcal{F}, \mathcal{T}))$ ←UOVGenerateKeys
2: **for** i from 1 to τ **do**
3: $\quad x_i \leftarrow i$-th element of $\mathbb{F}_q$ according to some arbitrary order
4: $\quad v_i \leftarrow \hat{\mathcal{P}}(x_i)$
5: **end for**
6: MerkleRoot ← CalculateMerkleRoot$(v_1, \cdots, v_\tau)$
7: **return** (MerkleRoot , $(\mathcal{P}, (\mathcal{F}, \mathcal{T}))$)

Alg. 5.1: The key generation algorithm

Algorithm UOVHashSign

input: M — A message to sign
$(\mathcal{P}, (\mathcal{F}, \mathcal{T}))$ — A private key

output: $(s, \mathcal{R}, v_{a_1}, \cdots, v_{a_\vartheta}$,MerklePaths) — A signature for M

1: s ←UOVSign$(M, \mathcal{F}, \mathcal{T})$
2: $\mathcal{S} \leftarrow \mathcal{H}_2(M||s)$
3: $\mathcal{R} \leftarrow \mathcal{S} \circ \mathcal{P}$
4: **for** i from 1 to τ **do**
5: $\quad x_i \leftarrow i$-th element of $\mathbb{F}_q$ according to some arbitrary order
6: $\quad v_i \leftarrow \hat{\mathcal{P}}(x_i)$
7: **end for**
8: $a_1, \cdots, a_\vartheta \leftarrow \mathcal{H}_3(M||s||\mathcal{R})$
9: MerklePaths ← empty list
10: **for** a_i from a_1 to a_ϑ **do**
11: $\quad$ MerklePaths$[i]$ ←OpenMerklePath$(v_1, \cdots, v_\tau, a_i)$
12: **end for**
13: **return** $(s, \mathcal{R}, v_{a_1}, \cdots, v_{a_\vartheta}$,MerklePaths)

Alg. 5.2: The signature generation algorithm

Algorithm UOVHashVerify

input: M — A message
$(s, \mathcal{R}, v_{a_1}, \cdots, v_{a_\vartheta}$,MerklePaths) — A candidate-signature for M
MerkleRoot — A public key

output: **True** if s is a valid signature for M, **False** otherwise

1: $\mathcal{S} \leftarrow \mathcal{H}_2(M||s)$
2: **if** $\mathcal{R}(s) \neq \mathcal{S} \circ \mathcal{H}_1(M)$ **then**
3: | **return False**
4: **end if**
5: $a_1, \cdots, a_\vartheta \leftarrow \mathcal{H}_3(M||s||\mathcal{R})$
6: **for** i from 1 to ϑ **do**
7: | **if** $\hat{\mathcal{R}}(x_{a_i}) \neq \mathcal{S}(v_{a_i})$ **then**
8: | | **return False**
9: | **end if**
10: | **if** OpenMerklePath(a_i, v_{a_i},MerklePaths[i],MerkleRoot) fails **then**
11: | | **return False**
12: | **end if**
13: **end for**
14: **return True**

Alg. 5.3: The signature verification algorithm

5.2 Security analysis of the new scheme

We are able to give a tight reduction of the EUF-CMA game of the new scheme to the EUF-CMA game of the original UOV signature scheme in the Quantum Random Oracle Model. This means that if there is an efficient EUF-CMA adversary against the new signature scheme, this can be converted in an equally efficient EUF-CMA adversary against the original UOV signature scheme. In other words, the new signature scheme is at least as secure as the UOV signature scheme.

This is proven using a sequence of games, which is a common proof strategy [32]. The initial game of the sequence is the EUF-CMA game for the new signature scheme, the last game of the sequence is the EUF-CMA game for the original signature scheme. Then we prove that between any two games in the sequence the probability of winning can only decrease by a tiny amount. This proves that if an adversary wins the first game with a large probability, there is an adversary which wins the last game with a large probability. The contrapositive of this statement says that if the original signature scheme is secure, then the new signature scheme is secure as well. Before proving the reduction, we introduce a few security properties of hash function families.

5.2.1 Multiple-target second-preimage resistance

The first property is called single-function multiple-target second-preimage resistance (SM-SPR) and is introduced by Hülsing et al. [22]. The problem is to, given a hash function $\mathcal{H}$ and a set of p messages $M_1, \cdots, M_p$ in the domain of $\mathcal{H}$, find a new message M such that for some i we have $M_i \neq M$ and $\mathcal{H}(M_i) = \mathcal{H}(M)$. This property is a natural generalization of the second preimage resistance property to multiple messages at once. Let $\text{InSec}^{\text{SM-SPR}}_{\mathcal{H},p}(Q)$ be the insecurity function of the SM-SPR property with p targets. For an adversary in the random oracle model there is no better strategy than to just evaluate the hash function at random inputs and hope that the output matches the output of one of the messages. Therefore we have $\text{InSec}^{\text{SM-SPR}}_{\mathcal{H},p}(Q) = (Q+1)\frac{p}{2^n}$. Hülsing et al. prove that in the quantum random oracle model we have $\text{InSec}^{\text{SM-SPR}}_{\mathcal{H},p}(Q) = \Theta\left(\frac{p(Q+1)^2}{2^n}\right)$, where the constant hidden in the Θ-notation is small [22].

Algorithm SM-SPR

input: $\mathcal{H}$ — A hash function family $\mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$
p — The number of targets
A — An adversary

output: **Win** / **Lose** — Whether A wins or loses the game

```
1: K ← 𝒦
2: for i from 1 to p do
3: |  M_i ←$ {0,1}^m
4: end for
5: M ← A(K, M_1, ··· , M_p)
6: for i from 1 to p do
7: |  if H_K(M) = H_K(M_i) and M ≠ M_i then
8: |  |  return Win
9: |  end if
10: end for
11: return Lose
```

Alg. 5.4: The game associated to the SM-SPR property

5.2.2 Multiple-target second-preimage one-wayness

A different security property is called single-function multiple-target one-wayness (SM-OW). This property is the generalization of the one-wayness property to multiple targets. Given a number of targets $x_1, \cdots, x_p$ in the range of a hash function $\mathcal{H}$, the problem is to find an M in the domain of the hash function such that $\mathcal{H}(M) = x_i$ for some i. It is shown that the bounds on the security function of SM-SPR also apply to the insecurity function of SM-OW, in the random oracle model and in the quantum random oracle model. Specifically, in the random oracle model we have $\text{InSec}^{\text{SM-OW}}_{\mathcal{H},p}(Q) = (Q+1)\frac{p}{2^n}$ and in the quantum random oracle model we have $\text{InSec}^{\text{SM-OW}}_{\mathcal{H},p}(Q) = \Theta\left(\frac{p(Q+1)^2}{2^n}\right)$, where the

constant hidden in the Θ-notation is small [22].

5.2.3 Multiple moving targets One-wayness

We define yet another security property called multiple moving targets one-wayness (MMT-OW). For this property the problem is to, given a computable target function $T : \{0,1\}^m \to P(\{0,1\}^n)$, (here $P(X)$ denotes the power set set of X) find an M such that $\mathcal{H}(M) \in T(M)$. This property is a generalization of the SM-OW property, since the latter corresponds to the case where T is the constant function $T : \{0,1\}^m \to P(\{0,1\}^n) : M \mapsto \{x_i\}_{i\in\{1,\cdots,p\}}$. In the (quantum) random oracle model, if the target sets are sufficiently small, then the InSec function of the MMT-OW property is also small:

Lemma 2. *Fix $T : \{0,1\}^m \to P(\{0,1\}^n)$ to be a computable target function and p an integer such that $|T(M)| < p$ for all M in $\{0,1\}^m$, then in the (quantum) random oracle model we have*

$$InSec_T^{MMT-OW}(Q) \leq InSec_p^{SM-OW}(Q)\,.$$

Proof. We will make a reduction to the SM-OW game. Suppose A is an adversary that plays the MMT-OW game and makes up to Q queries to the random oracle. We will show how to transform this into an adversary B^A that wins the SM-OW game with a probability at least al high as the probability that A wins and makes the same number of queries to the random oracle as A.

The adversary B is given a random oracle $\mathcal{H}$ and targets $x_1, \cdots, x_p$. Define $\mathcal{H}' : \{0,1\}^m \to \{0,1\}^n$ as

$$\mathcal{H}'(x) = \sigma_x^{-1}(\mathcal{H}(x))\,,$$

where $\sigma_x : \{0,1\}^n \to \{0,1\}^n$ is lexicographically the first permutation that maps $T(x)$ into $\{x_1, \cdots, x_p\}$. Note that such a permutation exists, since $|T(x)| < p$. Now the adversary B invokes A on the MMT-OW game with random oracle H' and target function T. After making some queries to $\mathcal{H}'$ the adversary A returns a message M, and B terminates his part of the SM-OW game by returning this message.
It is clear that the adversary B^A makes exactly one query to $\mathcal{H}$ each time A queries $\mathcal{H}'$. Moreover if A wins his MMT-OW game, that means that $\mathcal{H}'(m) \in T(m)$. This is equivalent to $\sigma_m^{-1}(\mathcal{H}(m)) \in T(m)$, or $\mathcal{H}(m) \in \sigma_m(T(m)) \subset \{x_1, \cdots, x_p\}$, so B^A wins the SM-OW game if A wins the MMT-OW game.

□

Algorithm MMT-OW

input: $\mathcal{H}$ — A hash function family $\mathcal{K} \times \{0,1\}^m \to \{0,1\}^n$
T — The target function
A — An adversary

output: **Win** / **Lose** — Whether A wins or loses the game

1: $K \xleftarrow{\$} \mathcal{K}$
2: $M \leftarrow A^{\mathcal{H}}(K)$
3: **if** $\mathcal{H}_K(M) \in T(M)$ **then**
4: | **return Win**
5: **else**
6: | **return Lose**
7: **end if**

Alg. 5.5: The game associated to the MMT-OW property

5.2.4 Reduction to the original UOV scheme

Theorem 3. *Let ORIGINAL be an MQ signature scheme whose public key has n variables, and let NEW be the transformed signature scheme, then in the (quantum) random oracle model we have*

$$\begin{aligned} InSec^{EUF\text{-}CMA}_{NEW}(t,Q) \leq & InSec^{EUF\text{-}CMA}_{ORIGINAL}(t+O(Q),Q) \\ & + InSec^{SM\text{-}SPR}_{\mathcal{H}_4,2\tau-1}(Q) \\ & + InSec^{SM\text{-}OW}_{\mathcal{H}_3,(n(n+1)/2)^{\vartheta}}(Q) \\ & + InSec^{SM\text{-}OW}_{\mathcal{H}_2,q^{\alpha\times(m-1)}}(Q)\,. \end{aligned}$$

Proof.

- $\mathbf{Game}_1$ is the EUF-CMA game of the new signature scheme.
- $\mathbf{Game}_2$ is similar to $\mathbf{Game}_1$, the difference being that in $\mathbf{Game}_2$ there is no Merkle tree. Instead of the root of the Merkle tree, the public key contains all the leaves $v_1, \cdots, v_\tau$. In the signature generation algorithm, no Merkle paths are generated or included in the signature. In the signature verification algorithm, the validity of the $v_{a_1}, \cdots, v_{a_\vartheta}$ is checked directly by comparing them with the v_i in the public key, instead of by verifying the Merkle paths.

  Given an adversary A that plays $\mathbf{Game}_1$ we define an adversary A' that plays $\mathbf{Game}_2$ as follows: When A' receives the public key $v_1, \cdots, v_\tau$ it calculates the root of the Merkle tree with the v_i's as leaves, and sends it as the public key of $\mathbf{Game}_1$ to A. Whenever A requests a message m to be signed, A' sends this message on to the challenger, who returns with a signature s for m, then A' calculates the Merkle paths for the v_{a_i} included in s, and appends them to the signature before sending it on to A. Eventually, when A responds with a message signature pair (m, s) A' simple removes the Merkle paths from the signature s to get s', and sends the modified

pair (m, s') on to the challenger.

Now, we prove that A' wins $\mathbf{Game}_2$ with a sufficiently large probability. If the pair (m, s) wins $\mathbf{Game}_1$, then (m, s') wins game $\mathbf{Game}_2$, unless one of the v_{a_i} included in the signature is not equal to the v_{a_i} in the public key. However, if (m, s) wins $\mathbf{Game}_1$, this implies that the Merkle path included in s is valid, so v_{a_i} can only be different from the v_{a_i} included in the public key if A has forged a Merkle path for a different a_i. This requires finding a second preimage for one of the $2\tau - 1$ values in the Merkle tree. Therefore, the probability that A' wins $\mathbf{Game}_2$ is at least equal to the probability that A wins $\mathbf{Game}_1$ minus $\mathrm{InSec}^{\text{SM-SPR}}_{\mathcal{H},2\tau-1}(Q)$.

- $\mathbf{Game}_3$ differs from $\mathbf{Game}_2$ in that the v_i's in the public key are replaced by $\mathcal{P}$. In the key generation and signature generation algorithms, the v_i no longer have to be calculated or included in the signature. In the verification algorithm, instead of verifying whether $\mathcal{R} = \mathcal{S} \circ \mathcal{P}$ through the v_{a_i}, this is done directly using the $\mathcal{P}$ which is included in the public key.

Given an adversary A that plays $\mathbf{Game}_2$, we define an adversary A' which plays $\mathbf{Game}_3$ as follows: When A' receives $\mathcal{P}$, the public key of $\mathbf{Game}_3$, A' uses this to calculate all the $v_1, \cdots, v_\tau$ and sends them as a public key of $\mathbf{Game}_2$ to A. When A requests a message m to be signed, A' passes it on to the challenger of $\mathbf{Game}_3$, who replies with a signature s. Then A' calculates $\mathcal{S}$ and $\mathcal{R}$, and evaluates $\mathcal{H}_3(m||s||\mathcal{R})$ to generate $a_1, \cdots, a_\vartheta$. Then A' computes the $v_{a_1}, \cdots, v_{a_\vartheta}$ and includes them in s to get a valid signature for m in $\mathbf{Game}_2$, which he can then send to A. Eventually, when A produces a message-signature pair (m, s) A' removes the v_{a_i} from s to get a signature s' for m in $\mathbf{Game}_3$. Then A' sends the pair (m, s') to the challenger of $\mathbf{Game}_3$.

Now we prove that A' wins $\mathbf{Game}_3$ with a sufficiently high probability. We show that the probability that A' returns a pair (m,s) that wins $\mathbf{Game}_2$, but such that (m,s') does not win $\mathbf{Game}_3$ is bounded by $\mathrm{InSec}^{SM-OW}_{\mathcal{H}_3,(n(n+1)/2)^\vartheta}(Q)$. In order for (m, s) to win $\mathbf{Game}_2$ it is required that $\hat{\mathcal{R}}(x_{a_i}) = \mathcal{S}(v_{a_i})$ for all $a_i \leftarrow \mathcal{H}_3(M||s||R)$. In order for (m, s') not to win $\mathbf{Game}_3$ we require that $\mathcal{R} \neq \mathcal{S} \circ \mathcal{P}$ or equivalently that $\hat{\mathcal{R}} \neq \mathcal{S} \circ \hat{\mathcal{P}}$. Finding m, s and $\mathcal{R}$ that satisfy these conditions can be formulated as a multiple moving target one-wayness problem for the $\mathcal{H}_3$ hash function with the target function $T : \{0,1\}^\star \times \{0,1\}^{\log_2(q)\times n} \times \{0,1\}^{\log_2(q)\times\alpha\times(n(n+1)/2)} \to P(\{0,1\}^{\log_2(\tau)\times\vartheta})$ defined as

$$T(m, s, \mathcal{R}) = \begin{cases} \emptyset & \text{if } \mathcal{R} = \mathcal{S} \circ \mathcal{P} \\ \{i \in \{1, \cdots, \tau\} | \hat{\mathcal{R}}(x_i) = \mathcal{S}(v_i)\}^\vartheta & \text{else} \end{cases}.$$

If $\mathcal{R} \neq \mathcal{S}\circ\mathcal{P}$ then $\hat{\mathcal{R}}-\mathcal{S}\circ\hat{\mathcal{P}}$ is a set of m univariate polynomials of degree $\leq n(n+1)/2$, and at least one of these polynomials is nonzero. Therefore the solution set of these polynomials contains at most $n(n + 1)/2$ solutions and

$$|T(m, s, \mathcal{R})| \leq \left(\frac{n(n + 1)}{2}\right)^\vartheta$$

for all m, s and $\mathcal{R}$. Lemma 2 states that the probability that A wins $\mathbf{Game}_2$ but A' does not win $\mathbf{Game}_3$ is bounded by $\mathrm{InSec}^{SM-OW}_{\mathcal{H}_3,(n(n+1)/2)^\vartheta}(Q)$.

- The last game of the sequence, $\mathbf{Game}_4$, is the EUF-CMA game of the original signature scheme. The differences with $\mathbf{Game}_3$ are that $\mathbf{Game}_3$ includes $\mathcal{R}$ in the signature whereas $\mathbf{Game}_4$ does not and that in the signature verification algorithm of $\mathbf{Game}_3$ it is verified that $\mathcal{R}(s) = \mathcal{S} \circ \mathcal{H}_1(d)$ and $\mathcal{R} = \mathcal{S} \circ \mathcal{P}$, whereas in the verification algorithm of $\mathbf{Game}_4$ only $\mathcal{P}(s) = \mathcal{H}_1(d)$ is verified.

  Given an adversary A that plays $\mathbf{Game}_3$, we construct an adversary A' that plays $\mathbf{Game}_4$ as follows: When A' receives the public key $\mathcal{P}$, he sends it on to A. When A requests a message m be signed, A' passes it on to the challenger of $\mathbf{Game}_4$ and receives a signature s from m. Then A' calculates $S = \mathcal{H}_2(m||s)$ and $\mathcal{R} = \mathcal{S} \circ \mathcal{P}$, includes $\mathcal{R}$ in the signature s to get a valid signature for $\mathbf{Game}_3$, and A' sends this signature to A. Eventually, when A returns a message-signature pair (m, s), A' removes $\mathcal{R}$ from s to get a signature s' for $\mathbf{Game}_4$. Then A' sends (m, s') to the challenger of $\mathbf{Game}_4$.

  Now we prove that A' wins $\mathbf{Game}_4$ with a sufficiently large probability. We show that the probability that the pair (m, s) wins $\mathbf{Game}_3$ but (m, s') does not win $\mathbf{Game}_4$ is bounded by $\mathrm{InSec}^{SM-OW}_{\mathcal{H}_2,q^{\alpha\times(m-1)}}$. If (m, s') does not win $\mathbf{Game}_4$ this means that $\mathcal{P}(s') - \mathcal{H}_1(m)$ is a nonzero vector. However, if (m, s) wins $\mathbf{Game}_3$ then $\mathcal{S} \circ \mathcal{P}(s') = \mathcal{S} \circ \mathcal{H}(m)$, so $\mathcal{P}(s') - \mathcal{H}_1(m)$ sits in the kernel of $\mathcal{S} \leftarrow \mathcal{H}_2(m||s')$. Finding m and s' that satisfy these conditions can be formulated as a multiple moving targets one-wayness problem for the hash function $\mathcal{H}_2$ and the target function $T : \{0,1\}^\star \times \{0,1\}^{\log_1(q)\times n} \to \{0,1\}^{\log_2(q)\times\alpha\times m}$ defined by

$$T(m, s') = \begin{cases} \emptyset & \text{if } \mathcal{P}(s') = \mathcal{H}(m) \\ \{\mathcal{S} \in \mathbb{F}_q^{\alpha\times m} | \mathcal{S}(\mathcal{P}(s') - \mathcal{H}_1(m)) = 0\} & \text{else} \end{cases}.$$

  For any choice of m and s' the set $T(m, s')$ is either empty, or the set of α-by-m matrices that maps a certain vector of length m to $\mathbf{0}$, so we have $|T(m, s')| \leq q^{\alpha\times(m-1)}$. Lemma 2 now says that the probability that A wins $\mathbf{Game}_3$ but A' does not win $\mathbf{Game}_4$ is bounded by $\mathrm{InSec}^{SM-OW}_{\mathcal{H}_2,q^{\alpha\times(m-1)}}(Q)$.

□

Corollary 3. *Let ORIGINAL be an MQ signature scheme whose public key has n variables, and let NEW be the transformed signature scheme, then in the* ***random oracle model*** *we have*

$$\begin{aligned} InSec^{EUF\text{-}CMA}_{NEW}(t, Q) \leq & InSec^{EUF\text{-}CMA}_{ORIGINAL}(t + O(Q), Q) \\ & + (2\tau - 1)\frac{Q+1}{2^\kappa} \\ & + \left(\frac{n(n+1)}{2\tau}\right)^\vartheta (Q+1) \\ & + q^{-\alpha}(Q+1)\,. \end{aligned}$$

Proof. This follows straightforwardly from Theorem 3, since in the random oracle model we have for hash functions $\mathcal{H}$ with range $\{0,1\}^n$

$$\mathrm{InSec}^{\text{SM-OW}}_{\mathcal{H},p}(Q) = \mathrm{InSec}^{\text{SM-SPR}}_{\mathcal{H},p}(Q) = p\frac{Q+1}{2^n},$$

and the ranges of $\mathcal{H}_4, \mathcal{H}_3$ and $\mathcal{H}_2$ are $\{0,1\}^\kappa, \{0,1\}^{\log_2(\tau)\times\vartheta}$ and $\{0,1\}^{\log_2(q)\times\alpha\times m}$ respectively. □

Corollary 4. *Let ORIGINAL be an MQ signature scheme whose public key has n variables, and let NEW be the transformed signature scheme, then in the* ***quantum random oracle model*** *we have*

$$\begin{aligned} InSec^{EUF\text{-}CMA}_{NEW}(t,Q) \leq & InSec^{EUF\text{-}CMA}_{ORIGINAL}(t+O(Q),Q) \\ & + \Theta\left((2\tau-1)\frac{(Q+1)^2}{2^\kappa}\right) \\ & + \Theta\left(\left(\frac{n(n+1)}{2\tau}\right)^\vartheta (Q+1)^2\right) \\ & + \Theta\left(q^{-\alpha}(Q+1)^2\right). \end{aligned}$$

Proof. This follows straightforwardly from Theorem 3, since in the quantum random oracle model we have, against quantum adversaries for hash functions $\mathcal{H}$ with range $\{0,1\}^n$

$$\mathrm{InSec}^{\text{SM-OW}}_{\mathcal{H},p}(Q) = \mathrm{InSec}^{\text{SM-SPR}}_{\mathcal{H},p}(Q) = \Theta\left(p\frac{(Q+1)^2}{2^n}\right),$$

and the ranges of $\mathcal{H}_4, \mathcal{H}_3$ and $\mathcal{H}_2$ are $\{0,1\}^\kappa, \{0,1\}^{\log_2(\tau)\times\vartheta}$ and $\{0,1\}^{\log_2(q)\times\alpha\times m}$ respectively. □

5.3 Small improvements

The improvements of the UOV signature scheme that are discussed in Sects. 3.7 and 3.8 can be applied to the new signature scheme as well. If we use the construction of Petzoldt (see Sect. 3.7) the verifier can generate the first part of $\mathcal{P}$. This means that we only have to put the last $m(m+1)/2$ coefficients of each component of $\mathcal{R}$ in the signature. Moreover, only the last $m(m+1)/2$ coefficients of each component of $\mathcal{R}$ have to be verified using the Merkle tree construction, which is more efficient. Moreover, the factor $\left(\frac{n(n+1)}{2\tau}\right)^\vartheta$ in Corollaries 3 and 4 can be replaced by $\left(\frac{m(m+1)}{2\tau}\right)^\vartheta$, so we can have smaller values of τ or ϑ.

The method of Sect. 3.8 that chooses $\mathcal{T}$ of some special form in order to speed up the algorithm applies without any problems. However the runtime of the key generation and signing algorithms is dominated by the building of the Merkle tree, so using this method only makes a relatively small impact.

5.4 Choice of parameters

In light of Corollary 3, it is clear that in order to design a signature scheme which is secure against classical adversaries it suffices to pick parameters such that the UOV signature scheme is secure against classical adversaries, and with α, τ, ϑ and κ large enough such that the three extra terms are very small. Similarly, in order to design a signature scheme which is secure against quantum adversaries it suffices to pick the parameters of the UOV signature scheme such that UOV is secure against quantum adversaries, and the other parameters large enough such that the terms in Corollary 4 are very small.

For convenience and efficiency we will work with binary finite fields whose elements are represented by a number of bits that is a multiple of 16, that is the finite fields we want to use are $\mathbb{F}_{2^{16}}, \mathbb{F}_{2^{32}}, \mathbb{F}_{2^{48}}$ and so on.

When designing a signature scheme of security level ℓ, we choose a finite field that is large enough such that the minimal number of equations in a determined regular system that is needed to reach the security level ℓ is minimized. Figure 3.2 shows that for 128-bit and 256-bit security the chosen fields are $\mathbb{F}_{2^{48}}$ and $\mathbb{F}_{2^{80}}$ respectively, and the minimal number of equations is 34 and 66 respectively or 40 and 81 when considering quantum attacks. For 100-bit and 192-bit security the chosen fields are $\mathbb{F}_{2^{32}}$ and $\mathbb{F}_{2^{64}}$, and the minimal number of equations is 27 and 50 for classical attackers or 33 and 60 for quantum attackers.

First, we consider the constraints on the parameters due to the different attacks against the original UOV scheme. In order to be safe against a direct attack we must have that

$$m - \lfloor v/m \rfloor \geq m_{min}\,,$$

with m_{min} equal to $27, 34, 50$ or 66 if the desired security level is 100 bits, 128 bits, 192 bits or 256 bits respectively. For quantum attackers m_{min} is equal to $33, 40, 60$ and 81 respectively. In order to be safe against the UOV attack we must have that

$$q^{v-o-1}n^4 > 2^\ell \quad \text{or} \quad q^{(v-o-1)/2}n^4 > 2^\ell\,,$$

depending on whether we want ℓ bits of security against classical, or quantum adversaries. It is customary to take $v = 2m$ in order to defend against these attacks [23]. Since we are working over large finite fields (i.e. q is large) it seems that such a large number of vinegar variables in not necessary in this case, however we will stick to this convention because this choice only has a relatively small effect on the speed and signature size of the signature scheme. To be secure against the UOV reconciliation attack it suffices that an attacker cannot solve a determined system with v equations over $\mathbb{F}_q$. Therefore it suffices to have $v > m_{min}$, which is automatically satisfied if $v = 2m$.

The UOV signature scheme is believed to be secure if the above constraints are satisfied, so we assume that the term $\mathrm{InSec}^{\text{EUF-CMA}}_{ORIGINAL}(t + O(Q), Q)$ in Corollaries 3 and 4 are sufficiently small. To make the other terms small, we choose the values of α, τ, ϑ and κ sufficiently large. To prevent an attacker from forging Merkle paths we choose $\kappa = \ell$ in the case of classical security, or $\kappa = 2\ell$ in the case of quantum security. To prevent a classical or quantum attacker from finding a $\mathcal{R} \neq \mathcal{S} \circ \mathcal{P}$ that is accepted by the verifier

Table 5.1: Parameter choices and corresponding signature sizes for different security levels. All parameter sets have $\tau = 2^{16}$. The size of the public key is equal to κ bits.

security level		(r, m, v)	$(\kappa, \vartheta, \alpha)$	|sig| (kB)	classical security
100 bits	classical	(32,29,58)	$(100, 14, 3)$	10	
	quantum	(32,35,70)	$(200, 28, 7)$	33	117-bit
128 bits	classical	(48,36,72)	$(128, 20, 3)$	22	
	quantum	(48,42,84)	$(256, 39, 6)$	62	143-bit
192 bits	classical	(64,52,104)	$(192, 35, 3)$	61	
	quantum	(64,62,124)	$(384, 69, 6)$	178	224-bit
256 bits	classical	(80,68,136)	$(256, 54, 4)$	156	
	quantum	(80,83,166)	$(512, 107, 7)$	434	296-bit

we pick τ and ϑ such that

$$\left(\frac{m(m+1)}{2\tau}\right)^{\vartheta} \leq 2^{-\ell} \text{ or } 2^{-2\ell},$$

respectively. Larger values of τ allow for smaller values of ϑ and smaller signatures, but result in a larger Merkle tree and thus a slower algorithm. We have chosen to take $\tau = 2^{16}$ and choose ϑ high enough to satisfy the inequality. Lastly, to prevent an attacker to forge a signature by brute force we take α such that $q^{-\alpha} \leq 2^{-\ell}$ or $2^{-2\ell}$ in the classical or quantum case respectively. The parameter sets displayed in Table 5.1 satisfy all the constraints for the targeted security level.

In our implementation we have chosen the Keccak hash function for all required hash functions $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and $\mathcal{H}_4$. An advantage of this hash function is that the size of the output is flexible. Also, some parameter choices of keccak are standardized by NIST, so it is natural to choose this hash function when designing a signature scheme that we want to submitted to NIST.

5.5 Implementation and results

We have implemented the algorithm in ANSI C. We have used the same lookup table-method for the arithmetic in finite fields as in the implementation of the Lifted UOV algorithm as explained in Sect. 4.4. Table 5.2 shows that the key generation and signature generation algorithms are quite slow. This is because during the key generation algorithm a Merkle tree with τ leaves has to be built and to obtain each leaf a univariate polynomial of degree $m(m+1)/2$ has to be evaluated. This means that in total we need roughly $\tau m^2/2$ multiplications in the finite field which makes key generation quite slow. In our implementation of the signature generation algorithm 2ϑ subtrees of the merkle tree with $\sqrt{\tau}$ leaves have to be built, which is also quite slow. Both algorithms can be sped up by choosing a smaller value of τ, but this results in larger signatures. The algorithm is very suitable for parallelization, which could be used to speed up the process without any significant overhead.

Table 5.2: Running times for the key generation, signing and verification algorithms on a single thread on an Intel®Core™ i7-4710MQ CPU at 2.5 GHz

security level		key gen (s)	sig gen (s)	verification (ms)
100 bits	classical	14.5	1.1	20
	quantum	23.8	3.2	42
128 bits	classical	32.7	3.6	40
	quantum	46.6	9.0	81
192 bits	classical	114	21.1	121
	quantum	185	59.9	256
256 bits	classical	286	84.3	366
	quantum	520	257	868

Chapter 6

Multiple Signatures for each message

In the previous chapter we developed a signature scheme that is provably as secure as UOV, but with a significant reduction in |pk| + |s|. In this chapter we will make a small change to the signature scheme to reduce |pk| + |s| even more. The signature scheme of the previous chapter has a parameter α, which has to be high enough to protect against a brute force attack. In this chapter we take α very small (e.g. $\alpha = 1$) and we protect against a brute force attack by producing multiple signatures for each message. The resulting signature scheme is no longer provably as secure as the original UOV scheme, but relies on the hardness of a natural generalization of the MQ problem, which we call the AMQ problem (Approximate Multivariate Quadratics).

6.1 Description of the new scheme

The new signature scheme is identical to the signature scheme described in the previous chapter except that there are σ signatures for each message instead of just one. This means that a signature for a message now looks like $(s_1, \cdots, s_\sigma, \mathcal{R}, v_{a_1}, \cdots, v_{a_\vartheta}$,MerklePaths) instead of $(s, \mathcal{R}, v_{a_1}, \cdots, v_{a_\vartheta}$,MerklePaths). Each s_i is a UOV signature for the message $M||i$ for i ranging from 1 to σ. The key generation algorithm has not changed, the signature generation algorithm now makes σ UOV signatures instead of just one. Also, $\mathcal{S}$ and the a_i are determined by $\mathcal{H}_2(M||s_1||\cdots||s_\sigma)$ and $\mathcal{H}_3(M||s_1||\cdots||s_\sigma||\mathcal{R})$ respectively. It is important that all the s_i are hashed into $\mathcal{S}$ and the a_i's because otherwise all the σ signatures could be brute forced independently. Lastly, the verification algorithm checks if $\mathcal{S} \circ \mathcal{P}(s_1) = \mathcal{S} \circ \mathcal{H}(M||i)$ for all i.

In the case $\sigma = 1$ the signature scheme is identical to the signature scheme of the previous chapter. For $\sigma > 1$ we hope that we can pick a smaller value of α than in the $\sigma = 1$ case. The signature contains α large quadratic polynomials, so reducing α is well worth including a few extra signatures, because the signatures are very small compared to the size of a quadratic polynomial in n variables.

6.2 Security Analysis of the new scheme

The security proof of the previous chapter still applies to the new signature scheme. That means that if $q^\alpha > 2^l$ the signature scheme is still as secure as UOV, given that κ, τ and ϑ are sufficiently large. However, a brute force attack against the new signature scheme has complexity $O(q^{\sigma\alpha})$ and we would want to reduce α by having multiple signatures (i.e. $\sigma > 1$). One could hope to adapt the security proof of the previous chapter in such a way to have the factor $q^{-\sigma\alpha}$ instead of $q^{-\alpha}$ in Corollaries 3 and 4, but sadly this is not possible.

In the security proof for the case $\sigma = 1$ we had that the probability that $\mathcal{S} \circ \mathcal{P}(s) = \mathcal{S} \circ \mathcal{H}(M)$ for a random linear map $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^\alpha$ is either 1 in the case that $\mathcal{P}(s) = \mathcal{H}(M)$ or $q^{-\alpha}$ otherwise. This means that an attacker has either to find a solution to $\mathcal{P}(s) = \mathcal{H}(M)$ or be extremely lucky to forge a signature. In the $\sigma > 1$ case this remains true for any of the individual signatures, but the probabilities of the events $\mathcal{S} \circ \mathcal{P}(s_i) = \mathcal{S} \circ \mathcal{H}(M||i)$ for the different i from 1 to σ are not independent, so if for all i we have $\mathcal{P}(s_i) \neq \mathcal{H}(M||i)$ then the probability that $\mathcal{S} \circ \mathcal{P}(s_i) = \mathcal{S} \circ \mathcal{H}(M||i)$ for all i is not necessarily equal to $q^{-\sigma\alpha}$.

The event that for all i we have $\mathcal{S} \circ \mathcal{P}(s_i) = \mathcal{S} \circ \mathcal{H}(M||i)$ happens if and only if all $\mathcal{P}(s_i) - \mathcal{H}(M||i)$ lie in the kernel of $\mathcal{S}$. Therefore, if d is the dimension of the subspace $\langle \mathcal{P}(s_1) - \mathcal{H}(M||1), \cdots \mathcal{P}(s_\sigma) - \mathcal{H}(M||\sigma) \rangle$ then the probability that for all i we have $\mathcal{S} \circ \mathcal{P}(s_i) = \mathcal{S} \circ \mathcal{H}(M||i)$ is equal to $q^{-d\alpha}$. Therefore, an attacker has to produce s_i such that the errors lie in a low dimensional space. We call this problem the Approximate Multivariate Quadratics problem.

6.2.1 AMQ Problem

While in the MQ problem the objective is to solve a quadratic system of equations exactly, the objective in the approximate MQ problem is to solve a quadratic system for a bunch of target vectors approximately. By this we mean that it is allowed to make errors, provided that the errors live in a subspace of $\mathbb{F}_q^m$ of small dimension. A more formal definition of the Approximate Multivariate Quadratics problem goes like this:

AMQ Problem. Given a quadratic polynomial map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ over a finite field $\mathbb{F}_q$, and target vectors $\mathbf{y}_1, \cdots, \mathbf{y}_\sigma \in \mathbb{F}_q^m$ find $\mathbf{x}_1 \cdots, \mathbf{x}_\sigma \in \mathbb{F}_q^n$ that satisfy

$$\dim(\langle \mathcal{P}(\mathbf{x}_1) - \mathbf{y}_1, \cdots, \mathcal{P}(\mathbf{x}_1) - \mathbf{y}_1 \rangle) \leq d\,.$$

We can immediately make a number of observations about the AMQ Problem. We write AMQ$[n, m, \sigma, d]$ to make the parameters explicit

1. AMQ$[n, m, \sigma, 0]$ is equivalent to σ instances of the MQ$[n, m]$ problem.

2. If $d \geq \sigma$ the AMQ$[n, m, \sigma, d]$ problem is trivial since any choice of $\mathbf{x}_i$ will be a solution.

3. AMQ$[n, m, \sigma, d] >$ AMQ$[n, m, \sigma, d+1]$, that is, the problem gets easier with increasing d.

4. AMQ$[n, m, \sigma, d] <$ AMQ$[n, m, \sigma + 1, d]$, that is, the problem gets harder with increasing σ.

5. AMQ$[n, m, \sigma, d] <$AMQ$[m, n, \sigma - 1, d]$+MQ$[m - d, n]$ because the AMQ problem can be solved by first solving the AMQ problem for the first $\sigma - 1$ target vectors to find the $\mathbf{x}_1, \cdots, \mathbf{x}_{\sigma-1}$ and then solve the quadratic system $\mathcal{S} \circ \mathcal{P}(\mathbf{x}_\sigma) = \mathcal{S}(\mathbf{y}_\sigma)$ where $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^{m-d}$ is a projection onto a linear complement of the subspace spanned by all the errors made by the $\mathbf{x}_1 \cdots \mathbf{x}_{\sigma-1}$.

We can combine observations 5 and 2 into an algorithm for solving the AMQ problem. Using observation 5 a number of times we can reduce AMQ$[n, m, \sigma, d]$ to solving AMQ$[n, m, d, d]$ and solving $\sigma - d$ instances of the MQ$[n, m - d]$ problem. Observation 2 tells that the former is trivial. In practice this boils down to:

1. Choose the $\mathbf{x}_1, \cdots, \mathbf{x}_d$ at random.

2. If we are lucky the dimension of $\langle \mathcal{P}(\mathbf{x}_1) - \mathbf{y}_1, \cdots, \mathcal{P}(\mathbf{x}_d) - \mathbf{y}_d \rangle$ is not equal to d and we can choose more $\mathbf{x}_i$ at random until the space spanned by the error has dimension d.

3. Calculate $\mathcal{S}$, a projection onto a linear complement of the space of errors. This means that for a vector $v \in \mathbb{F}_q^m$ we have $\mathcal{S}(v) = 0$ if and only if v lies in the space of errors. This can be done as follows: Calculate a basis $v_1, \cdots, v_d$ for the subspace spanned by the errors and complete it to a basis $v_1, \cdots, v_m$ for all of $\mathbb{F}_q^m$. Let V be the matrix whose columns are the v_i, then a matrix representation of $\mathcal{S}$ is given by the submatrix of the lowest $m - d$ rows of V^{-1}.

4. Solve the quadratic systems $\mathcal{S}(\mathcal{P}(\mathbf{x}_i) - \mathbf{y}_i) = 0$ to find the remaining $\mathbf{x}_i$.

The security of the new signature scheme depends on the hardness of the AMQ problem, because every time an attacker can solve an instance of the AMQ$[n, m, \sigma, d]$ problem he has a probability of $q^{-d\alpha}$ that the errors lie in the kernel of $\mathcal{S}$ and that they go unnoticed by the verifier. So in order for the signature scheme to be secure it should be impossible to produce a large number of solutions to an AMQ$[n, m, \sigma, d]$ problem with a sufficiently low value of d. This seems likely, but more research into the hardness of solving the AMQ problem and how it relates to the hardness of the MQ problem is needed.

6.2.2 AMQ attack

The AMQ attack repeatedly solves the AMQ problem with the public map $\mathcal{P}$, the target set $\mathbf{y}_i = \mathcal{H}(M||i)$ and some value of d. For each solution set $\mathbf{s}_1, \cdots, \mathbf{s}_\sigma$ the attacker checks if $\mathcal{S} \circ \mathcal{P}(\mathbf{s_i}) = \mathcal{S} \circ \mathcal{H}(M||i)$ for all i, which happens with probability $q^{-d\alpha}$. If such a solution set is found it can be used to make a valid signature by adding the appropriate $\mathcal{R}$, v_{a_i} and Merkle paths. The attack has a complexity of

$$\min_d q^{d\alpha}(\text{AMQ}[n, m, \sigma, d] + C)\,,$$

where C stands for the complexity of checking whether a solution set $\mathbf{s}_i$ satisfies $\mathcal{S}\circ\mathcal{P}(\mathbf{s_i}) = \mathcal{S}\circ\mathcal{H}(M||i)$ for all i.

We know of no better algorithm for solving the AMQ problem than the algorithm we described earlier. If we use that approach the AMQ problem reduces to a number of MQ problems and the complexity of the attack becomes

$$\min_d q^{d\alpha}((\sigma-d)\mathrm{MQ}[n,m-d]+C)\,.$$

If we use (3.1) to estimate the complexity of the MQ problem we see that as d increases the MQ problem gets easier, but it does not get easier fast enough to make up for the increase in the factor $q^{d\alpha}$, even if $\alpha = 1$. Therefore the complexity of the attack grows with d until the case $d = \sigma$ where suddenly the AMQ problem is trivial and we do not have to solve any MQ problem at all. Therefore the most efficient choice of d is always $d = 0$ or $d = \sigma$, so the complexity of the attack is equal to

$$\min\left(\sigma\mathrm{MQ}[n,m], q^{\sigma\alpha}C\right)\,.$$

This suggests that the signature scheme is secure if $\mathrm{MQ}[n,m] > 2^\ell$ and $q^{\sigma\alpha} > 2^\ell$, where ℓ is the desired security level.

6.3 Choice of Parameters

When choosing the parameters for this signature scheme the same constraints have to be satisfied as in the case of the previous chapter. That is, the parameters q, n, m should guarantee that the original UOV scheme is secure and κ, τ and ϑ should be large enough such that the inequalities $\kappa > \ell$ and $\left(\frac{m(m+1)}{2\tau}\right)^\vartheta > 2^\ell$ hold[1]. However, instead of $q^\alpha > 2^\ell$ we now demand that $q^{\sigma\alpha} > 2^\ell$.

In the previous chapter it was beneficial to work over a large finite field, because having a large value q allows for a lower value of α, which makes the signatures smaller. However, now we can compensate for a small value of α by increasing σ, so it is no longer necessary to work over large finite fields. A small problem when working over a smaller field is that q is no longer higher than τ, which is a problem because we need to evaluate $\hat{\mathcal{P}}$ at τ different values to produce the leaves of the Merkle tree. The obvious solution is to define the polynomial $\hat{\mathcal{P}}$ as a polynomial over a degree k field extension of $\mathbb{F}_q$, with k large enough such that $q^k > \tau$.

6.4 Implementation and results

As the signature scheme is very similar to the signature introduced in the previous chapter it did not take much work to implement this signature scheme. The major difference is the finite field that is used. We work with the field $\mathbb{F}_{127}$, which is straightforwardly implemented with addition and multiplication modulo 127. We also needed a degree 3

[1] replace ℓ by 2ℓ for quantum security

Table 6.1: Parameter choices and corresponding signature sizes for different security levels. All parameter sets have $q = 127, k = 3$ and $\tau = 2^{16}$. The size of the public key is equal to $\kappa = l$ or $\kappa = 2l$ bits.

security level		(m, v)	$(\vartheta, \sigma, \alpha)$	|sig| (kB)	classical security
100 bits	classical	(39,78)	$(13, 8, 2)$	6.1	
	quantum	(47,94)	$(27, 15, 2)$	18.0	122-bit
128 bits	classical	(50,100)	$(18, 10, 2)$	10.3	
	quantum	(60,120)	$(38, 19, 2)$	30.9	153-bit
192 bits	classical	(76,152)	$(32, 14, 2)$	26.0	
	quantum	(92,184)	$(70, 28, 2)$	82.9	231-bit
256 bits	classical	(102,204)	$(49, 19, 2)$	51.3	
	quantum	(124,248)	$(110, 37, 2)$	170	310 -bit

Table 6.2: Running times for the key generation, signing and verification algorithms on a single thread on an Intel®Core™ i7-4710MQ CPU at 2.5 GHz

security level		key gen (s)	sig gen (s)	verification (ms)
100 bits	classical	25	1.6	45
	quantum	39	4.9	135
128 bits	classical	46	4.1	102
	quantum	79	13	322
192 bits	classical	148	22	446
	quantum	263	79	1558
256 bits	classical	351	77	1409
	quantum	644	289	4901

extension of $\mathbb{F}_{127}$, for which we used $\mathbb{F}_{127}[X]/(X^3 - 3)$.

By comparing Table 6.1 to Table 5.1 we see that, using the idea of using multiple UOV signatures for the same message, we are able to reduce the size of the signatures by a factor of 2. Comparing Table 5.2 and Table 5.2 we see that the key generation and signing algorithms of the modified signature scheme is roughly as fast as the original UOVHash scheme, which means that it is quite slow. This is because the bottleneck is building the Merkle trees, which is largely unaffected by the modification. The signature verification algorithm is slower, because we have to verify a large number of signatures instead of just one.

6.5 Application to other MQ signature schemes

The idea presented in this chapter and the previous chapter can be applied to any MQ signature scheme. The reduction proof of the previous chapter did not depend on any of the details of the UOV signature scheme and holds for any other MQ signature scheme. That means that if we start from a MQ signature scheme that provides ℓ bits of security and we apply the modification with parameters such that $\kappa > l$, $\left(\frac{n(n+1)}{2k\tau}\right)^{\vartheta} < 2^{-l}$ and $q^{\alpha} > 2^{l}$ we get a signature scheme that also provides ℓ bits of security. The new signature

Table 6.3: Applying our modification to other MQ signature schemes. For all modified MQ schemes we have chosen $\tau = 2^{16}$

Signature scheme		parameters	|pk|	|s|
RainbowLRS2 (128 bit security)	original	$q = 256, n = 79, m = 43$	45 kB	632 bits
	modification 1	$\alpha = 16, k = 2, \vartheta = 19$	128 bits	23 kB
	modification 2	$\alpha = 1, \sigma = 16$	128 bits	8.6 kB
Gui-127 (120 bit security)	original	$q = 2, n = 133, m = 123$	139 kB	163 bits
	modification 1 [2]	$\alpha = 120, k = 16, \vartheta = 18$	120 bits	144 kB
	modification 2	$\alpha = 2, \sigma = 60$	120 bits	12 kB

scheme has as public key a Merkle root of κ bits. A signature consists of one signature of the original signature scheme, α polynomials and ϑ roots of the Merkle tree and Merkle paths. The total size of the signature is therefore

$$|s_{original}| + \frac{\alpha}{m}|pk_{original}| + \vartheta\left(\kappa\lceil\log_2(\tau)\rceil + mk\lceil\log_2(q)\rceil\right) \text{bits}.$$

Alternatively, using the approach introduced in this chapter, we can introduce the parameter σ and relax the condition $q^\alpha > 2^l$ to $q^{\sigma\alpha} > 2^l$. If the AMQ problem is hard enough for the public system of the MQ signature scheme we started from this results in a secure signature scheme. The size of the public key is then still κ bits and the size of the signatures is

$$\sigma|s_{original}| + \frac{\alpha}{m}|pk_{original}| + \vartheta\left(\kappa\lceil\log_2(\tau)\rceil + mk\lceil\log_2(q)\rceil\right) \text{bits}.$$

In Table 6.3 we apply the two modifications to the RainbowLRS2[28] signature scheme and the Gui-127 signature scheme [29]. In the case of the RainbowLRS2 signature scheme we can reduce the combined cost of a signature and the public key by a factor of 2 and provably have the same level of security as the original RainbowLRS2 signature scheme. Or, relying on the hardness of the AMQ problem we can reduce the size by a factor of 5. For the Gui signature scheme, we cannot reduce the sizes whist provably having the same security level as the original Gui-127 scheme. This is because Gui works over the field with 2 elements, so in order to have $q^\alpha > 2^l$ we need $\alpha = 120$. However, if we use multiple signatures we can have a very small value of α, which reduces the combined size of a signature and the public key by an order of magnitude.

[2] To protect against a birthday attack the original Gui-127 signature scheme uses a strategy that signs a message with 4 rounds of HFEv-. With our modification this approach is no longer usable, instead we can just use $\sigma = 4$. This has only a very small impact on the total size of the signature.

Chapter 7

Conclusion

Multivariate cryptography is a promising candidate for providing secure post-quantum signature schemes. However, these signature schemes have large public keys, which is a problem for many applications. In this thesis two new multivariate signature schemes are proposed with much smaller public keys. Figure 7.1 compares the new signature schemes (LUOV and UOVHash) to some other signature schemes from the various branches of post-quantum cryptography. SPHINCS-256 is a hash-based signature scheme that provides 128-bits of post-quantum security with a public key of 1kB and signatures of 41kB [5]. From the branch of lattice based cryptography we have the BLISS-II signature scheme which provides the same level of security with public keys of 7kB and signatures of 5kB. Lastly, we have included UOVRand and RainbowLRS, two other multivariate signature schemes which have small signatures of around 0.1kB but large public keys of around 50kB. Figure 7.1 shows that the new signature schemes have smaller keys and signatures than the existing post-quantum signature schemes.

7.1 Lifted UOV

The simple idea of lifting a UOV key pair from $\mathbb{F}_2$ to an extension field $\mathbb{F}_{2^r}$ increases the security against direct attacks without affecting the size of the public key. At the same time, thanks to the method of Petzoldt, we can increase the number of vinegar variables to protect against key recovery attacks without increasing the size of the public key. These two ideas come together to create a secure signature scheme whose public key is an order of magnitude smaller than other MQ signature schemes, with slightly larger signatures. The signature scheme is very competitive with other post-quantum signature schemes. By choosing the parameter r it is possible to make a trade-off between larger public keys and smaller signatures or vice versa. We made a rudimentary ANSI C implementation of the Lifted UOV signature scheme which shows that key generation, signing and verification takes only a few milliseconds for 100-bit security instantiations of the scheme and up to a few hundred milliseconds for 256-bit security instantiations. However it is very likely that these times can be improved significantly.

The idea of lifting public keys to an extension field can be used to reduce the size of the public keys of the Rainbow signature scheme, but is probably not beneficial for MQ schemes that do not have semi-regular public keys such as HFE and C*.

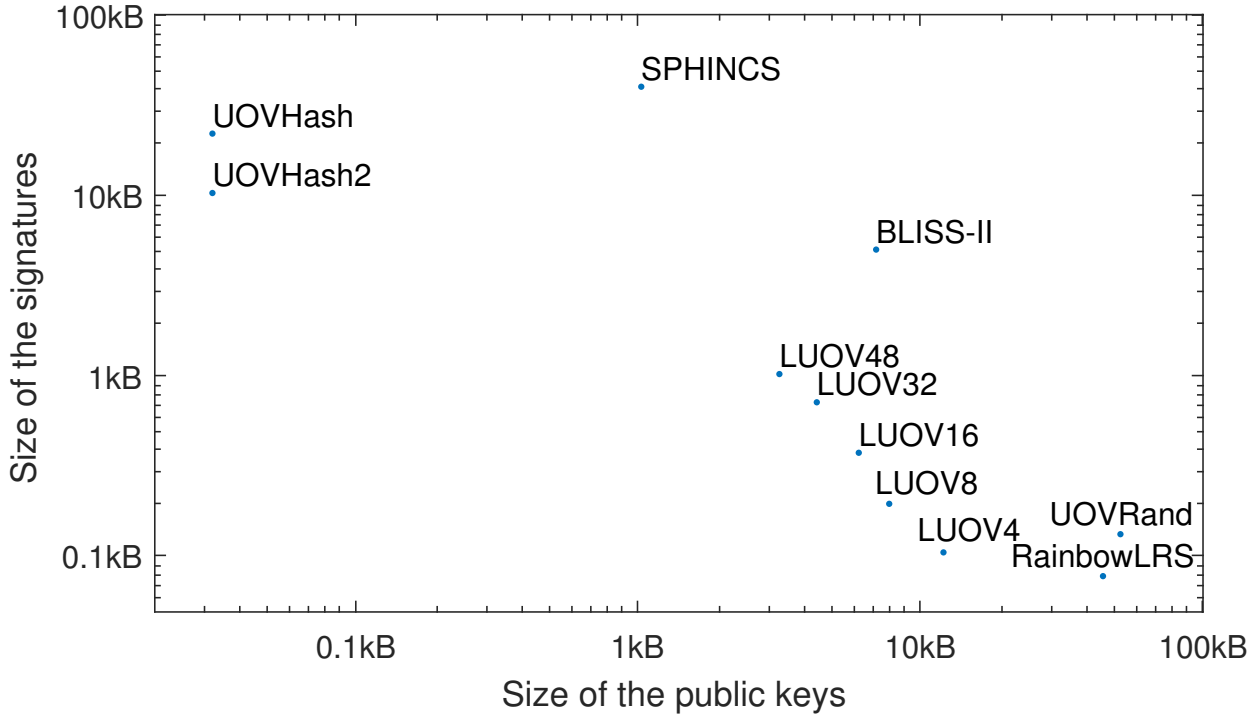

Figure 7.1: Comparison of key and signature sizes of some post-quantum signature schemes providing 128 bits of security

7.2 UOVHash

The UOVHash signature scheme achieves tiny public keys and somewhat larger signatures but reduces the sum of both sizes significantly. In this signature scheme the verifier only checks a few random linear combinations of the public system instead of the full system. This way the verifier does not need to know the full system, which saves a lot of bits in communication. To make the idea work we borrow the concept of a Merkle tree from hash based cryptography. The resulting signature scheme is provably as secure as the original UOV signature scheme. By using multiple UOV signatures for the same message, we can reduce the size of the signatures significantly. The security of this modification depends on the hardness of a new problem, the Approximate MQ problem, which is a natural generalization of the MQ problem.

This construction can be applied to any other MQ signature scheme to decrease the combined size of a signature and a public key significantly. The parameters can be chosen such that there is provably no loss in security by applying the modification. Alternatively, it is possible to choose the parameters more aggressively to achieve smaller signatures whose security depends on the AMQ problem. In Sect. 6.5 we show that this idea can reduce the combined cost of a signature and a public key by a factor 5 in the case of the RainbowLRS2 signature scheme and in the case of Gui-127 by a factor of 10.

List of Figures

List of Tables

List of Algorithms

List of Symbols

General Signature schemes

d	A document to sign
pk	A public key
sk	A secret key
s	A signature
$\|d\|$	The number of bits (or Bytes, kiloBytes) needed to represent a document to sign
$\|pk\|$	The number of bits (or Bytes, kiloBytes) needed to represent a public key
$\|sk\|$	The number of bits (or Bytes, kiloBytes) needed to represent a secret key
$\|s\|$	The number of bits (or Bytes, kiloBytes) needed to represent a signature
ℓ	The security level of a signature scheme

MQ signature schemes

$\mathcal{F}$	Central map
$\mathcal{S}$	Linear map that shuffles the components
$\mathcal{T}$	Linear map that shuffles the variables
m	Number of components of public key
n	Number of variables of public key
$\mathcal{P}$	Public map

Algebra over finite fields

$\mathbb{F}_q^n$ A n-dimensional vector space over $\mathbb{F}_q$

$\mathbb{F}_q$ A finite field with q elements

$\mathbf{M}_{\mathcal{E}}$ A matrix representation for a linear map $\mathcal{E}$ (with respect to some basis which is not mentioned explicitly)

$\mathbf{M}_f$ A matrix representation for a quadratic form f (with respect to some basis which is not mentioned explicitly)

$\mathbf{A}$ A matrix

$GL(q, n)$ The general linear group of order n over the field $\mathbb{F}_q$

q The number of elements of a finite field

$\mathbf{A}^\top$ The transpose of the matrix $\mathbf{A}$

$\mathbf{I}_a$ The unit matrix of size $a \times a$

$\mathbf{0}_{a\times b}$ The zero $a \times b$ matrix

Other Symbols

$\mathcal{H}$ A hash function

$\mathrm{InSec}^p(s; t, q)$ An insecurity function for property p (see Sect. 2.5)

$H_{d,i}$ The i-th node of a Merkle tree at depth d

$\oplus$ The bitwise XOR operation

Appendix A

Software implementation

This appendix gives an overview of the software implementations of the UOV signature scheme and the two new signature schemes that are developed in this thesis. The source code merely serves as a demonstration of the algorithms and is by no means a secure implementation. For example, this implementation does not use cryptographically secure randomness and makes no attempt at preventing cache timing attacks.

The source code is publicly available at `https://github.com/WardBeullens/ThesisCode`, consists of 42 files and roughly 3000 lines of code, excluding blank lines and comments. The source code is divided in four folders:

- **common** Contains code which is used by all 3 algorithms such as field arithmetic. This folder also contains code for running and timing the algorithms in the "main.c" file.
- **UOVClassic** Contains code for the implementation of UOV, with Petzold's method of reducing the size of the public key.
- **LUOV** Contains code for the implementation of UOV, with Petzold's method of reducing the size of the public key LUOV/ : Contains the implementation for a version of UOV with public and private with coefficients in $\mathbb{F}_2$, but lifted to a large extension field of $\mathbb{F}_2$.
- **UOVHash** Contains the implementation for a version of UOV which reduced the public key size by using techniques from hash based crypto.

Folder	Files	Explanation
common\	api.h	Defines the size of the keys and signatures. This file is required to use the SUPERCOP framework.
	buffer.(h/c)	Implementation of writer and reader structs, which are used for serializing and deserializing objects such as keys and signatures.
	csprng.(h/c), Keccak-readable-and-compact-c89.c	Implementation of a cryptographically secure pseudo-random number generator based on Keccak.

Folder	Files	Explanation
common\	F16Field.(h/c), F32Field.(h/c), F48Field.(h/c), F64Field.(h/c), F80Field.(h/c), PrimeField.(h/c)	Implements the arithmetic in the finite fields $\mathbb{F}_{2^{16}}, \mathbb{F}_{2^{32}}, \mathbb{F}_{2^{48}}, \mathbb{F}_{2^{64}}, \mathbb{F}_{2^{80}}, \mathbb{F}_{31}$ and $\mathbb{F}_{127}$.
	LinearAlgebra.(h/c)	Implementation of matrices over a finite field and linear algebra subroutines such as matrix multiplication, gaussian reduction and solving systems of linear equations.
	main.c, parameters.h	Defines a main function which tests a signature scheme and reports the time and signature sizes of the scheme. The parameters.h file determines which signature scheme is used (UOVClassic, LUOV or UOVHash) and how many keys and signatures are generated and verified.
	twister.(h/c)	Implementation of the Mersenne Twister, a pseudo-random number generator. This PRNG is used to generate the first part of the public key.
UOVClassic\	UOVClassic.(h/c)	Implementation of the Key generation, Signature generation and Verification algorithms of the UOV signature scheme.
	UOVClassicApi.h	Defines the size of the keys and signatures. This file is required to use the SUPERCOP framework.
	UOVClassicParameters.h	Defines q, m and v, the parameters of the UOV signature scheme.
LUOV\	LUOV.(h/c)	Implementation of the Key generation, Signature generation and Verification algorithms of the Lifted UOV signature scheme.
	LUOVApi.h	Defines the size of the keys and signatures. This file is required to use the SUPERCOP framework.
	LUOVParameters.h	Defines r, m and v, the parameters of the Lifted UOV signature scheme.
	128Bitontainer.(h/c)	Defines a struct that contains up to 128 bits.

Folder	Files	Explanation
UOVHash\	UOVHash.(h/c)	Implementation of the Key generation, Signature generation and Verification algorithms of the UOVHash signature scheme.
	UOVHashApi.h	Defines the size of the keys and signatures. This file is required to use the SUPERCOP framework.
	UOVHashParameters.h	Defines $q, m, v, \alpha, \tau, \kappa, k$ and ϑ, the parameters of the UOVHash signature scheme.
	MACField.(h/c)	The implementation of arithmetic over a degree-k field extension of $\mathbb{F}_q$. The MAC polynomials are defined with coefficients in this field.
	MerkleTree.(h/c)	The implementation of the merkle tree construction.

Bibliography

[1] Nist post-quantum crypto project. http://csrc.nist.gov/groups/ST/post-quantum-crypto/.

[2] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. PhD thesis, Université Pierre et Marie Curie-Paris VI, 2004.

[3] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *Journal of Complexity*, 29(1):53–75, 2013.

[4] Daniel J Bernstein. Cost analysis of hash collisions: Will quantum computers make sharcs obsolete? *SHARCS09 Special-purpose Hardware for Attacking Cryptographic Systems*, page 105, 2009.

[5] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-OHearn. SPHINCS: practical stateless hash-based signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer, 2015.

[6] Luk Bettale, Jean-Charles Faugere, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, 3(3):177–197, 2009.

[7] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: Improved analysis of the hybrid approach. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 67–74. ACM, 2012.

[8] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 41–69. Springer, 2011.

[9] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 203–218. Springer, 2010.

[10] Charles Bouillaguet, Pierre-Alain Fouque, and Amandine Véber. Graph-theoretic algorithms for the isomorphism of polynomials problem. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 211–227. Springer, 2013.

[11] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[12] Crystal Clough, John Baena, Jintai Ding, Bo-Yin Yang, and Ming-Shing Chen. Square, a new multivariate encryption scheme. In *Cryptographers Track at the RSA Conference*, pages 252–264. Springer, 2009.

[13] Stephen Cook. The P versus NP problem. *The millennium prize problems*, pages 87–104, 2006.

[14] Peter Czypek. *Implementing Multivariate Quadratic Public Key Signature Schemes on Embedded Devices*. PhD thesis, Ph. D. thesis, Diploma Thesis, Chair for Embedded Security, Ruhr-Universität Bochum, 2012.

[15] Jintai Ding, Christopher Wolf, and Bo-Yin Yang. l-invertible cycles for$\mathcal{M}$ultivariate $\mathcal{Q}$uadratic ($\mathcal{MQ}$) public key cryptography. In *International Workshop on Public Key Cryptography*, pages 266–281. Springer, 2007.

[16] Jintai Ding, Bo-Yin Yang, Chia-Hsin Owen Chen, Ming-Shing Chen, and Chen-Mou Cheng. New differential-algebraic attacks and reparametrization of Rainbow. In *International Conference on Applied Cryptography and Network Security*, pages 242–257. Springer, 2008.

[17] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013.

[18] Jean-Charles Faugere and Antoine Joux. Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using gröbner bases. In *Annual International Cryptology Conference*, pages 44–60. Springer, 2003.

[19] Jean-Charles Faugère and Ludovic Perret. On the security of UOV. *IACR Cryptology ePrint Archive*, 2009:483, 2009.

[20] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[21] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. Software speed records for lattice-based signatures. In *International Workshop on Post-Quantum Cryptography*, pages 67–82. Springer, 2013.

[22] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography–PKC 2016*, pages 387–416. Springer, 2016.

[23] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999.

[24] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil and vinegar signature scheme. In *Annual International Cryptology Conference*, pages 257–266. Springer, 1998.

[25] Jianqiang Luo, Kevin D Bowers, Alina Oprea, and Lihao Xu. Efficient software implementations of large finite fields $\mathbb{F}_{2^n}$ for secure storage applications. *ACM Transactions on Storage (TOS)*, 8(1):2, 2012.

[26] R Garey Michael and S Johnson David. Computers and intractability: a guide to the theory of NP-completeness. *WH Free. Co., San Fr*, 1979.

[27] Jacques Patarin. The oil and vinegar signature scheme. In *Dagstuhl Workshop on Cryptography1997*, 1997.

[28] Albrecht Petzoldt. Selecting and reducing key sizes for multivariate cryptography. 2013.

[29] Albrecht Petzoldt, Ming-Shing Chen, Bo-Yin Yang, Chengdong Tao, and Jintai Ding. Design principles for HFEv-based multivariate signature schemes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 311–334. Springer, 2015.

[30] Albrecht Petzoldt, Enrico Thomae, Stanislav Bulygin, and Christopher Wolf. Small public keys and fast verification for $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic public key systems. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 475–490. Springer, 2011.

[31] Peter W Shor. Polynomial time algorithms for discrete logarithms and factoring on a quantum computer. In *International Algorithmic Number Theory Symposium*, volume 877, pages 289–289. Springer, 1994.

[32] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology EPrint Archive*, 2004:332, 2004.

[33] Alan Szepieniec, Ward Beullens, and Bart Preneel. MQ signatures for PKI. In *The Eighth International Conference on Post-Quantum Cryptography*. Springer, 2017.

[34] Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In *International Workshop on Public Key Cryptography*, pages 156–171. Springer, 2012.

[35] Christopher Wolf and Bart Preneel. Equivalent keys in multivariate quadratic public key systems. *Journal of Mathematical Cryptology*, 4(4):375–415, 2011.

[36] Christof Zalka. Grovers quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999.

AFDELING
Straat nr bus 0000
3000 LEUVEN, BELGIË
tel. + 32 16 00 00 00
fax + 32 16 00 00 00
www.kuleuven.be