University of BRISTOL

SCHOOL OF COMPUTER SCIENCE

EUF-CMA security of MAYO in the random oracle model

Matthew Swann

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Sunday 23rd February, 2025

Abstract

Quantum computers present a formidable challenge to conventional cryptography, endangering the very foundations of internet security. MAYO is a quantum resistant signature scheme based on the Oil and Vinegar scheme, one of the oldest and most studied multivariate quadratic signature schemes. The security of cryptographic schemes can be studied in the random oracle model providing confidence in their construction by bounding the insecurity of the system. This work proves a tightened bound for the EUF-CMA security of MAYO signatures and discusses the implications of the randomisation bit R on the proof. Each game of the proof is fully described and presented as a program to aid accessibility and auditing. Further, this work provides a proof of SUF-CMA security for a modified version of the scheme, MAYO$^-$ by introducing a new hardness assumption, and demonstrating that its hardness is required for SUF-CMA to be achievable.

Dedication and Acknowledgements

I would like to thank my supervisor François Dupressoir for all his support during this project. I would also like to thank Sofía Celi for her help in understanding MAYO. I would like to thank the MEng weekly stand-up group and my friends for keeping me motivated and entertained throughout this project. I would finally like to thank my family for all their support and help throughout my undergraduate degree.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others including AI methods, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Matthew Swann, Sunday 23rd February, 2025

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, François Dupressoir

Notation and Acronyms

Acronyms

PQC	Post Quantum Cryptography
QRC	Quantum Resistant Cryptography
CRQC	Cryptographically Relevant Quantum Computer
EUF	Existential Unforgeability
SUF	Strong Existential Unforgeability
CMA	Chosen Message Attack
KOA	Key Only Attack
MQ	Multivariate Quadratic Polynomials

Notation

pk	public key
sk	secret key
M	a message
sig	a signature
$\mathcal{B}$	The alphabet of $0, 1$. $\mathcal{B}^*$ denotes an arbitrary length bitstring.
$x \leftarrow y$	set the variable x to the value of y
$x \overset{\$}{\leftarrow} \mathcal{D}$	set the variable x to a randomly sampled value from the distribution $\mathcal{D}$
$x \leftarrow \mathsf{Cache}[y]$	Table lookup retrieving the value stored at y

Chapter 1

Introduction

Proofs provide confidence in cryptographic schemes, protocols, and primitives. Proofs validate constructions are useful in practice by bounding their insecurity in relation to the security of other constructions or the difficulty of solving hard problems.

The majority of classical asymmetric cryptography [15] is weak against quantum computers due to Shor's Algorithm [21], which can solve both the Large Prime Factorisation (LPF) problem and Discreet Logarithm Problem (DLP) in polynomial time. This allows for the decryption of data or forging of signatures in a short amount of time. Symmetric cryptography is still thought to be resistant against quantum computers as algorithms such as Grover Search [12] can be mitigated by doubling the key size. Researchers generally accept that a Cryptographically Relevant Quantum Computer (CRQC) will be created in the future [16]. The huge number of devices which need updating to use quantum safe algorithms [15] motivates early research and standardisation so that the transition is finished before a CRQC is developed. This also mitigates retrospective decryption (store now, decrypt later) attacks on high-value sensitive information.

The National Institute of Standards and Technology (NIST) hosts standardisation competitions, inviting researchers to submit schemes to be peer reviewed and tested before being standardised for widespread usage. Though proof of security is not a requirement for a scheme to be considered viable, schemes are "evaluated based on how well they appear to provide" [17] EUF-CMA security. Therefore creating detailed security proofs which can be easily verified is advantageous for schemes looking to be standardised.

MAYO [4] is a post quantum signature scheme with attractive signature sizes as well as fast signing and verifying times [23]. Its submission to the NIST Digital Signature Schemes competition (by Beullens, Campos, Celi, Hess, and Kannwischer [22]) contains a proof for EUF-CMA security, however, the proof is complex and omits details which would make the proof more accessible and easy to verify.

The objectives of this work are to:

- Reason about the optional randomisation parameter R and its impact on security.
- Tighten the security bound for EUF-CMA security for the MAYO scheme.
- Provide programs representing each game used in the proof, allowing for greater accessibility and understanding of each step.
- Present the EUF-KOA proof outlined by the MAYO team with additional supporting information.
- Outline a basic SUF-CMA proof, introduce a new hardness assumption, and suggest further work which could be used to improve upon it.

Chapter 2

Background

2.1 Digital Signatures

Digital Signatures provide a mechanism for providing messages with authentication and integrity. Authentication ensures messages are produced by the expected party and integrity ensures messages are not modified after signing. Signature schemes typically consist of the following functions:

- Gen() - creates a public key, secret key pair $(\mathsf{pk}, \mathsf{sk})$ randomly. The secret key sk should be unpredictable and is used for creating new signatures. It must be kept secret as anyone can use it to sign messages.
- Sign$(\mathsf{M}, \mathsf{sk})$ - creates a signature, sig for the message, $\mathsf{M} \in \mathcal{M}$ using the secret key, sk. sig can be used to verify the message.
- Verf$(\mathsf{M}, \mathsf{sig}, \mathsf{pk})$ - returns true if and only if sig is a valid signature for M using the public key, pk.

Without sk, a valid pair $(\mathsf{M}, \mathsf{sig})$ for a given public key pk should be very hard to compute. This implies that any $(\mathsf{M}, \mathsf{sig})$ pair was created with knowledge of the secret key sk and therefore produced by the owner of the key.

Hash and Sign

Many signature schemes require the message M to be a specific size for the signing to be successful and secure. This can be achieved using a hash function $h : \{0,1\}^* \to \{0,1\}^n$, which takes an arbitrarily long message and outputs a constant length bitstring. During signing, the hash of the message (or message digest) is signed and sent along with the message. When verifying, the message digest is calculated and used to check whether the signer produced the message.

This introduces extra requirements on the hash function for the signature scheme to be secure:

Pre-image resistance
Given a hash value h, it should be difficult to find any message m such that $h = \mathsf{hash}(m)$. This helps to prevent an adversary from finding multiple preimages with the same digest.

Second pre-image resistance
Given an input m_1, it should be difficult to find a different input m_2 such that $\mathsf{hash}(m_1) = \mathsf{hash}(m_2)$. Without this property, an adversary could create a signature for any message m_2 after observing a valid pair (m_1, sig).

Collision resistance
It should be difficult to find two different messages m_1 and m_2 such that $\mathsf{hash}(m_1) = \mathsf{hash}(m_2)$. This prevents adversaries from requesting a signature for m_1 and using it to create a forgery for m_2.

In MAYO, the hash function used is $\mathsf{SHAKE256}$. The function maps bitstrings of arbitrary length to infinitely long bitstrings, which are truncated to the required length. $\mathsf{SHAKE256}$ requires the message, M, and length of the output in bytes, l, and outputs a digest for the provided message as specified in the SHA-3 standard [9].

2.2 Security Proofs

2.2.1 Security Notions

Security Notions allow cryptographers to reason about the security of schemes against adversaries with differing goals, resources, and access. To prove confidence in a scheme, cryptographers prove it secure against as powerful an adversary as possible, where the goal is as weak as possible.
Goldwasser, Micali, and Rivest [11] outline two basic types of attacks against signature schemes.

- *Key-only attacks* where the adversary only has access to the public key
- *Message attacks* where the adversary can inspect valid $(\mathsf{M}, \mathsf{sig})$ pairs produced using the secret key.

In this work, we focus on Key-Only Attacks (KOA) and Chosen Message Attacks (CMA). CMA is a type of message attack where the adversary is given access to a signing oracle. The adversary can provide messages to the signing oracle, which will return valid signatures without leaking any other information. In the following proofs we assume the CMA is adaptive, meaning the adversary can pick the messages they query based on the $(\mathsf{M}, \mathsf{sig})$ pairs they have already observed. This is the most general, and therefore the most severe type of attack outlined by Goldwasser et al [11].

The goal of an adversary is related to compromising the security of the scheme in some way. The most powerful goal, total break [10], is for the adversary to extract the secret trapdoor information from interacting with the system. The weakest goal is for the adversary to distinguish between an output from a scheme and a randomly sampled bit string. In this work, we focus on proving security against the weaker goal of existential forgery. This is where the adversary is successful if she can forge any $(\mathsf{M}, \mathsf{sig})$ pair. The scheme is strongly unforgeable if it is hard for an adversary to produce a forgery $(\mathsf{M}, \mathsf{sig})$ where sig is not the output of sending M to the signing oracle.

- An Existentially Unforgeable under Chosen Message Attack scheme (EUF-CMA Figure 2.1) is one where there is no (efficient) adversary with a non-negligible probability of producing a $(\mathsf{M}, \mathsf{sig})$ pair which verifies with the corresponding public key, and M was not sent to the signing oracle.
- A Strong Existentially Unforgeable under Chosen Message Attack scheme (SUF-CMA Figure 2.2) is one where there is no (efficient) adversary with a non-negligible probability of producing a $(\mathsf{M}, \mathsf{sig})$ pair which verifies with the corresponding public key, and sig was not the result of sending M to the signing oracle.

It is important to bound the resources an adversary has access to. With an infinitely long amount of time, any adversary can solve any hardness problem (with a valid solution) via exhaustive search.
To reason about the probability of an adversary breaking a security notion, we create a game which represents this security. Any adversary has the same probability of winning the game as they do at breaking security. This probability is described as $\mathcal{A}$'s advantage over problem P and is expressed as: $\mathsf{Adv}^{P}(\mathcal{A})$.

```
$\mathsf{Exp}^{\mathsf{EUF\text{-}CMA}}_{\mathsf{MAYO}}(\mathcal{A})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{MAYO.Sign}_{\mathsf{sk}}(\cdot)}.\mathsf{Forge}(\mathsf{pk})$
$\mathsf{win} \leftarrow \mathsf{MAYO.Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \mathsf{M}^*$ is fresh
```

Figure 2.1: Existential Unforgeability under Chosen Message Attack experiment played by an adversary $\mathcal{A}$ against the Mayo signature scheme.

```
$\mathsf{Exp}^{\mathsf{SUF\text{-}CMA}}_{\mathrm{MAYO}}(\mathcal{A})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{MAYO.Sign}_{\mathsf{sk}}(\cdot)}.\mathsf{Forge}(\mathsf{pk})$
$\mathsf{win} \leftarrow \mathsf{MAYO.Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge (\mathsf{M}^*, \mathsf{sig}^*)$ fresh
```

Figure 2.2: Strong Unforgeability under Chosen Message Attack experiment played by an adversary $\mathcal{A}$ against the Mayo signature scheme.

2.2.2 Reductions

A reduction is a way of transforming one problem into one or more instances of another. These can be useful for bounding the difficulty of problems. For example: if a hard problem B can be solved by another problem A, then A must be at least as difficult as B.

Consider an adversary $\mathcal{A}$ attempting to find the second preimage of a hash function $\mathcal{H}$. That is, given a message M_1, $\mathcal{A}$ attempts to find a second message M_2 such that $\mathsf{hash}(\mathsf{M}_1) = \mathsf{hash}(\mathsf{M}_2)$. The probability of $\mathcal{A}$ finding such a message is $\mathsf{Adv}^{\text{second-preimage}}_{\mathcal{H}}(\mathcal{A})$. If another adversary $\mathcal{B}$, who has access to adversary $\mathcal{A}$ (denoted as $\mathcal{B}^{\mathcal{A}}$) wants to find a collision in $\mathcal{H}$, they can pick a random message and give it to $\mathcal{A}$ to find a second preimage. If $\mathcal{A}$ is successful then $\mathcal{B}$ has found a collision in $\mathcal{H}$. This can be represented as $\mathsf{Adv}^{collision}_{\mathcal{H}}(\mathcal{B}^{\mathcal{A}}) \geq \mathsf{Adv}^{\text{second-preimage}}_{\mathcal{H}}(\mathcal{A})$ as any improvements an adversary $\mathcal{A}$ makes at finding second preimages can be used by $\mathcal{B}$. If we believe $\mathsf{Adv}^{collision}_{\mathcal{H}}(\mathcal{C})$ to be low, that implies that there are no adversaries who can find a second preimage of $\mathcal{H}$ with higher probability. We use similar reasoning in the MAYO security proof to show that forging messages is likely hard.

It is also important to consider the reductions complexity. Reductions are only useful if the time taken to run the reduction is within an appropriate bound. Constructing a reduction which takes longer than finding the solution through exhaustive search gives use no useful security metrics.

2.2.3 Game Based Proofs

Game based proofs help split reductions into multiple smaller steps, simplifying the reasoning and bounding smaller security losses one after another, rather then all at once at the end. Each of these intermediate steps are referred to as a Game the adversary is playing, and is expressed as a program. As each step is much smaller, it is easier to reason about the difference between programs. This technique also clarifies the reduction complexity, as it is clear in each step how much overhead is introduced.

To reason about the difference between games we first demonstrate that an adversary cannot differentiate between the two games, unless some bad event bad_n occurs. We then calculate the probability of bad_n occurring and aim to demonstrate that it is small. In more formal proofs, it is also required to prove both programs terminate with the same probability.

2.2.4 Random Oracle Model

The random oracle model allows cryptographers to reason about hash functions and their use in schemes. We replace hash functions with an idealised black box where every distinct input is mapped to a randomly sampled value (Figure 2.3). They do not imply security in practice, as schemes have been constructed that are secure in the random oracle model but are trivially insecure when implemented [7]. However, creating proofs in the random oracle model helps provide confidence in the scheme and ensures no obvious design flaws. An adversary's access to the random oracle is also bound as a resource.

```
$\mathcal{H}(\mathsf{M})$
if $\mathsf{M} \notin \mathsf{H}$ then
    $\mathsf{H}[\mathsf{M}] \xleftarrow{\$} \mathcal{B}^{8\mathsf{digest\_bytes}}$
return $\mathsf{H}[\mathsf{M}]$
```

Figure 2.3: Lazy sampling random oracle

2.3 Mayo Signature Scheme

Mayo is a Quantum Resistant signature scheme based on Unbalanced Oil and Vinegar signatures. It is one of the signature schemes currently being considered for the NIST Digital Signature Scheme Competition [18] and has attractive signature and key sizes [23].

2.3.1 Unbalanced Oil and Vinegar

Oil and Vinegar signatures are based on the difficulty of solving a set of multivariate quadratic equations over a small finite field [19]. The set of equations is commonly referred to as a map. This problem is thought to be NP-hard [3] but can be made easy by creating the map with a "trapdoor", some secret information making the problem easy to solve. This trapdoor construction can be used to create a simple signature scheme using the map as a public key. To sign a message, the signer calculates the message's preimage and publishes it along with the message. Now any verifier can apply the map and check the output is equal to the message. As long as only the signer can efficiently compute these preimages, the chance of an adversary signing the message is negligible.

More formally, the public key is the map $\mathcal{P}(\mathbf{x}) = (p_1(\mathbf{x}), ..., p_m(\mathbf{x})) : \mathbb{F}_q^n \to \mathbb{F}_q^m$, where $\mathbb{F}_q$ is a finite field modulo q, $p_1(\mathbf{x}), ..., p_m(\mathbf{x})$ are a set of m multivariate quadratic polynomials (MQ) each consisting of n variables, and $\mathbf{x}$ is a vector of n values, each assigned to one of these variables.

The map $P(\cdot)$ is constructed with a secret oilspace $O \subset \mathbb{F}_q^n$ with dimension less than m on which $\forall \mathbf{o} \in O, \mathcal{P}(\mathbf{o}) = 0$. This allows a signer to efficiently calculate pre-images of the map. The polar form of our map $\mathcal{P}'(\mathbf{x}, \mathbf{y}) = \mathcal{P}(\mathbf{x} + \mathbf{y}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{y})$ is bilinear, meaning (without loss of generality), for a given $\mathbf{x}$ the equation is linear in $\mathbf{y}$, enabling a signer to efficiently compute a signature $\mathbf{s} = \mathbf{v} + \mathbf{o}$. When signing, the signer picks a random vinegar vector $\mathbf{v}$ and can solve for $\mathbf{o}$ using Equation 2.1 and Gaussian elimination. The pre-image of $\mathbf{t}$ is $\mathbf{s} = \mathbf{v} + \mathbf{o}$, which is used as part of the signature.

$$\mathcal{P}(\mathbf{v} + \mathbf{o}) = \underbrace{\mathcal{P}'(\mathbf{v}, \mathbf{o})}_{\text{Linear in } \mathbf{o}} + \underbrace{\cancel{\mathcal{P}(\mathbf{o})}}_{=0} + \underbrace{\mathcal{P}(\mathbf{v})}_{\text{fixed}} = \mathbf{t} \tag{2.1}$$

Rearrangement of a homogeneous MQ polar form [22]

The map $\mathcal{P}$ with secret oilspace O can be constructed by creating a secret map pair $(\mathcal{F}, \mathcal{T})$ with $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$. Each equation $f_0, \dots, f_m \in \mathcal{F}$ are chosen uniformly at random and have the form:

$$y_i = \sum \alpha_{ijk} \cdot o_j v_k + \sum \beta_{ijk} \cdot v_j v_k + \sum \gamma_{ijk} \cdot o_j + \sum \delta_{ijk} \cdot v_j$$

Where α, β, γ and δ are all randomly sampled values, o denotes the oil variables, and v denotes the vinegar variables. Once the values of vinegar variables are set, it is clear that each equation becomes linear in the oil variables. The map $\mathcal{T}$ is randomly sampled from the General linear group, the group of matrices with an inverse. This hides the oil subspace in the map $\mathcal{P}$ while still allowing a signer with knowledge of $(\mathcal{F}, \mathcal{T})$ to calculate the preimage $\mathbf{s}'$ under $\mathcal{F}$ and convert it to a preimage $\mathbf{s} = \mathcal{T}^{-1}(\mathbf{s}')$ under $\mathcal{P}$.

Unbalanced

In [14] Kipins and Shamir devised an attack against the Oil and Vinegar Scheme which recovered an eigenspace for O, allowing for Universal Forgery. In [13] A.Kipins, J. Patarin, and L. Goubin analysed the unbalanced variant, where there are more vinegar variables than oil variables. Specifically when $v \geq 2o$ the attack for recovering an eigenspace of O becomes infeasible. As the number of Oil and Vinegar variables are no longer in equal quantities, this variation is referred to as the Unbalanced Oil and Vinegar scheme (UOV). The Unbalanced Oil and Vinegar hardness problem has been extensively studied with no powerful attacks against the scheme found. This makes it a good basis to create schemes from as there are less likely to be issues introduced from the UOV construction.

Key compression

When creating a map $\mathcal{P}$ which contains an oilspace O of dimension m, there are only $\binom{m+1}{2}$ linear constraints [5]. This means the majority of the public map $\mathcal{P}$ can be pseudo-randomly generated from a public seed ($\mathsf{seed}_{\mathsf{pk}}$), and the remaining coefficients can be calculated such that $\mathcal{P}(O) = 0$. Only these coefficients are required along with the public seed for anyone to construct the entire public map. This means a smaller public key can be sent rather than the entire map [20]. The public map is fully described by three smaller matrices, $(\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m}, \{\mathbf{P}_i^{(3)}\}_{i\in m})$ where $(\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$ can be pseudorandomly generated and $\{\mathbf{P}_i^{(3)}\}_{i\in m}$ is calculated using the linear constraints as well as $(\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$.

2.3.2 Whipping

Mayo improves upon the Unbalanced Oil and Vinegar scheme by reducing the public key size by shrinking the oilspace. This allows a larger proportion of the public key to be pseudo-randomly generated from the public $\mathsf{seed}_{\mathsf{pk}}$ (demonstrated in [20]). The problem of finding O for a given $\mathcal{P}$ also becomes much harder meaning the size of $\mathcal{P}$ can be considerably reduced and still achieve the same security level, resulting in a smaller public key.

Mayo makes $o = \mathsf{dim}(O) \leq m$, and "whips up" the smaller map $\mathcal{P} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$ into a larger map $\mathcal{P}^\star : \mathbb{F}_q^{kn} \rightarrow \mathbb{F}_q^m$ (Equation 2.2), with $n \leq m \leq kn$. Additionally, we must have $ko > m$ so that the signing algorithm can successfully sample signatures with high probability, and $n > 2o$ to prevent the extended Kipnis-Shamir attack [8].

$$\mathcal{P}^\star(\mathbf{x}_1, \ldots, \mathbf{x}_k) := \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{x}_i) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{x}_i, \mathbf{x}_j) \tag{2.2}$$

Whipped map $\mathcal{P}^\star$ with public $\mathbf{E}$-matrices [22]

The matrices $\mathbf{E}_{ij} \in \mathbb{F}_q^{m\times m}$ are public and selected so that their non-trivial linear combinations have rank m. This prevents the introduction of extra oil spaces in $\mathcal{P}^\star$ [5]. Importantly, this construction has the property that for all vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$, if $(\forall_{i\in[k]} : \mathbf{x}_i \in O)$ then $\mathcal{P}^\star(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 0$. This allows the signer to calculate preimages efficiently as for a given vinegar $\mathbf{v}$, the equation is still linear in $\mathbf{o}$. This introduces a new assumption:

Definition 2.3.1 (Multi-Target Whipped MQ problem (MTWMQ) [4]). *For some matrices $\{\mathbf{E}_{ij}\}_{1\leq i\leq j\leq k} \in \mathbb{F}_{q^m}$, given random $P \in \mathsf{MQ}_{n,m,q}$ and access to an unbounded number of random targets $\mathbf{t}_i \in \mathbb{F}_q^m$ for $i \in \mathbb{N}$, the multi-target whipped MQ problem asks to compute $(I, s_1, ..., s_k)$, such that*

$$\sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{x}_i) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{t}_I$$

Let $\mathcal{A}$ be an adversary. We say that the advantage of $\mathcal{A}$ against the multi-target whipped MQ problem is:

$$\mathrm{Adv}^{\mathrm{MTWMQ}}_{\{\mathbf{E}_{ij}\},n,m,k,q}(\mathcal{A}) = \Pr\left[\sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{s}_i) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j) = \mathbf{t}_I \;\middle|\; \begin{array}{c} \mathcal{P} \xleftarrow{\$} \mathrm{MQ}_{n,m,q} \\ \{\mathbf{t}_i\} \xleftarrow{\$} \mathbb{F}_q^{m\times\mathbb{N}} \\ (I, \mathbf{s}_1, \ldots, \mathbf{s}_k) \leftarrow \mathcal{A}^{t_i}(\mathcal{P}) \end{array}\right]$$

Definition 2.3.1 captures the probability of an adversary calculating a primage $\mathbf{s}$ for the value of $\mathbf{t}_I$. She picks I and is given access to an oracle which returns targets $\{\mathbf{t}_i\} \leftarrow \mathbb{F}_q^{m\times\mathbb{N}}$ as well as the public map $\mathcal{P}$. While assumed to be hard, as a recent assumption it has not received the same rigorous analysis as other cryptographic hardness problems.

2.3.3 MAYO Signatures

In MAYO both secret and public keys are generated from a randomly sampled secret seed ($\mathsf{seed}_{\mathsf{sk}}$). This is used to derive the public seed ($\mathsf{seed}_{\mathsf{pk}}$) and the oil space information O. The public map is described by three matrices $\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m}$, and $\{\mathbf{P}_i^{(3)}\}_{i\in m}$. Like in UOV, $\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$ are generated using $\mathsf{seed}_{\mathsf{pk}}$, and $\{\mathbf{P}_i^{(3)}\}_{i\in m}$ is generated from the oilspace information $\mathbf{O}$, $\{\mathbf{P}_i^{(1)}\}_{i\in m}$, and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$. This means the public map can be generated from just $\mathsf{seed}_{\mathsf{pk}}$ and $\{\mathbf{P}_i^{(3)}\}_{i\in m}$. This is referred to as the compacted public key.

Algorithm 1 MAYO.Gen()

```
Output: Compacted public key, secret key pair $(\mathsf{cpk}, \mathsf{csk})$
1: $\mathsf{seed_{sk}} \xleftarrow{\$} \mathcal{B}^{\mathsf{sk\_seed\_bytes}}$
2: $(\mathsf{seed_{pk}}, \mathbf{O}) \leftarrow \mathsf{SHAKE256}(\mathsf{seed_{sk}}, \mathsf{pk\_seed\_bytes} + \mathsf{O\_bytes})$
3: $(\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m}) \leftarrow \mathsf{AES\text{-}128\text{-}CTR}(\mathsf{seed_{pk}}, \mathsf{P1\_bytes} + \mathsf{P2\_bytes})$
4: $\{\mathbf{P}_i^{(3)}\}_{i\in m} \leftarrow \mathsf{ComputeP3}(\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m}, \mathbf{O})$
5: $\mathsf{pk} \leftarrow \mathsf{seed_{pk}} \mathbin\Vert \{\mathbf{P}_i^{(3)}\}_{i\in m}$
6: $\mathsf{sk} \leftarrow \mathsf{seed_{sk}}$
7: return $(\mathsf{cpk}, \mathsf{csk})$
```

Algorithm 1 shows a simplified version of the generation algorithm. The encoding and decoding of types are all omitted for simplicity, and the computation of $\{\mathbf{P}_i^{(3)}\}_{i\in m}$ is simply replaced with the function ComputeP3. $\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$ are derived by AES-128-CTR which acts as a source of randomness. This doesn't need to be secure as both the input ($\mathsf{seed_{pk}}$) and output ($\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$) are public knowledge. AES-128-CTR returns the encryption of an empty message P1_bytes+P2_bytes long encrypted with the key $\mathsf{seed_{pk}}$ as described by the MAYO team [22]. This provides sufficient randomisation for us to treat $\{\mathbf{P}_i^{(1)}\}_{i\in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in m}$ to be sampled uniformly at random for this proof (given $\mathsf{seed_{sk}}$ is also sampled uniformly at random).

When signing, the trapdoor information is used to find a preimage of the target $\mathbf{t}$.

Algorithm 2 MAYO.Sign(esk, M)

```
Input: Expanded secret key $\mathsf{esk} \in \mathcal{B}^{\mathsf{esk\_bytes}}$, Message $\mathsf{M} \in \mathcal{B}^*$
Public Constant: $\mathbf{E} \in \mathbb{F}_q^{m\times m}$
Output: Optional Signature $\mathsf{sig} \in \mathcal{B}^{\mathsf{sig\_bytes}}$
1: $(\mathsf{seed_{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{L}_i\}_{i\in m}) \leftarrow \text{decodeEsk}(\mathsf{esk})$
2:
3: //Hash message, and derive salt and $\mathbf{t}$.
4: $\mathsf{M\_digest} \leftarrow \mathsf{SHAKE256}(M, \mathsf{digest\_bytes})$
5: $\mathsf{R} \leftarrow \mathcal{B}^{\mathsf{R\_bytes}}$
6: $\mathsf{salt} \leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \mathbin\Vert \mathsf{R} \mathbin\Vert \mathsf{seed_{sk}}, \mathsf{salt\_bytes})$
7: $\mathbf{t} \leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \mathbin\Vert \mathsf{salt}, \lceil m \log(q)/8 \rceil)$
8:
9: //Attempt to find a preimage for $\mathbf{t}$.
10: $\mathsf{ctr} \leftarrow 0, x \leftarrow \text{None}$
11: while $\mathsf{ctr} \leq 255 \wedge x \neq \text{None}$ do
12:     $(\mathbf{v}, \mathbf{r}) \leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \mathbin\Vert \mathsf{salt} \mathbin\Vert \mathsf{seed_{sk}} \mathbin\Vert \mathsf{ctr}, k * \mathsf{v\_bytes} + \lceil ko \log(q)/8 \rceil)$ //Derive $\mathbf{v}_i$ and $\mathbf{r}$.
13:     $(\mathbf{A}, \mathbf{y}) \leftarrow \text{BuildLinearSystem}(\mathbf{v}, \mathbf{t}, \{\mathbf{L}_i\}_{i\in m}, \{\mathbf{P}_i^{(1)}\}_{i\in m}, \mathbf{E})$ //Build linear system $\mathbf{Ax} = \mathbf{y}$.
14:     $\mathbf{x} \leftarrow \mathsf{SampleSolution}(\mathbf{A}, \mathbf{y}, \mathbf{r})$ //Try to solve the system, returns optional x.
15:     $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
16:
17: $\mathsf{sig} \leftarrow$ if $\mathbf{x} = \text{None}$ then None else $\text{CalculateS}(\mathbf{x}, \mathbf{v}, \mathbf{O}) \mathbin\Vert \mathsf{salt}$
18: return sig
```

Algorithm 2 shows a simplified version of the MAYO signing function. Again, all encoding and decoding are omitted for simplicity, and large sections of deterministic code which are not involved in the security proof are replaced with functions.

Simply, line 1 extracts the public map $\mathcal{P}^\star$ from the secret key sk, as well as the private trapdoor information $\{\mathbf{L}_i\}_{i\in m}$ and $\mathbf{O}$. Lines 4 - 7 derive a random salt (salt) and the value of the target value $\mathbf{t}$. Lines

10 - 15 attempt to find a preimage $\mathbf{s}$ such that $\mathcal{P}^\star(\mathbf{s}) = \mathbf{t}$. Line 17 forms a signature sig if a preimage was found.
In order for a message, signature pair $(\mathsf{M}, \mathbf{s} \parallel \mathsf{salt})$ to verify, it must be that:

$$\mathcal{P}^\star(\mathbf{s}) = \mathsf{SHAKE256}(\mathsf{SHAKE256}(\mathsf{M}, \mathsf{digest\_bytes}) \parallel \mathsf{salt}, \lceil m \log(q)/8 \rceil) \tag{2.3}$$

Algorithm 3 MAYO.Verf(epk, M, sig)

```
Input: Expanded public key epk ∈ $\mathcal{B}^{\mathsf{epk\_bytes}}$, Message M ∈ $\mathcal{B}^*$, Signature sig ∈ $\mathcal{B}^{\mathsf{sig\_bytes}}$
Public Constant: $\mathbf{E} \in \mathbb{F}_q^{m\times m}$
Output: Boolean to indicate if (M, sig) is valid for epk
1: $(\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m}, \{\mathbf{P}_i^{(3)}\}_{i\in m}) \leftarrow \mathsf{epk}$
2: $(\mathbf{s}, \mathsf{salt}) \leftarrow \mathsf{sig}$
3: $\mathsf{M\_digest} \leftarrow \mathsf{SHAKE256}(\mathsf{M}, \mathsf{digest\_bytes})$
4: $\mathbf{t} \leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{salt}, \lceil m\log(q)/8\rceil)$
5: $\mathbf{y} = \mathsf{ComputePStar}(\mathbf{s}, \mathbf{E}, \{\mathbf{P}_\mathsf{i}^{(1)}\}_{\mathsf{i\in m}}, \{\mathbf{P}_\mathsf{i}^{(2)}\}_{\mathsf{i\in m}}, \{\mathbf{P}_\mathsf{i}^{(3)}\}_{\mathsf{i\in m}})$
6: return $\mathbf{y} = \mathbf{t}$
```

Once again the encoding and decoding of types are omitted for simplicity. The mathematics calculating the value of $\mathcal{P}^\star(\mathbf{s})$ is replaced with the function ComputePStar.

2.4 Definitions and Lemmas

Below we outline some definitions used in the following proof.
fresh - A value is fresh if it has not interacted with, or the result of an interaction, with a signing oracle. E.g. a message is fresh if it has not been sent to the signing oracle, and a salt is fresh if it has not been returned as output from a signing oracle. A pair (x, y) is fresh if x or y is fresh.

Lemma 2.4.1 (Lemma 1 from the MAYO specification [22]). *For $0 \le i \le j < k$, let the $\mathbf{E}_{ij} \in \mathbb{F}_q^{m\times m}$ be matrices such that*

$$\mathbf{E} = \begin{pmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} & \dots & \mathbf{E}_{1k} \\ \mathbf{E}_{12} & \mathbf{E}_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{E}_{1k} & \dots & \dots & \mathbf{E}_{kk} \end{pmatrix}$$

is nonsingular. If $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}, \mathcal{P} \in \mathrm{MQ}_{n,m,q}(\mathbf{O})$ and $\{\mathbf{v}_i\}_{i\in[k]}$ in $\mathbb{F}_q^{n-m} \times \{0\}^m$ are chosen uniformly at random, then as a function of $\{\mathbf{o}_i\}_{i\in[k]} \in O$ the affine map

$$\mathcal{P}^\star(\mathbf{v}+\mathbf{o}) = \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{v}_i + \mathbf{o}_i) + \sum_{1\le i<j\le k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{v}_i + \mathbf{o}_i, \mathbf{v}_j + \mathbf{o}_k)$$

has full rank except with probability bounded by $\frac{q^{k-(n-o)}}{q-1} + \frac{q^{m-ko}}{q-1}$. This is commonly represented as the variable B

Definition 2.4.1 (OV problem (Definition 1 from the MAYO specification [22])). *For $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}$, let $\mathsf{MQ}_{n,m,q}(\mathbf{O})$ denote the set of multivariate maps $\mathcal{P} \in \mathsf{MQ}_{n,m,q}$ that vanish on the rowspace of $\begin{pmatrix} \mathbf{O}^\top & \mathbf{I}_o \end{pmatrix}$. The OV problem asks to distinguish a random multivariate quadratic map $\mathcal{P} \in \mathsf{MQ}_{n,m,q}$ from a random multivariate quadratic map in $\mathsf{MQ}_{n,m,q}(\mathbf{O})$ for a random $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}$. Let $\mathcal{A}$ be an OV distinguisher algorithm. We say the distinguishing advantage of $\mathcal{A}$ is:*

$$\mathsf{Adv}^{\mathsf{OV}}_{n,m,o,q}(\mathcal{A}) = \left| \mathsf{Pr}\left[\mathcal{A}(\mathcal{P}) = 1 \mid \mathcal{P} \xleftarrow{\$} \mathsf{MQ}_{m,n,q}\right] - \mathsf{Pr}\left[\mathcal{A}(\mathcal{P}) = 1 \;\middle|\; \begin{array}{c} \mathbf{O} \xleftarrow{\$} \mathbb{F}_q^{(n-o)\times o} \\ \mathcal{P} \xleftarrow{\$} \mathsf{MQ}_{n,m,q}(\mathbf{O}) \end{array}\right] \right|$$

Chapter 3

Mayo Security Proof

The proof shown in this work is based on the security proof outlined in the MAYO specification [22]. It is outlined in three sections:

In section 3.1 we discuss a MAYO variant MAYO$^-$, which allows us to reason about the security surrounding the optional randomisation parameter R.

In section section 3.2 we prove a tightened bound on EUF-CMA security for MAYO$^-$. This follows a similar set of steps to the EUF-CMA proof described by the MAYO team.

In section 3.3 we include the proof of EUF-KOA exactly as outlined in the MAYO specification. This is included for completeness.

Finally, in section 3.4 we discuss a proof of SUF-CMA for MAYO$^-$

For simplicity, deterministic sections of code, which are unaffected by the proof, are replaced with functions. This allows the proof to be completed without proving the correctness of the underlying mathematics used in MAYO. This does not affect the security of the scheme, as the functions exactly replicate the deterministic behaviour of the calculations they replace. Additionally, the parameters n, m, o, k, and q are omitted when describing the adversaries' advantages as they remain unchanged in the reductions made.

Oracle Cloning

In the MAYO signing algorithm, the $\mathsf{SHAKE256}$ hash function is used in five different places:

$$\begin{aligned}
\mathsf{seed}_{\mathsf{pk}} &\leftarrow \mathsf{SHAKE256}(\mathsf{seed}_{\mathsf{sk}}) \\
\mathsf{M\_digest} &\leftarrow \mathsf{SHAKE256}(\mathsf{M}) \\
\mathsf{salt} &\leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{R} \parallel \mathsf{seed}_{\mathsf{sk}}) \\
\mathbf{t} &\leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{salt}) \\
(\mathbf{v}, \mathbf{r}) &\leftarrow \mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{salt} \parallel \mathsf{seed}_{\mathsf{sk}} \parallel \mathsf{ctr})
\end{aligned}$$

In the security proof outlined by the MAYO team, the $\mathsf{SHAKE256}$ function is treated as a single random oracle. In this work, we modify the proof to treat each usage of the $\mathsf{SHAKE256}$ function as a distinct random oracle. This simplifies the reasoning behind probabilities and makes changes to the oracles more modular. This is only possible however if each use of the $\mathsf{SHAKE256}$ oracle is fully independent from the others [2]. One way to achieve this is to ensure the length of inputs to $\mathsf{SHAKE256}$ are distinct for each use case.

$$\mathsf{len}(\mathsf{seed}_{\mathsf{sk}}) \neq \mathsf{len}(\mathsf{M}) \neq \mathsf{len}(\mathsf{M\_digest}) + \mathsf{len}(\mathsf{R}) + \mathsf{len}(\mathsf{seed}_{\mathsf{sk}}) \neq \dots$$

This is true across all five usages of $\mathsf{SHAKE256}$[1], except for when deriving $\mathsf{M\_digest}$, as the adversary has full control over the length of queries. For this reason, we add an extra requirement in the security proof that messages must be larger than $\mathsf{digest\_bytes} + 2 \cdot \mathsf{sk\_seed\_bytes} + 1$ bytes long for the security proof to hold.

[1]This is true for all possible parameter sets because of $\mathsf{sk\_seed\_bytes} = \mathsf{salt\_bytes} = \mathsf{R\_bytes}$.

With this assumption, we can treat each usage of SHAKE256 as a separate random oracle.

$$\begin{gathered}
\mathsf{SHAKE256}(\mathsf{seed_{sk}}) \sim \mathcal{K}(\mathsf{seed}) \\
\mathsf{SHAKE256}(\mathsf{M}) \sim \mathcal{G}(\mathsf{M}) \\
\mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{R} \parallel \mathsf{seed_{sk}}) \sim \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}) \\
\mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{salt}) \sim \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt}) \\
\mathsf{SHAKE256}(\mathsf{M\_digest} \parallel \mathsf{salt} \parallel \mathsf{seed_{sk}} \parallel \mathsf{ctr}) \sim \mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})
\end{gathered}$$

$\mathcal{L}$ is used to denote the random oracle which represents the AES-128-CTR function used to derive $\{\mathbf{P}_i^{(1)}\}_{i \in m}$ and $\{\mathbf{P}_i^{(2)}\}_{i \in m}$ during key generation and public key expansion.

3.1 MAYO^{-} and optional randomisation R

The randomisation parameter R is included in MAYO to help prevent fault injection attacks, where an attacker causes an error during signing to gain advantage over the system. The randomisation is optional and should not affect the security bound relating to the CMA or KOA security notions, as these do not cover fault injection attacks. To validate this, we prove the scheme is still secure even if the randomisation R is picked by the adversary.

In this proof, we prove a modified version of the MAYO protocol to the one outlined in the MAYO specification. The signing function is changed, allowing the randomisation bits R to be provided along with the message. This ensures that the randomisation is optional and does not affect the resulting bound as the adversary is given full control over its value.

Lemma 3.1.1. *Suppose there exists an adversary $\mathcal{A}$ that runs in time T against the EUF-CMA security of the MAYO signature in the random oracle model which makes Q_h queries to the random oracle and Q_s queries to the signing oracle. Then there exists an adversary $\mathcal{B}$ against the EUF-CMA security of the MAYO^{-} signature that runs in time T with:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{MAYO}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{MAYO^{-}}(\mathcal{B})$$

```
MAYO.$\mathcal{S}$(M)
--------------------------------
M_digest ← $\mathcal{G}(M)$
R $\xleftarrow{\$}$ $\mathcal{R}$
salt ← $\mathcal{H}$(M_digest, R, seed$'_{\mathsf{sk}}$)
// { Omitted for brevity }
return sig
```

```
MAYO$^{-}$.$\mathcal{S}$(M, R)
--------------------------------
M_digest ← $\mathcal{G}(M)$

salt ← $\mathcal{H}$(M_digest, R, seed$'_{\mathsf{sk}}$)
// { Omitted for brevity }
return sig
```

Figure 3.1: A side by side comparison of the signing oracles for MAYO and MAYO^{-}. Differences are highlighted for clarity.

```
MAYO.$\mathcal{S}$(M) (simulated)
--------------------------------
R $\xleftarrow{\$}$ $\mathcal{B}^{\mathsf{R\_bytes}}$
sig ← MAYO$^{-}$.$\mathcal{S}$(M, R)
return sig
```

Figure 3.2: MAYO.$\mathcal{S}$(M) (simulated) shows how the signing oracle for MAYO^{-} can be used to create a signing oracle for MAYO.

The proof proceeds as follows:

The only difference between MAYO and MAYO^{-} is the signing oracle $\mathcal{S}$. In MAYO^{-} the adversary $\mathcal{A}$ provides the randomisation R which is used to derive salt, whereas in MAYO the randomisation R is sampled uniformly at random (Figure 3.1).

The signing oracle used in the MAYO scheme can be perfectly simulated using the MAYO^{-} signing oracle by randomly sampling the value of R and passing it with the message to the MAYO^{-} signing oracle (Figure 3.2). Additionally, the winning conditions in both EUF-CMA games are the same, so any valid forgery for the MAYO scheme is also a valid forgery for MAYO^{-}.

Therefore an adversary $\mathcal{B}$ with access to an adversary $\mathcal{A}$ who can solve the MAYO EUF-CMA problem can solve the MAYO^{-} EUF-CMA problem.

Therefore:

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{MAYO}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{MAYO}^{-}}(\mathcal{B})$$

□

3.2 MAYO$^-$ EUF-CMA proof

First, we transform our original CMA game into another game (Game_5) which an adversary can win with similar probability through a series of steps.

Game_5 is constructed so that the oracle outputs are independent of the secret key used in the original game. This allows us to use it in a reduction, where an adversary can win a key-only attack game (KOA) by simulating Game_5. As $\mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}(\mathcal{A})$ is thought to be small (and itself is addressed in section 3.3) it implies $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A})$ must also be small.

The random oracles $\mathcal{K}$ and $\mathcal{L}$ are omitted as they remain unmodified for all games in this section. Full versions of the games can be found in Appendix A

Lemma 3.2.1. *Suppose there exists an adversary $\mathcal{A}$ that runs in time T against the EUF-CMA security of the MAYO$^-$ signature in the random oracle model which makes Q_h queries to the random oracle and Q_s queries to the signing oracle, with the length of all messages strictly greater than $\mathsf{digest\_bytes} + 2 \cdot \mathsf{sk\_seed\_bytes} + 1$ bytes long. Let $\mathsf{B} = \frac{q^{k-(n-o)}}{q-1} + \frac{q^{m-ko}}{q-1}$ derived from Lemma 2.4.1 and suppose $Q_s\mathsf{B} < 1$, then, there exists an adversary $\mathcal{B}$ against the EUF-KOA security of the MAYO$^-$ signature scheme that runs in time $T + O(Q_h + Q_s)$ with:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{MAYO^-}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{MAYO^-}(\mathcal{B})(1 - Q_s\mathsf{B})^{-1} + Q_hQ_s \cdot 2^{-8\mathsf{salt\_bytes}} + \\ 3Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} + \binom{Q_h + Q_s + 1}{2} \cdot 2^{-8\mathsf{digest\_bytes}}$$

The proof proceeds as follows:

3.2.1 Initial Setup - Game_0

First, we construct Game_0 (Figure 3.3), which represents the adversary $\mathcal{A}$'s advantage against MAYO in the EUF-CMA game. This precisely reflects the EUF-CMA security notion so we can therefore conclude:

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{MAYO}^-}(\mathcal{A}) = \Pr[\mathsf{Game}_0() = 1] \tag{3.1}$$

```
$\mathsf{Game}_0(\mathcal{A})$
    $\mathcal{S}(\mathsf{M}, \mathsf{R})$
        $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
        $\mathsf{sig} \leftarrow \mathrm{MAYO}^-_{\mathcal{G},\mathcal{H},\mathcal{I},\mathcal{J}}.\mathsf{Sign}(\mathsf{sk}, \mathsf{M}, \mathsf{R})$
        return $\mathsf{sig}$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathrm{MAYO}^-.\mathsf{Gen}()$
    $Q_s \leftarrow \{\}$
    $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H},\mathcal{I},\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
    return $\mathsf{M}^* \notin Q_s \wedge \mathrm{MAYO}^-_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*)$
```

Figure 3.3: EUF-CMA Game played by an adversary $\mathcal{A}$. Q_s is the log of messages signed by the signing oracle. $\mathcal{G}, \mathcal{H}, \mathcal{I}, \mathcal{J}$ are random oracles used in the signing process.

3.2.2 Deriving salt independently of $\mathsf{seed_{sk}}$ - Game_1

Game_1 (Figure 3.4) modifies how signing queries are handled by using a randomly generated $\mathsf{seed}'_{\mathsf{sk}}$ instead of $\mathsf{seed_{sk}}$ when deriving salt. $\mathsf{seed}'_{\mathsf{sk}}$ is generated during key generation and is randomly sampled from $\mathcal{B}^{\mathsf{sk\_seed\_bytes}}$.

```
$\mathsf{Game}_1(\mathcal{A})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$
        // { Omitted for brevity }
        $\mathsf{M\_digest} \leftarrow \mathcal{G}(M)$
        $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$
        // { Omitted for brevity }
        return $\mathsf{sig}$

    $\mathcal{H}'(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed})$
        if $\mathsf{seed} = \mathsf{seed_{sk}} \vee \mathsf{seed} = \mathsf{seed}'_{\mathsf{sk}}$ then
            $\mathsf{bad}_1 \leftarrow \mathsf{True}$
        return $\mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed})$

    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
    $\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
    $Q_s \leftarrow \{\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{False}$
    $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I},\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
    return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1$
```

Figure 3.4: Game_1 played by an adversary $\mathcal{A}$. The highlighted line shows where $\mathsf{seed}'_{\mathsf{sk}}$ is used instead of $\mathsf{seed_{sk}}$ for deriving the salt. $\mathcal{H}'$ is provided to the adversary and acts as a wrapper around the random oracle $\mathcal{H}$.

As $\mathsf{seed_{sk}}$ and $\mathsf{seed}'_{\mathsf{sk}}$ are sampled from the same distribution, $\mathcal{A}$ cannot distinguish between the two games unless she makes a query to the hashing oracle of the form $(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed_{sk}})$ or $(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$. This would allow her to compare the salt generated during signing and the salt output by the hashing oracle, allowing her to distinguish between the two programs. We capture the chance of this happening with the event bad_1.

The chance of the adversary randomly including either seed in a hash query is $2 \cdot 2^{-8\mathsf{sk\_seed\_bytes}}$ as both $\mathsf{seed_{sk}}$ and $\mathsf{seed}'_{\mathsf{sk}}$ are sampled uniformly at random. $\mathcal{A}$ makes Q_h queries to the hashing oracle so the probability of bad_1 occurring is therefore at most $2Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}}$.

If bad_1 doesn't occur then the adversary's view over the random oracles is identical. Therefore we have:

$$\Pr[\mathsf{Game}_1() = 1] \geq \Pr[\mathsf{Game}_0() = 1] - 2Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} \tag{3.2}$$

3.2.3 Deriving t during signing - Game_2

Game_2 is the same as Game_1 but modifies the random oracle so that $\mathbf{t}$ can be sampled during signing. It also modifies the game such that signatures output by the signing oracle must have had the corresponding $\mathbf{t}$ sampled during signing (which is required in later games). This step is split into two smaller steps, $\mathsf{Game}_{1.5}$ (Figure 3.5) and Game_2 (Figure 3.6), for simplicity, and numbered so that Game_2 still corresponds to Game_2 in the MAYO specification [22].

$\mathsf{Game}_{1.5}$ introduces a $\mathsf{SigCache}$ which stores triples $(\mathbf{t}, \mathsf{sig}, \mathsf{fromOracle})$. If a $(\mathsf{M\_digest}, \mathsf{salt})$ pair has already been signed in $\mathcal{S}$, then the signature is fetched from $\mathsf{SigCache}$. Otherwise, the signature is computed as

```
$\mathsf{Game}_{1.5}(\mathcal{A})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$

    // { Omitted for brevity - calculates M_digest and salt }
    if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ then
        if $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle}$ then
            $\mathsf{bad}_2 \leftarrow \mathsf{True}$
        return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$
    $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$

    // { Omitted for brevity - calculates sig }
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
    return sig

    $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$

    if $(\mathsf{M\_digest}, \mathsf{salt}) \notin \mathsf{SigCache}$ then
        $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{None}, \mathrm{fromOracle} : \mathsf{True})$
    return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I},\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2$
```

Figure 3.5: $\mathsf{Game}_{1.5}$ played by an adversary $\mathcal{A}$. SigCache is used to store the signature relating to $(\mathsf{M\_digest}, \mathsf{salt})$ pairs. $\mathcal{H}'$ is the same as in Game_1 but is omitted for brevity.

usual and saved. This is only possible as, after deriving $(\mathsf{M\_digest}, \mathsf{salt})$, the signing process is deterministic. SigCache must also save the value of $\mathbf{t}$, to provide consistency with the $\mathcal{I}'$ random oracle.

$\mathcal{I}'$ must also save to SigCache for consistency when signing $(\mathsf{M\_digest}, \mathsf{salt})$ pairs. As it does not produce a signature, a flag is set and no signature is saved in the triple. This introduces the only distinguishing behaviour between Game_1 and Game_2. In the case where a $\mathcal{S}(\mathsf{M}, \mathsf{R})$ derives a $(\mathsf{M\_digest}, \mathsf{salt})$ pair which was first seen by the $\mathcal{I}'$ random oracle, no signature is returned. This is captured with the event bad_2. For all signatures output by $\mathcal{S}$ with bad_2 not occurring, it must be that $\mathbf{t}$ was derived in $\mathcal{S}$.

We bound the probability of bad_2 occurring as follows. As the adversary makes at most Q_h queries to $\mathcal{I}'$, there are at most Q_h entries in SigCache with fromOracle set to True. During signing, the value of salt is uniformly sampled and unpredictable (unless bad_1 occurred in which case the game is already lost). Therefore the probability of a $(\mathsf{M\_digest}, \mathsf{salt})$ pair being already in the SigCache is at most $Q_h \cdot 2^{-8\mathsf{salt\_bytes}}$. Therefore across all Q_s signings, the probability of bad_2 occurring is at most $Q_s Q_h \cdot 2^{-8\mathsf{salt\_bytes}}$. This gives $\Pr[\mathsf{Game}_{1.5}() = 1] \geq \Pr[\mathsf{Game}_1]() = 1] - Q_s Q_h \cdot 2^{-8\mathsf{salt\_bytes}}$.

The only change between $\mathsf{Game}_{1.5}$ and Game_2 is how the value of $\mathbf{t}$ is calculated. In $\mathsf{Game}_{1.5}$ $\mathbf{t}$ is calculated from the $\mathcal{I}$ oracle but in Game_2 $\mathbf{t}$ is sampled uniformly at random. As this is the same output distribution of $\mathcal{I}$ the values of $\mathbf{t}$ are indistinguishable. Further once $\mathbf{t}$ is randomly sampled its value is saved in SigCache so future calls to $\mathcal{I}'$ are consistent. Suppose the adversary had previously queried $\mathcal{I}'$ with $(\mathsf{M\_digest}, \mathsf{salt})$ then she could distinguish between the values of $\mathbf{t}$. However, this event is already captured by bad_2. Therefore we have $\Pr[\mathsf{Game}_2() = 1] = \Pr[\mathsf{Game}_{1.5}() = 1]$.

```
$\mathsf{Game}_2(\mathcal{A})$

$\mathcal{S}(\mathsf{M}, \mathsf{R})$
// { Omitted for brevity - derives $\mathsf{M\_digest}$ and $\mathsf{salt}$ }
if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ then
    if $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle}$ then
        $\mathsf{bad}_2 \leftarrow \mathsf{True}$
    return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$
$\mathbf{t} \xleftarrow{\$} \mathcal{T}$
// { Omitted for brevity - calculates sig }
$\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
return $\mathsf{sig}$

$\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$
if $(\mathsf{M\_digest}, \mathsf{salt}) \notin \mathsf{SigCache}$ then
    $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{None}, \mathrm{fromOracle} : \mathsf{True})$
return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I}',\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2$
```

Figure 3.6: Game_2 played by an adversary $\mathcal{A}$. $\mathsf{SigCache}$ maps $(\mathsf{M\_digest}, \mathsf{salt})$ pairs to the triple $(\mathbf{t}, \mathsf{sig}, \mathsf{fromOracle})$. $\mathcal{H}'$ is the same as in Game_1 but is omitted for brevity.

Combining these inequalities, we get:

$$\Pr[\mathsf{Game}_2() = 1] \geq \Pr[\mathsf{Game}_1]() = 1] - Q_s Q_h \cdot 2^{-8\mathsf{salt\_bytes}} \tag{3.3}$$

This improves upon the bound given by the MAYO team [22] where the probability of bad_2 occurring is at most $(Q_h + Q_s)Q_s \cdot 2^{-8\mathsf{salt\_bytes}}$. This is because in our construction of Game_2, bad_2 occurs based on which oracle adds $(\mathsf{M\_digest}, \mathsf{salt})$ pairs to $\mathsf{SigCache}$, rather than when None signatures are recalled from $\mathsf{SigCache}$. Game_2 as described in the MAYO spec also triggers a bad event when the same $(\mathsf{M\_digest}, \mathsf{salt})$ pair, for which a preimage cannot be found, is used in signing twice. The probability of this is small due to the randomisation R. However, this is not accounted for in the original bound.

3.2.4 Deriving $\mathbf{v}$ independently of $\mathsf{seed}_{\mathsf{sk}}$ - Game_3

Game_3 (Figure 3.7) replaces the random oracle $\mathcal{J}$ so that $\mathbf{v}$ is randomly sampled. Similar to Game_1, the adversary can only distinguish between this game and Game_2 by making a query of the form $(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}_{\mathsf{sk}}, \mathsf{ctr})$ to $\mathcal{J}$. Given $\mathsf{seed}_{\mathsf{sk}}$ has $\mathsf{8sk\_seed\_bytes}$ min-entropy, the probability of a message of this form being queried is less than or equal to $2^{-\mathsf{8sk\_seed\_bytes}}$. With Q_h queries made by the adversary, the probability of bad_3 occurring is at most $Q_h \cdot 2^{-\mathsf{8sk\_seed\_bytes}}$.

$$\Pr[\mathsf{Game}_3() = 1] \geq \Pr[\mathsf{Game}_2() = 1] - Q_h \cdot 2^{-\mathsf{8sk\_seed\_bytes}} \tag{3.4}$$

```
$\mathsf{Game}_3(\mathcal{A})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$
        // { Omitted for brevity }
        $\mathsf{ctr} \leftarrow 0, \mathbf{v} \leftarrow$ None
        while $\mathsf{ctr} \leq 255 \wedge \mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank do
            $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
            $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$

        $\mathbf{o} \xleftarrow{\$} O^k$
        $\mathbf{t} \leftarrow \mathcal{P}^\star(\mathbf{v} + \mathbf{o})$
        $\mathsf{sig} \leftarrow$ if $\mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank then None else $(\mathbf{v} + \mathbf{o}) \parallel \mathsf{salt}$
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
        return $\mathsf{sig}$

    $\mathcal{J}'(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$
        if $\mathsf{seed} = \mathsf{seed}_{\mathsf{sk}}$ then
            $\mathsf{bad}_3 \leftarrow$ True
        return $\mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$

    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
    $\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{\mathsf{8sk\_seed\_bytes}}$
    $Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$
    $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I}',\mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
    return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2 \wedge \neg\mathsf{bad}_3$
```

Figure 3.7: Game_3 played by an adversary $\mathcal{A}$. $\mathbf{v}$ and $\mathbf{o}$ now sampled uniformly at random. $\mathcal{H}'$ and $\mathcal{I}'$ are the same as in Game_2 but are omitted for brevity.

3.2.5 Preventing hash collisions on M_digest - Game$_4$

```
$\mathsf{Game}_4(\mathcal{A})$
    $\mathcal{S}(\mathsf{M}, \mathsf{R})$
        $\mathsf{M\_digest} \leftarrow \mathcal{G}'(M)$
        // { Omitted for brevity - calculate $\mathsf{sig}$ }
        return $\mathsf{sig}$
    $\mathcal{G}'(\mathsf{M})$
        $\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
        if $\mathsf{M\_digest} \in Q_h \wedge Q_h[\mathsf{M\_digest}] \neq \mathsf{M}$ then
            $\mathsf{bad}_4 \leftarrow \mathsf{True}$
        $Q_h[\mathsf{M\_digest}] \leftarrow \mathsf{M}$
        return $\mathsf{M\_digest}$
    $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
    $\mathsf{seed}'_{\mathsf{sk}} \overset{\$}{\leftarrow} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
    $Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$; $Q_h \leftarrow \{\}$
    $\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$; $\mathsf{bad}_4 \leftarrow \mathsf{False}$
    $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G}',\mathcal{H}',\mathcal{I}',\mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
    return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G}',\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2 \wedge \neg\mathsf{bad}_3 \wedge \neg\mathsf{bad}_4$
```

Figure 3.8: Game_4 played by an adversary $\mathcal{A}$. The game is lost if there is a hash collision on $\mathcal{G}'$. The highlighted code shows where the $\mathcal{G}$ oracle has been replaced with the $\mathcal{G}'$ random oracle. $\mathcal{H}'$, $\mathcal{I}'$, and $\mathcal{J}'$ are the same as in Game_3 and are omitted for brevity.

Game_4 (Figure 3.8) is the same as Game_3 but adds the extra condition that there are no hash collisions in the random oracle $\mathcal{G}$ which is used to derive M_digest. This occurs when two distinct messages M_1 and M_2 are both queried and $\mathsf{Hash}(M_1) = \mathsf{Hash}(M_2)$. This change is unobservable to the adversary as they have no view over the state of the game, and all random oracles' input and outputs are unchanged.

Throughout the game, there are at most (Q_h+Q_s+1) queries made to this random oracle. The additional $+1$ queries come from the verification step in Game_4, where the M_digest is calculated in order to check the provided pre-image is valid. This means there are at most $(Q_h + Q_s + 1)$ different messages queried, each of which must map to a distinct output for there to be no collisions. If there is a collision then there exists a pair (M_1, M_2) with $\mathsf{Hash}(M_1) = \mathsf{Hash}(M_2)$. As the output of $\mathcal{G}$ is uniformly distributed, the chance of collision between two messages is $2^{-8\mathsf{digest\_bytes}}$. There are $(Q_h + Q_s + 1)^2$ pairs of messages, but as the order of messages in the pair is inconsequential, the number of pairs in which a collision can occur can be tightened to $\binom{Q_h+Q_s+1}{2}$. This gives the bound:

$$\Pr[\mathsf{Game}_4() = 1] \geq \Pr[\mathsf{Game}_3() = 1] - \binom{Q_h + Q_s + 1}{2} \cdot 2^{-8\mathsf{digest\_bytes}} \tag{3.5}$$

This bound is only possible by assuming the random oracles are fully distinct. Consider the case where the output of the other random oracles could trigger a hash collision. The $\mathcal{H}$ and $\mathcal{J}$ random oracles would provide the adversary with no additional advantage, as the adversary would learn the value of $\mathcal{G}(\mathsf{M\_digest} \parallel \mathsf{R} \parallel \mathsf{seed}_{\mathsf{sk}})$ and $\mathcal{G}(\mathsf{M\_digest} \parallel \mathsf{salt} \parallel \mathsf{seed}_{\mathsf{sk}} \parallel \mathsf{ctr})$. Since these include $\mathsf{seed}_{\mathsf{sk}}$ (or $\mathsf{seed}'_{\mathsf{sk}}$), and the adversary cannot guess this value (else it would be captured by bad_1 or bad_3), this gives them no extra information. Signing also leaks the value of $\mathcal{I}(\mathsf{M\_digest}, \mathsf{salt}) = \mathcal{G}(\mathsf{M\_digest} \parallel \mathsf{salt})$. Therefore in the case where SHAKE256 is modelled as a single random oracle, each query to $\mathcal{S}$ leaks information about two calls to SHAKE256. This changes the number of distinct pair the adversary can observe to be at most $\binom{Q_h+2Q_s+1}{2}$.

```
Game_5($\mathcal{A}$)

    $\mathcal{S}$(M, R)
        // { Omitted for brevity }
        $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
        if $\mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank then
            $\mathsf{bad}_5 \leftarrow \mathsf{True}$
        // { Omitted for brevity - calculate sig }
        return sig

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$; $\mathsf{bad}_4 \leftarrow \mathsf{False}$; $\mathsf{bad}_5 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G}',\mathcal{H}',\mathcal{I}',\mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
return $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G}',\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge$
        $\neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2 \wedge \neg\mathsf{bad}_3 \wedge \neg\mathsf{bad}_4 \wedge \neg\mathsf{bad}_5$
```

Figure 3.9: Game_5 played by an adversary $\mathcal{A}$. The value of $\mathbf{v}$ is sampled only once rather than retrying until $\mathcal{P}^\star(\mathbf{v} + \cdot)$ is full rank.

3.2.6 Removing Hash-and-Sign with Retry - Game_5

Game_5 (Figure 3.9) is the same as Game_4 but samples $\mathbf{v}$ only once. This removes the signing oracle's output's dependency on sk as the value of $\mathbf{v}$ is independent of $\mathcal{P}^\star(\mathbf{v} + \cdot)$ having full rank. If $\mathcal{P}^\star(\mathbf{v} + \cdot)$ is full rank then Game_5 is identical to Game_4. We capture the event of this not happening with the variable bad_5. As this derivation always returns a signature, a distinguishing behaviour is introduced. To avoid this, a $(\mathsf{M\_digest}, \mathsf{salt})$ pair fails with the same probability standard signing fails (B^{256}) so that the output distributions are the same given bad_5 doesn't occur. Since the probability of $\mathcal{P}^\star(\mathbf{v}+\cdot)$ not being full rank for a randomly sampled $\mathbf{v}$ is B (by Lemma 2.4.1), the probability of any signing query triggering bad_5 is B. As the adversary makes at most Q_s queries, the probability of $\mathsf{bad}_5 = \mathsf{True}$ is bound by $Q_s\mathsf{B}$. By the law of total probability, we can derive the following equation:

$$\begin{aligned}
\Pr[\mathsf{Game}_5() = 1] &= \Pr[\mathsf{Game}_5() = 1 \mid \neg\mathsf{bad}_5]\Pr[\neg\mathsf{bad}_5] + \Pr[\mathsf{Game}_5() = 1 \mid \mathsf{bad}_5]\Pr[\mathsf{bad}_5] \\
&\geq \Pr[\mathsf{Game}_5() = 1 \mid \neg\mathsf{bad}_5]\Pr[\neg\mathsf{bad}_5] \\
&\geq \Pr[\mathsf{Game}_4() = 1]\Pr[\neg\mathsf{bad}_5] \\
&\geq \Pr[\mathsf{Game}_4() = 1](1 - Q_s\mathsf{B})
\end{aligned} \tag{3.6}$$

3.2.7 Reducing Game_5 to EUF-KOA

In this step, we show that an adversary $\mathcal{B}^{\mathcal{A}}$ can use an adversary $\mathcal{A}$ for solving the EUF-KOA game.

The signing oracle's output in Game_5 is fully independent of the secret key picked for the game so an adversary $\mathcal{B}$ can simulate it fully (Figure 3.10). The random oracles $\mathcal{G}$, $\mathcal{H}$, $\mathcal{J}$ provided to $\mathcal{B}$ are provided to $\mathcal{A}$ unmodified. $\mathcal{B}$ also constructs $\mathcal{I}'$ from the random oracle $\mathcal{I}$, identically to the $\mathcal{I}'$ described in Game_2. From adversary $\mathcal{A}$'s perspective these random oracles act identically to those of Game_5 as the only differences are game events (e.g. bad_1) which the adversary cannot view.

$\mathcal{A}$ wins Game_5 if it produces a forgery which is valid under the oracles $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}'$, and $\mathcal{J}$, if the message M^* is not sent to the signing oracle $\mathcal{S}(\cdot,\cdot)$, and there are no collisions in the random oracle $\mathcal{G}$. For the EUF-KOA game, the only condition required to win is for the forgery to be valid under the random oracles.

We argue that any $(\mathsf{M}^*, \mathsf{sig}^*)$ pair which wins in Game_5 also wins in the EUF-KOA game. The only case in which a pair produced by $\mathcal{A}$ is not valid in the EUF-KOA game, is if the value in $\mathsf{SigCache}$

```
$\mathcal{B}^{\mathcal{A}}(\mathsf{pk})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$
        $\mathsf{M\_digest} \leftarrow \mathcal{G}(M)$
        $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$

        if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ then
            return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$

        $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
        $\mathbf{o} \xleftarrow{\$} O^k$
        $\mathbf{t} \leftarrow \mathcal{P}^\star(\mathbf{v} + \mathbf{o})$
        $\mathsf{fails} \xleftarrow{\$} \mathsf{True}$ with probability $\mathsf{B}^{256}$

        $\mathsf{sig} \leftarrow$ if $\mathsf{fails}$ then None else $(\mathbf{v} + \mathbf{o}) \parallel \mathsf{salt}$
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
        return $\mathsf{sig}$

    $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$
        if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ then
            return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].t$
        $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{None}, \mathrm{fromOracle} : \mathsf{True})$
        return $\mathbf{t}$

    $\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
    $\mathsf{SigCache} \leftarrow \{\}$
    $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H},\mathcal{I}',\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
    if $\mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*)$ then
        return $(\mathsf{M}^*, \mathsf{sig}^*)$
    return None
```

Figure 3.10: EUF-KOA played by an adversary $\mathcal{B}$ with access to an adversary $\mathcal{A}$. If $\mathcal{A}$ can efficiently win Game_5 then $\mathcal{B}$ can use it to solve the EUF-KOA problem. $\mathcal{G}, \mathcal{H}, \mathcal{I}, \mathcal{J}$ are all random oracles provided to $\mathcal{B}$.

for the corresponding $(\mathsf{M\_digest}, \mathsf{salt})$ pair is set by the signing oracle $\mathcal{S}$. When it is not set by $\mathcal{S}$, its value is equal to that of $\mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$, so forms a valid forgery for EUF-KOA. For the value of $\mathsf{SigCache}(\mathsf{M\_digest}, \mathsf{salt})$ to be set in the signing oracle, either: M^* has already been queried; or M^* has a hash collision with another message M which has already been queried. In both cases, we reach a contrading on the winning conditions of Game_5. Therefore the forged pair $(\mathsf{M}^*, \mathsf{sig}^*)$ must be a valid pair for EUF-KOA. We can then conclude:

$$\mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}(\mathcal{B}^{\mathcal{A}}) \geq \Pr[\mathsf{Game}_5() = 1] \tag{3.7}$$

Combining Equation 3.1, Equation 3.2, Equation 3.3, Equation 3.4, Equation 3.5, Equation 3.6, and Equation 3.7, and assuming $Q_s\mathsf{B} < 1$ we can obtain the following:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathrm{MAYO}^-}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{\mathrm{MAYO}^-}(\mathcal{B})(1 - Q_s\mathsf{B})^{-1} + Q_hQ_s \cdot 2^{-8\mathsf{salt\_bytes}} +\\ 3Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} + \binom{Q_h + Q_s + 1}{2} \cdot 2^{-8\mathsf{digest\_bytes}}\end{aligned}$$

□

3.3 Bounding EUF-KOA

```
$\mathcal{B}(\mathcal{P})$ - $\mathsf{Game}_6$
    $\mathcal{L}'(\mathsf{seed})$
        if $\mathsf{seed} = \mathsf{seed}_{\mathsf{pk}}$ then
            return $(\{\mathbf{P}^{(1)}_i\}_{i\in m}, \{\mathbf{P}^{(2)}_i\}_{i\in m})$
        return $\mathcal{L}(\mathsf{seed})$
$(\{\mathbf{P}^{(1)}_i\}_{i\in m}, \{\mathbf{P}^{(2)}_i\}_{i\in m}, \{\mathbf{P}^{(3)}_i\}_{i\in m}) \leftarrow \mathcal{P}$
$\mathsf{seed}_{\mathsf{sk}} \leftarrow \mathcal{B}^{\mathsf{sk\_seed\_bytes}}$
$(\mathsf{seed}_{\mathsf{pk}}, \mathbf{O}) \leftarrow \mathcal{K}(\mathsf{seed}_{\mathsf{sk}})$
$\mathsf{pk} \leftarrow \mathsf{seed}_{\mathsf{pk}} \parallel \{\mathbf{P}^{(3)}_i\}_{i\in m}$
$(\mathsf{M}, \mathsf{s} \parallel \mathsf{salt}) \leftarrow \mathcal{A}^{\mathcal{K},\mathcal{L}',\mathcal{G},\mathcal{H},\mathcal{I},\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
$\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
return $\mathcal{P}^\star(\mathbf{s}) = \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
```

Figure 3.11: Oil and Vinegar distinguishing adversary $\mathcal{B}$. $\mathcal{K}$, $\mathcal{L}'$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$ and $\mathcal{J}$ are all lazily sampling random oracles simulated by $\mathcal{B}$.

This proof is identical to Lemma 3 as described in the MAYO specification [22] and is included for completeness.

Lemma 3.3.1. *Let A be an EUF-KOA adversary that runs in time T against the MAYO$^-$ signature in the random oracle model. Then, there exists an adversary $\mathcal{B}$ against the OV problem, and an adversary $\mathcal{B}'$ against the MTWMQ problem, that both run in time bounded by $T + O(Qh)$ such that:*

$$\mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{MAYO^-}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{OV}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{MTWMQ}}(\mathcal{B}')$$

First, we construct an adversary $\mathcal{B}^{\mathcal{A}}$ which can solve the OV distinguishing game (Figure 3.11). The adversary's advantage $\mathsf{Adv}^{\mathsf{OV}}(\mathcal{B})$ measures how dependent her output is on the map $\mathcal{P}$ containing an oilspace $\mathbf{O}$. $\mathcal{B}$ simulates the random oracles $\mathcal{K}$, $\mathcal{L}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$ and $\mathcal{J}$ as lazy random oracles. $\mathcal{L}'$ is created as a wrapper around $\mathcal{L}$ which, on input $\mathsf{seed}_{\mathsf{pk}}$, returns the encoding of $(\{\mathbf{P}^{(1)}_i\}_{i\in m}, \{\mathbf{P}^{(2)}_i\}_{i\in m})$. This construction introduces distinguishing behaviour with the $\mathcal{L}$ and $\mathcal{J}$ random oracles. However, this difference appears in both games with equal probability. Therefore it does not affect the remaining bound as the difference between the games is still the same.

$$\mathsf{Adv}^{\mathsf{OV}}_{n,m,o,q}(\mathcal{B}) = \left| \mathsf{Pr}\left[\underbrace{\mathcal{B}^{\mathcal{A}}(\mathcal{P}) = 1 \mid \mathcal{P} \xleftarrow{\$} \mathsf{MQ}_{m,n,q}}_{\mathsf{Game}_7} \right] - \mathsf{Pr}\left[\underbrace{\mathcal{B}^{\mathcal{A}}(\mathcal{P}) = 1 \,\middle|\, \begin{array}{c} \mathbf{O} \xleftarrow{\$} \mathbb{F}_q^{(n-o)\times o} \\ \mathcal{P} \xleftarrow{\$} \mathsf{MQ}_{n,m,q}(\mathbf{O}) \end{array}}_{\mathsf{Game}_6} \right] \right|$$

When $\mathcal{P}$ is created with an oilspace $\mathbf{O}$ we call the game Game_6. As $\mathcal{B}$ simulates the random oracles perfectly, it is indistinguishable from solving the EUF-KOA game. Therefore we have $\mathsf{Pr}[\mathsf{Game}_6() = 1] = \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}(\mathcal{A})$.

```
$\mathcal{B}'^{\mathcal{A}}(\mathcal{P})$ - $\mathsf{Game}_7$

    $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$

    if $(\mathsf{M\_digest}, \mathsf{salt}) \notin \mathsf{SigCache}$ then
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{i} : i, \mathsf{t} : \mathbb{F}_q^{m\times\mathbb{N}}[i])$
        $i \leftarrow i + 1$

    return $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

// { Omitted for brevity }
$(\mathsf{M}, \mathsf{s} \,\|\, \mathsf{salt}) \leftarrow \mathcal{A}^{\mathcal{K},\mathcal{L}',\mathcal{G},\mathcal{H},\mathcal{I}',\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
$\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
$I \leftarrow \mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{i}$
return $(I, \mathbf{s})$
```

Figure 3.12: $\mathcal{B}'$ using an adversary $\mathcal{A}$ to win the MTWMQ game.

Recall the definition of the Multi-Target Whipped MQ problem (Definition 2.3.1):

$$\mathrm{Adv}^{\mathrm{MTWMQ}}_{\{\mathbf{E}_{ij}\},n,m,k,q}(\mathcal{A}) = \Pr\left[\sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{s}_i) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j) = \mathbf{t}_I \,\middle|\, \begin{array}{c} \mathcal{P} \leftarrow \mathrm{MQ}_{n,m,q} \\ \{\mathbf{t}_i\} \leftarrow \mathbb{F}_q^{m\times\mathbb{N}} \\ (I, \mathbf{s}_1, \ldots, \mathbf{s}_k) \leftarrow \mathcal{A}^{t_i}(\mathcal{P}) \end{array} \right]$$

When $\mathcal{P}$ does not have an oilspace we call the game Game_7. We argue that Game_7 is equivalent to the Multi-Target Whipped MQ problem. We define an adversary $\mathcal{B}'$ who modifies how the random oracle $\mathcal{I}$ answers queries. Rather then sampling them randomly, each time a fresh $(\mathsf{M\_digest}, \mathsf{salt})$ pair is seen it is paired with a corresponding I and $\mathbf{t}_i$ from the MTWMQ problem. Then when a forgery is created by $\mathcal{A}$, the corresponding I can be recovered to create a valid solution $(I, \mathbf{s})$ for the MTWMQ problem. As the oracle $\mathcal{I}$ still has the same output distribution, it is indistinguishable from the game simulated by $\mathcal{B}$ to $\mathcal{A}$. As it solves the MTWMQ problem with the same probability of solving Game_6 we can conclude $\mathsf{Pr}[\mathsf{Game}_6() = 1] = \mathsf{Adv}^{\mathsf{MTWMQ}}(\mathcal{B}'^{\mathcal{A}})$. Therefore we can conclude:

$$\mathsf{Adv}^{\mathsf{OV}}(\mathcal{B}) = \left| \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{\mathrm{MAYO}^-}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{MTWMQ}}(\mathcal{B}') \right| \tag{3.8}$$

This can be rearranged to obtain our bound, let $a = \mathsf{Adv}^{\mathsf{OV}}(\mathcal{B})$, $b = \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{\mathrm{MAYO}^-}(\mathcal{A})$, and $c = \mathsf{Adv}^{\mathsf{MTWMQ}}(\mathcal{B}')$.

$$a = |b - c|$$
$$a = \begin{cases} b - c, & \text{if } b \geq c \\ c - b, & \text{otherwise} \end{cases}$$
$$b = \begin{cases} a + c, & \text{if } b \geq c \\ c - a, & \text{otherwise} \end{cases}$$

All advantages are positive as they represent a probability. Therefore:

$$\begin{aligned} b &\leq a + c \\ \mathsf{Adv}^{\mathsf{EUF\text{-}KOA}}_{\mathrm{MAYO}^-}(\mathcal{A}) &\leq \mathsf{Adv}^{\mathsf{OV}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{MTWMQ}}(\mathcal{B}') \end{aligned} \tag{3.9}$$

□

3.4 MAYO SUF-CMA proof

For MAYO to be SUF, it must be hard for an adversary to produce a fresh valid pair $(\mathsf{M}, \mathsf{sig})$. This means that either M is fresh (not sent to the signing oracle), or sig is fresh (not returned by the signing oracle).

This proof introduces a new hardness property:

Definition 3.4.1 (Multi-Target MAYO Secondary Preimage problem (MTMayoSec))**.** *For* $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}$*, let* $\mathsf{MQ}_{n,m,q}(\mathbf{O})$ *denote the set of multivariate maps* $\mathcal{P} \in \mathsf{MQ}_{n,m,q}$ *that vanish on the rowspace of* $\begin{pmatrix} \mathbf{O}^\top & \mathbf{I}_o \end{pmatrix}$*. For some matrices* $\{\mathbf{E}_{ij}\}_{1\le i\le j\le k} \in \mathbb{F}_{q^m}$*, given random* $P \in \mathsf{MQ}_{n,m,q}(\mathbf{O})$ *and access to a preimage oracle which produces an unbounded number of random preimages* $\mathbf{s}_i \in \mathbb{F}_q^{kn}$ *with* $\mathbf{s}_i = \mathbf{v}_i + \mathbf{o}_i$ *and* $\mathcal{P}^\star(\mathbf{v}_i + \cdot)$ *full rank for* $i \in \mathbb{N}$*, the multi-target MAYO secondary preimage problem asks to compute* $(I, \mathbf{s}')$*, such that*

$$\mathcal{P}^\star(\mathbf{s}_I) = \mathcal{P}^\star(\mathbf{s}') \wedge \mathbf{s}_I \neq \mathbf{s}'$$

Where

$$\mathcal{P}^\star\left(\mathbf{x}_1, \ldots, \mathbf{x}_k\right) := \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}\left(\mathbf{x}_i\right) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

Let $\mathcal{A}$ *be an adversary. We say that the advantage of* $\mathcal{A}$ *against the multi-target MAYO secondary preimage problem is:*

$$\mathrm{Adv}^{\mathrm{MTMayoSec}}_{\{\mathbf{E}_{ij}\},n,m,k,q}(\mathcal{A}) = \Pr\left[\mathcal{P}^\star(s_I) = \mathcal{P}^\star(s') \wedge \mathbf{s} \neq \mathbf{s}' \;\middle|\; \begin{array}{c} \mathcal{P} \xleftarrow{\$} \mathrm{MQ}_{n,m,q}(\mathbf{O}) \\ \{\mathbf{s}_i = \mathbf{v}_i + \mathbf{o}_i\} \xleftarrow{\$} \mathbb{F}_q^{kn\times\mathbb{N}} \textit{ with } \mathcal{P}^\star(\mathbf{v}_i + \cdot) \textit{ full rank} \\ (I, \mathbf{s}') \leftarrow \mathcal{A}^{\mathbf{s}_i}(\mathcal{P}) \end{array}\right]$$

Lemma 3.4.1. *Let* A *be an SUF-CMA adversary that runs in time* T *against the* $MAYO^-$ *signature in the random oracle model which makes* Q_h *queries to the random oracle and* Q_s *queries to the signing oracle. Then, there exists an adversary* $\mathcal{B}$ *against the EUF-CMA security of MAYO, and an adversary* $\mathcal{B}'$ *against the MTMayoSec problem, that both run in time bounded by* $T + O(Q_h + Q_s)$ *such that:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{MAYO^-}(\mathcal{A}) \le Q_h Q_s \cdot \left(2^{-8\mathsf{salt\_bytes}} + |\mathcal{M}|^{-1}\right) + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{MAYO^-}(\mathcal{B}) \,+ \\ 2Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} + \mathsf{Adv}^{\mathsf{MTMayoSec}}(\mathcal{B}')\end{aligned}$$

Consider an adversary $\mathcal{A}$ against the SUF-CMA problem of MAYO. For a forged message $(\mathsf{M}, \mathsf{s} \,\|\, \mathsf{salt})$ to win the SUF-CMA game it must be that either: one of M and salt are fresh; or both M and salt are not fresh. Therefore, by capturing the advantage as a game, it can be written as:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A}) = {} & \Pr[\mathsf{Game}_8() = 1] = \\ & \Pr[\mathsf{Game}_8() = 1 \wedge (\mathsf{M}\text{ fresh} \vee \mathsf{salt}\text{ fresh})] \,+ \\ & \Pr[\mathsf{Game}_8() = 1 \wedge \neg(\mathsf{M}\text{ fresh}) \wedge \neg(\mathsf{salt}\text{ fresh})]\end{aligned} \tag{3.10}$$

We then bound the probability of each of these cases occurring.

Case 1

Consider the case where $\mathcal{A}$ creates a successful SUF-CMA forgery with either M or salt fresh. We can construct an adversary $\mathcal{B}^{\mathcal{A}}$ who uses $\mathcal{A}$ to create EUF-CMA forgeries (Figure 3.13). The intuition behind this proof is that $\mathcal{I}(\mathsf{M\_digest} \,\|\, \mathsf{salt}')$ with fresh salt' is indistinguishable from $\mathcal{I}(\mathsf{M\_digest}' \,\|\, \mathsf{salt})$ with a fresh $\mathsf{M\_digest}'$.

$\mathcal{B}$ creates a wrapper around the provided oracle $\mathcal{I}$. $\mathcal{I}'$ is constructed such that a valid forgery where M or salt is fresh for the $\mathcal{I}'$ oracle can be used to create a valid EUF forgery for the $\mathcal{I}$ oracle.

$\mathcal{B}$ simulates the random oracle $\mathcal{I}'$ such that each $(\mathsf{M\_digest}, \mathsf{salt})$ pair corresponds to a pair $(\mathsf{M}^*, \mathsf{salt}^*)$ with fresh M^*. Then if $(\mathsf{M}, \mathsf{s} \,\|\, \mathsf{salt})$ are output as a forgery for $\mathcal{I}'$, then $(\mathsf{M}^*, \mathsf{s} \,\|\, \mathsf{salt}^*)$ is a valid forgery for $\mathcal{I}$, with a fresh M^* (which by definition is EUF). When a query is made with a new pair $(\mathsf{M\_digest}, \mathsf{salt})$ a fresh message M^* is sampled from the message space, and a random salt^* is generated. $\mathsf{M\_digest}^*$ is then derived from M^* and used to calculate $\mathbf{t} = \mathcal{I}(\mathsf{M\_digest}^*, \mathsf{salt}^*)$. The values of M^*, salt^*, and $\mathbf{t}$ are all saved in a Cache under the key of $(\mathsf{M\_digest}, \mathsf{salt})$. If a valid forgery under $\mathcal{I}'$ is created with $(\mathsf{M\_digest}, \mathsf{salt})$, it can be converted into a valid EUF forgery $(\mathsf{M}^*, \mathsf{s} \parallel \mathsf{salt}^*)$

The signing oracle $\mathcal{S}'$ acts as a wrapper around $\mathcal{S}$, faithfully singing (M, R) pairs. The resulting $(\mathsf{M\_digest}, \mathsf{salt})$ pair is saved in the Cache with $\mathsf{M}^* = \mathsf{salt}^* = \mathsf{None}$.

When a valid forgery is returned by $\mathcal{A}$, $\mathsf{M}^* = \mathsf{None}$ only occurs if the message's digest $\mathsf{M\_digest}$ has been used in signing before. For the forgery to be valid from $\mathcal{A}$'s perspective, M must be fresh. This means that $\mathcal{A}$ has found two messages with the same $\mathsf{M\_digest}$ and signed one using $\mathcal{S}'$. By storing the value of $\mathbf{s}$ output from $\mathcal{S}'$ in Cache, we can reuse $\mathbf{s}$ to create a forgery for the fresh message M.

There is only one case where a successful forgery does not win the EUF game. When $\mathcal{A}$ queries $\mathcal{S}'$ with M^* (as M^* would no longer be fresh). There are Q_h queries made to $\mathcal{I}'$, so there are at most Q_h fresh messages. There are Q_s messages queried by the adversary which could possibly collide. As the fresh messages are sampled uniformly at random, the chance of any queried M being equal to any "fresh" message is $Q_s Q_h |\mathcal{M}|^{-1}$. This is captured by the event bad_2.

We must also bound the probability of the adversary distinguishing between $\mathcal{B}$'s simulated game and the SUF game. If an adversary queries $\mathcal{S}'$ and the derived $(\mathsf{M\_digest}, \mathsf{salt})$ pair has already been queried in $\mathcal{I}'$, then the produced signature will not be a valid signature for the $\mathcal{I}'$ oracle. This is captured by the event bad_1. As salt is uniformly distributed and unpredictable, and there are at most Q_h values in Cache from the $\mathcal{I}'$ oracle, the overall probability of this happening is at most $Q_s Q_h \cdot 2^{-8\mathsf{salt\_bytes}}$.

In the case where $(\mathsf{M\_digest}, \mathsf{salt}) \notin \mathsf{Cache}$, the adversary hasn't queried the random oracle with $\mathsf{M\_digest}$ and salt (or used $\mathcal{S}'$ to sign a corresponding M which produces salt). The adversary is therefore guessing the value of $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$ which occurs with equal probability to guessing $\mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$ (as both outputs are uniformly distributed).

Therefore, we can conclude:

$$\Pr[\mathsf{Game}_8() = 1 \wedge (\mathsf{M} \text{ fresh} \vee \mathsf{salt} \text{ fresh})] - Q_s Q_h \cdot 2^{-8\mathsf{salt\_bytes}} - Q_s Q_h |\mathcal{M}|^{-1} \leq \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}^{\mathcal{A}}) \quad (3.11)$$

$\mathcal{B}^{\mathcal{A}}$ - Case 1

```
$\mathcal{S}'(\mathsf{M}, \mathsf{R})$
---
$(\mathbf{s} \,\|\, \mathsf{salt}) \leftarrow \mathcal{S}(\mathsf{M}, \mathsf{R})$
$\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{Cache} \wedge \mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t} \neq \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$ then
    $\mathsf{bad}_1 \leftarrow \mathsf{True}$
if $\mathsf{M} \in \mathsf{supp}(\mathsf{Cache})$ then
    $\mathsf{bad}_2 \leftarrow \mathsf{True}$
$\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{None}, \mathsf{None}, \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt}), \mathbf{s})$
return $(\mathbf{s} \,\|\, \mathsf{salt})$
```

```
$\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$
---
if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{Cache}$ then
    return $\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

$\mathsf{M}^* \xleftarrow{\$} \mathcal{M}$ such that $\mathsf{M}^*$ is fresh
$\mathsf{M\_digest}^* \leftarrow \mathcal{G}(\mathsf{M}^*)$
$\mathsf{salt}^* \xleftarrow{\$} \mathcal{B}^{\mathsf{salt\_bytes}}$
$\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}^*, \mathsf{salt}^*)$
$\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{M}^*, \mathsf{salt}^*, \mathbf{t}, \mathsf{None})$
return $\mathbf{t}$
```

```
$(\mathsf{M}, \mathbf{s} \,\|\, \mathsf{salt}) \leftarrow \mathcal{A}^{\mathcal{S}'(\cdot,\cdot),\mathcal{G},\mathcal{H},\mathcal{I}',\mathcal{J}}.\mathsf{Forge}()$      // creates forgeries with fresh M or fresh salt
$\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{Cache}$ then
    $(\mathsf{M}^*, \mathsf{salt}^*, \mathbf{t}, \mathbf{s}^*) \leftarrow \mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})]$
    if $\mathsf{M}^* = \mathsf{None}$ then
        return $(\mathsf{M}, \mathbf{s}^* \,\|\, \mathsf{salt})$        // hash collision found on M. M must be fresh
    return $(\mathsf{M}^*, \mathbf{s} \,\|\, \mathsf{salt}^*)$                        // create forgery with fresh M
else
    return $(\mathsf{M}, \mathbf{s} \,\|\, \mathsf{salt})$              // $\mathcal{A}$ is guessing the value of $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$
```

Figure 3.13: An adversary $\mathcal{B}$'s program which, using an adversary $\mathcal{A}$ who produces valid SUF forgeries with fresh $(\mathsf{M}, \mathsf{salt})$ pairs, creates valid EUF forgeries with high probability. $\mathcal{K}$, $\mathcal{L}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$, $\mathcal{J}$, and $\mathcal{S}$ are provided to the adversary $\mathcal{B}$ from a EUF-CMA game.

Case 2

For bounding the probability in the case of a forgery being produced with a fresh $\mathbf{s}$ we introduce the new hardness assumption MTMayoSec (Definition 3.4.1).

We construct an adversary $\mathcal{B}'$ which, when given access to $\mathcal{A}$ who outputs valid SUF forgeries with $\neg(\mathsf{M}\text{ fresh}) \wedge \neg(\mathsf{salt}\text{ fresh})$, can solve the MTMayoSec problem (Figure 3.14). This is done similarly to Game_6, where every time adversary $\mathcal{A}$ requests a fresh pair $(\mathsf{M\_digest}, \mathsf{salt})$ the next $\mathbf{s}_i$ is returned.

$\mathcal{B}$ picks $\mathsf{seed}_{\mathsf{sk}}$ at random and derives $\mathsf{seed}_{\mathsf{pk}}$. It then simulates the random oracles $\mathcal{K}$, $\mathcal{L}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$, $\mathcal{J}$ and the signing oracle $\mathcal{S}$ so that with high probability they are indistinguishable from the random oracles in the SUF-CMA game.

$\mathcal{L}$ is simulated as a lazily sampling random oracle, except that when the input is equal to $\mathsf{seed}_{\mathsf{pk}}$, the encoding of $(\{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{P}_i^{(2)}\}_{i\in m})$ is returned.

$\mathcal{I}$ is simulated such that for every fresh pair $(\mathsf{M\_digest}, \mathsf{salt})$, a preimage $\mathbf{s}_i$ is sampled and $\mathcal{P}^\star(\mathbf{s}_i)$ is returned. $(\mathbf{s}_i, \mathbf{t}_i)$ is saved in a cache for future use.

$\mathcal{S}$ derives $\mathsf{M\_digest}$, salt and $\mathbf{t}$ as in the MAYO signing oracle, using the random oracles $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{I}$. As $\mathbf{t}$ was derived using $\mathcal{I}$, its preimage $\mathbf{s}$ can be fetched from the cache. The signature $\mathbf{s} \,\|\, \mathsf{salt}$ is then returned. As this signing process is successful every time, an additional flag is stored in the cache signalling if the signing fails. This is set with probability B^{256}, so that $\mathcal{S}$ fails with the same probability as MAYO.$\mathsf{Sign}()$. $\mathcal{K}$, $\mathcal{L}$, $\mathcal{G}$, $\mathcal{H}$, and $\mathcal{J}$ are simulated as lazily sampling random oracles.

If $\mathcal{A}$ outputs a valid SUF forgery $(\mathsf{M}', \mathsf{s}' \,\|\, \mathsf{salt}')$ with $\neg(\mathsf{M}'\text{ fresh}) \wedge \neg(\mathsf{salt}'\text{ fresh})$ then is must be the case that s' is fresh and $\mathcal{P}^\star(\mathsf{s}') = \mathcal{I}(\mathcal{G}(\mathsf{M}'), \mathsf{salt}')$. By our construction, we can retrieve the preimage $\mathbf{s}$ for $\mathcal{I}(\mathcal{G}(\mathsf{M}'), \mathsf{salt}')$. In the case where $\mathbf{s} = \mathsf{s}'$ then $(\mathsf{M}', \mathsf{s}' \,\|\, \mathsf{salt}')$ is not a valid forgery as the signature must be output from the signing oracle (if it was output by $\mathcal{I}$ then $(\mathsf{M\_digest}, \mathsf{salt})$ was not seen by the signing oracle and this forgery would fall under **case 1**). Therefore $\mathbf{s}'$ must be a valid second preimage of $\mathbf{s}$, so $\mathcal{B}$ has solved the MTMayoSec problem.

$\mathcal{A}$ can distinguish between the game simulated by $\mathcal{B}$ and a valid SUF-CMA game in two ways. If $\mathcal{A}$ guesses the value of $\mathsf{seed}_{\mathsf{sk}}$ and queries $\mathcal{K}'$, then the derived oil space will differ from that of $\mathcal{P}$. Over the entire game, the probability of this occurring is at most $Q_h \cdot 2^{8\mathsf{sk\_seed\_bytes}}$. The step in MAYO.$\mathsf{Sign}$ when the vinegar variables are derived from a call to $\mathcal{J}$ is skipped in $\mathcal{B}$'s simulated game. An adversary can identify this and distinguish between the two games. This can only happen if they make a query of the form $(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}_{\mathsf{sk}}, \mathsf{ctr})$ to the random oracle $\mathcal{J}'$. As $\mathsf{seed}_{\mathsf{sk}}$ has $8\mathsf{sk\_seed\_bytes}$ bits of min-entropy, the probability of this occurring at any point of the game is at most $Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}}$. Therefore we can conclude:

$$\Pr[\mathsf{Game}_0() = 1 \wedge \neg(\mathsf{M}\text{ fresh}) \wedge \neg(\mathsf{salt}\text{ fresh})] - 2Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} \leq \mathsf{Adv}^{\mathsf{MTMayoSec}}(\mathcal{C}^{\mathcal{A}}) \tag{3.12}$$

Combining Equation 3.10, Equation 3.11 and Equation 3.12 we obtain:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}(\mathcal{A}) \leq Q_s Q_h \cdot \left(2^{-8\mathsf{salt\_bytes}} + |\mathcal{M}|^{-1}\right) + \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}(\mathcal{B}^{\mathcal{A}}) +\\ 2Q_h \cdot 2^{-8\mathsf{sk\_seed\_bytes}} + \mathsf{Adv}^{\mathsf{MTMayoSec}}(\mathcal{C}^{\mathcal{A}})\end{aligned} \tag{3.13}$$

□

```
$\mathcal{C}^{\mathcal{A}}$ - Case 2

$\mathcal{S}(\mathsf{M}, \mathsf{R})$
    $\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}_{\mathsf{sk}})$
    $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
    if $\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{FailsToSign}$ then
        return None
    $\mathbf{s} \leftarrow \mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{s}$
    return $(\mathbf{s} \parallel \mathsf{salt})$

$\mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$
    if $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{Cache}$ then
        return $\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

    $\mathsf{FailsToSign} \leftarrow \mathsf{True}$ with probability $\mathsf{B}^{256}$
    $\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{i} : i,\ \mathsf{s} : \mathbb{F}_q^{kn \times \mathbb{N}}[i],\ \mathsf{t} : \mathcal{P}^\star(\mathbb{F}_q^{kn \times \mathbb{N}}[i]),\ \mathsf{FailsToSign})$
    $i \leftarrow i + 1$
    return $\mathsf{Cache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{t}$

$\mathcal{L}'(\mathsf{seed})$
    if $\mathsf{seed} = \mathsf{seed}_{\mathsf{pk}}$ then
        return $(\{\mathbf{P}_i^{(1)}\}_{i \in m}, \{\mathbf{P}_i^{(2)}\}_{i \in m})$
    return $\mathcal{L}(\mathsf{seed})$

$\mathcal{J}'(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$
    if $\mathsf{seed} = \mathsf{seed}_{\mathsf{sk}}$ then
        $\mathsf{bad}_1 \leftarrow \mathsf{True}$
    return $\mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$

$\mathcal{K}'(\mathsf{seed})$
    if $\mathsf{seed} = \mathsf{seed}_{\mathsf{sk}}$ then
        $\mathsf{bad}_2 \leftarrow \mathsf{True}$
    return $\mathcal{K}(\mathsf{seed})$

$(\{\mathbf{P}_i^{(1)}\}_{i \in m}, \{\mathbf{P}_i^{(2)}\}_{i \in m}, \{\mathbf{P}_i^{(3)}\}_{i \in m}) \leftarrow \mathcal{P}$
$\mathsf{seed}_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{\mathsf{sk\_seed\_bytes}}$
$(\mathsf{seed}_{\mathsf{pk}}, \mathbf{O}) \leftarrow \mathcal{K}(\mathsf{seed}_{\mathsf{sk}})$
$\mathsf{pk} \leftarrow \mathsf{seed}_{\mathsf{pk}} \parallel \{\mathbf{P}_i^{(3)}\}_{i \in m}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}; \mathsf{bad}_2 \leftarrow \mathsf{False}$
// creates forgeries with $\neg(\mathsf{M}'$ fresh$) \wedge \neg(\mathsf{salt}'$ fresh$)$
$(\mathsf{M}', \mathsf{s}' \parallel \mathsf{salt}') \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot), \mathcal{K}', \mathcal{L}', \mathcal{G}, \mathcal{H}, \mathcal{I}, \mathcal{J}'}(\mathsf{pk})$
$\mathsf{M\_digest}' \leftarrow \mathcal{G}(\mathsf{M}')$
$i \leftarrow \mathsf{Cache}[(\mathsf{M\_digest}', \mathsf{salt}')].\mathsf{i}$                // corresponding $i$ with $\mathbf{s}_i$
return $(i, \mathbf{s}')$
```

Figure 3.14: An adversary $\mathcal{C}$ who, when given access to an adversary $\mathcal{A}$ who can win the SUF-CMA game with $\neg(\mathsf{M}$ fresh$) \wedge \neg(\mathsf{salt}$ fresh$)$, can win the MTMayoSec game. The random oracles $\mathcal{K}$, $\mathcal{L}$, $\mathcal{G}$, $\mathcal{H}$, $\mathcal{I}$, and $\mathcal{J}$ are all lazy sampling random oracles simulated by $\mathcal{C}$.

3.4.1 MTMayoSec reduction to SUF-CMA security of MAYO$^-$

Finally, we consider the case where there exists an adversary with a high advantage over the MTMayoSec problem and its impact on the SUF-CMA security of MAYO$^-$.

Lemma 3.4.2. *Let A be an MTMayoSec adversary that runs in time T in the random oracle model, which makes Q_p queries to the preimage oracle. Then, there exists an adversary $\mathcal{B}$ against the SUF-CMA security of MAYO$^-$ that runs in time bounded by $T + O(Q_p)$ such that:*

$$\mathsf{Adv}^{\mathsf{MTMayoSec}}(\mathcal{A}) \leq \ \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{MAYO^-}(\mathcal{B}) + Q_p \cdot \mathsf{B}^{256} + \binom{Q_p}{2} \cdot 2^{-8\mathsf{digest\_bytes}}$$

Consider a SUF-CMA forger $\mathcal{B}$ with access to an adversary $\mathcal{D}$ who can solve the MTMayoSec problem with probability $\mathsf{Adv}^{\mathsf{MTMayoSec}}(\mathcal{D})$, and makes at most Q_p queries to the preimage oracle. $\mathcal{B}$ simulates the preimage oracle such that every time $\mathcal{D}$ requests a preimage $\mathbf{s}_i$, $\mathcal{B}$ uses its signing oracle to calculate a s for a randomly chosen pair (M, R). Then, when $\mathcal{D}$ produces a solution to the MTMayoSec problem (I, s'), $\mathcal{B}$ can create a valid SUF forgery $(\mathsf{M}_I, \mathsf{s}' \,\|\, \mathsf{salt}_I)$, where M_I was the Ith message requested for signing and salt_I was output during that signing. This does introduce some distinguishing behaviour however. In the MTMayoSec problem, the adversary has access to a random oracle which returns preimages $\mathsf{s} \in \mathbb{F}_q^{kn}$. In the random oracle simulated by $\mathcal{B}$, the random oracle returns the preimage of $\mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$ with a chance of failing. If $\mathcal{B}$'s query to the signing oracle returns None then she cannot return a valid preimage in the preimage oracle. As a signing fails with probability at most B^{256}, and $\mathcal{D}$ makes Q_p queries to the preimage oracle, the probability of this event occurring is at most Q_pB^{256}. Further, as $\mathsf{M\_digest}$ and salt have a limited number of possible values, the probability of receiving the same s multiple times from the two oracles differs. This happens when two queries made to the signing oracle result in the same pair $(\mathsf{M\_digest}, \mathsf{salt})$ pair being sent to $\mathcal{I}$. As $\mathcal{B}$ makes one query to the signing oracle for each preimage oracle query, the probability of a collision occurring when deriving $\mathsf{M\_digest}$ is at most $\binom{Q_p}{2} \cdot 2^{-8\mathsf{digest\_bytes}}$.

□

Chapter 4

Conclusion

4.1 Contributions

This work achieves the following:

- We reason about the optional randomisation parameter R and its impact on security, demonstrating that it has no impact on the bound for EUF-CMA or SUF-CMA security given.
- Proven the bound for EUF-CMA security for the MAYO scheme and tightened it considerably.
- Outlined programs representing each game used in each proof, allowing for greater accessibility and understanding of each step.
- Presented the EUF-KOA proof outlined by the MAYO team along with supporting programs.
- Outlined a basic SUF-CMA proof by introducing a new hardness assumption and demonstrated this is required for MAYO$^-$ to achieve SUF-CMA security.

4.2 Future Works

4.2.1 Domain Separation

This work focuses on proving the security of MAYO with restricted message lengths rather than arbitrary message lengths.

The MAYO scheme could be modified so the SHAKE256 function can be modelled as five distinct random oracles. Oracle cloning can be achieved by ensuring each usage has a different prefix. This ensures that (input, output) pairs for one use of SHAKE256 give no usable information about another instantiation of SHAKE256 as the prefixes will not match. This could however impact the performance of the scheme. Each call to SHAKE256 would require an additional byte prepended to the input (in order to maintain byte alignment for efficiency) which, depending on the chosen parameters, could meaningfully impact the computation time of SHAKE256. The additional byte could be avoided by shortening M_digest and $\mathsf{seed}_{\mathsf{sk}}$ by one bit and using the first bit as a domain separator. As only the derivation of M_digest can overlap with the other oracles, the domains can be separated using a single bit. For example, in the call to derive salt, the first bit of the M_digest could be overwritten with a 1, ensuring it's separated from the call deriving M_digest (which would have a 0 prepended). This would however lead to weakened security, as the range of M_digest and $\mathsf{seed}_{\mathsf{pk}}$ values would be reduced, which in turn could require larger parameters (for some parameter sets), requiring an extra byte regardless.

4.2.2 Formalisation

Formalising cryptographic proofs involves using a proof assistant to validate that a scheme is secure under specific assumptions. When validating schemes, auditors can see all of the assumptions made for a proof to hold and all of the restrictions the result is conditional on. This provides greater confidence in the scheme. Further, sections of proof can be reused and modified to fit other schemes, allowing for the faster iteration of more reliable schemes.

This work started as a formalisation of the MAYO proof however understanding each step was too complex. Outlining each game in the proof is difficult and must be correct to prove security over the MAYO scheme.

It is only by looking from the perspective of creating a machine checked proof that some of the security loss omitted in the MAYO specification was identified. This is because many proof assistants use probabilistic relational Hoare logic to reason about the similarity of programs [1]. This requires output distributions to be strictly equal under specific conditions for games to be comparable, making it easy to identify uncaptured differences.

4.2.3 Furthering the SUF-CMA proof

The SUF-CMA proof outlined in section 3.4 does not allow the resulting bound to be trivially combined with the EUF-CMA bound proven in section 3.2. Additionally, some security loss introduced in the SUF-CMA proof has conditions similar to those seen in the EUF-CMA proof. It could be beneficial to construct a proof of SUF-CMA which relies on the hardness assumptions as this will provide a tighter bound and give greater confidence in the scheme.

4.2.4 Analysis of the MTMayoSec problem

The SUF-CMA proof for MAYO$^-$ introduces a new hardness assumption (MTMayoSec) and proves it must be hard if MAYO wishes to achieve SUF-CMA. It may be beneficial to explore the difficulty of solving this problem. It may be possible to construct an oil and vinegar distinguishing adversary using this hardness assumption and create a bound based on a new hardness problem (similar to Game_6 and Game_7 in the MAYO$^-$ EUF-CMA proof). In doing so this new assumption, could be made with $\mathcal{P} \overset{\$}{\leftarrow} \mathrm{MQ}_{n,m,q}$ and $\mathbf{s} \overset{\$}{\leftarrow} \mathbb{F}_q^{kn}$. This would better separate the whipping problem from the oil and vinegar problem. By constructing this proof in a more generic way, if an attack is found that targets the construction of $\mathcal{P}^\star$, it could be easily replaced. The majority of the security proof could remain unmodified as only the hardness assumptions would need changing.

4.2.5 Additional proofs

This work focuses only on proving EUF-KOA, EUF-CMA, and SUF-CMA of MAYO in the random oracle model. It would be beneficial to create additional proofs to provide greater confidence in the scheme. Including a proof of correctness, that the algorithms outlined in the MAYO specification [22] use the described theory, would ensure that signatures output during signing always verify. Further, it could be advantageous to explore the security of MAYO in the quantum random oracle model [6], where the random oracles can have quantum access.

Bibliography

[1] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. *International School on Foundations of Security Analysis and Design*, pages 146–166, 2012.

[2] Mihir Bellare, Hannah Davis, and Felix Günther. Separate your domains: Nist pqc kems, oracle cloning and read-only indifferentiability. Cryptology ePrint Archive, Paper 2020/241, 2020. https://eprint.iacr.org/2020/241. URL: https://eprint.iacr.org/2020/241.

[3] Ward Beullens. Improved cryptanalysis of uov and rainbow. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 348–373, Cham, 2021. Springer International Publishing.

[4] Ward Beullens. Mayo: Practical post-quantum signatures from oil-and-vinegar maps. Cryptology ePrint Archive, Paper 2021/1144, 2021. https://eprint.iacr.org/2021/1144. URL: https://eprint.iacr.org/2021/1144.

[5] Ward Beullens. Mayo: Practical post-quantum signatures from oil-and-vinegar maps. URL: https://www.youtube.com/watch?v=mgW-waIhPf0, September 2021.

[6] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 41–69, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[7] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *CoRR*, cs.CR/0010019, 2000. URL: https://arxiv.org/abs/cs/0010019.

[8] Weiwei Cao, Lei Hu, Jintai Ding, and Zhijun Yin. Kipnis-shamir attack on unbalanced oil-vinegar scheme. In Feng Bao and Jian Weng, editors, *Information Security Practice and Experience*, pages 168–180, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015. doi:10.6028/NIST.FIPS.202.

[10] Marc FISCHLIN. *Signatures and Security Notions*, chapter 2, pages 47–62. John Wiley & Sons, Ltd, 2022. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781394188369.ch2, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781394188369.ch2, doi:10.1002/9781394188369.ch2.

[11] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988. doi:10.1137/0217017.

[12] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[13] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 206–222, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[14] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil and vinegar signature scheme. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 257–266, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[15] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. *CoRR*, abs/1804.00200, 2018. URL: http://arxiv.org/abs/1804.00200, arXiv:1804.00200.

[16] Michele Mosca and Marco Piani. Quantum threat timeline report 2023. *Global Risk Institute*, Dec 2023.

[17] National Institute of Standards and Technology. Call for additional digital signature schemes for the post-quantum cryptography standardization process, sep 2022. Available at https://csrc.nist.gov/CSRC/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf.

[18] National Institute of Standards and Technology. Round 1 additional signatures - post-quantum cryptography: Digital signature schemes: Csrc, Jun 2023. URL: https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[19] J. PATARIN. The oil and vinegar signature scheme. *Presented at the Dagstuhl Workshop on Cryptography, September 1997. transparencies*, 1997. URL: https://cir.nii.ac.jp/crid/1571417126306462848.

[20] 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 Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 475–490, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[21] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi:10.1109/SFCS.1994.365700.

[22] Beullens Ward, Campos Fabio, Celi Sofía, Hess Basil, and J. Kannwischer Matthias. Mayo specification document, 2023. URL: https://pqmayo.org/assets/specs/mayo.pdf.

[23] Thom Wiggers. Post-quantum signatures zoo, Sep 2023. URL: https://pqshield.github.io/nist-sigs-zoo/.

Appendix A

Appendix A: Full EUF-CMA Games

```
Game_1(𝒜)

  𝒮(M, R)
    $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
    $(\mathsf{seed_{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i \in m}, \{\mathbf{L}_i\}_{i \in m}) \leftarrow \text{decodeEsk } \mathsf{sk}$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}(M)$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed'_{sk}})$
    $\mathbf{t} \leftarrow \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt})$

    $\mathsf{ctr} \leftarrow 0, x \leftarrow \text{None}$
    while $\mathsf{ctr} \leq 255 \land x \neq \text{None}$ do
        $(\mathbf{v}, \mathbf{r}) \leftarrow \mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed_{sk}}, \mathsf{ctr})$
        $(\mathbf{A}, \mathbf{y}) \leftarrow \text{BuildLinearSystem}(\mathbf{v}, \mathbf{t}, \{\mathbf{L}_i\}_{i \in m}, \{\mathbf{P}_i^{(1)}\}_{i \in m})$
        $\mathbf{x} \leftarrow \mathsf{SampleSolution}(\mathbf{A}, \mathbf{y}, \mathbf{r})$ //Try to solve the system.
        $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$

    $\mathsf{sig} \leftarrow$ if $\mathbf{x} = \text{None}$ then None else $\text{CalculateS}(\mathbf{x}, \mathbf{v}, \mathbf{O}) \parallel \mathsf{salt}$
    return sig

  $\mathcal{H}'(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed})$
    if $\mathsf{seed} = \mathsf{seed_{sk}} \lor \mathsf{seed} = \mathsf{seed'_{sk}}$ then
        $\mathsf{bad_1} \leftarrow \mathsf{True}$
    return $\mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed})$

  $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
  $\mathsf{seed'_{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
  $Q_s \leftarrow \{\}$
  $\mathsf{bad_1} \leftarrow \mathsf{False}$
  $(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot), \mathcal{G}, \mathcal{H}', \mathcal{I}, \mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
  return $\mathsf{M}^* \notin Q_s \land \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \land \neg\mathsf{bad_1}$
```

Figure A.1: $\mathsf{Game_1}$ played by an adversary $\mathcal{A}$. The highlighted line shows where $\mathsf{seed'_{sk}}$ is used instead of $\mathsf{seed_{sk}}$ for deriving the salt. $\mathcal{H}'$ is provided to the adversary and acts as a wrapper around the random oracle $\mathcal{H}$.

```
$\mathsf{Game}_2(\mathcal{A})$

    $\mathcal{S}(\mathsf{M},\mathsf{R})$

    $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
    $(\mathsf{seed}_{\mathsf{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{L}_i\}_{i\in m}) \leftarrow \mathrm{decodeEsk}(\mathsf{sk})$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}(M)$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$

    $\mathbf{if}$ $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ $\mathbf{then}$
        $\mathbf{if}$ $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle}$ $\mathbf{then}$
            $\mathsf{bad}_2 \leftarrow \mathsf{True}$
        $\mathbf{return}$ $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$
    $\mathbf{t} \xleftarrow{\$} \mathcal{T}$

    $\mathsf{ctr} \leftarrow 0, x \leftarrow \mathrm{None}$
    $\mathbf{while}$ $\mathsf{ctr} \leq 255 \wedge x \neq \mathrm{None}$ $\mathbf{do}$
        $(\mathbf{v}, \mathbf{r}) \leftarrow \mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}_{\mathsf{sk}}, \mathsf{ctr})$
        $(\mathbf{A}, \mathbf{y}) \leftarrow \mathrm{BuildLinearSystem}(\mathbf{v}, \mathbf{t}, \{\mathbf{L}_i\}_{i\in m}, \{\mathbf{P}_i^{(1)}\}_{i\in m})$
        $\mathbf{x} \leftarrow \mathsf{SampleSolution}(\mathbf{A}, \mathbf{y}, \mathbf{r})$ //Try to solve the system.
        $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$

    $\mathsf{sig} \leftarrow \mathbf{if}\ \mathbf{x} = \mathrm{None}\ \mathbf{then}\ \mathrm{None}\ \mathbf{else}\ \mathrm{CalculateS}(\mathbf{x}, \mathbf{v}, \mathbf{O}) \parallel \mathsf{salt}$
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{t} : \mathbf{t}, \mathsf{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
    $\mathbf{return}$ $\mathsf{sig}$

    $\mathcal{I}'(\mathsf{M\_digest}, \mathsf{salt})$

    $\mathbf{if}$ $(\mathsf{M\_digest}, \mathsf{salt}) \notin \mathsf{SigCache}$ $\mathbf{then}$
        $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathsf{t} : \mathcal{I}(\mathsf{M\_digest}, \mathsf{salt}), \mathsf{sig} : \mathsf{None}, \mathsf{fromOrace} : \mathsf{True})$
    $\mathbf{return}$ $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].t$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I}',\mathcal{J}}.\mathsf{Forge}(\mathsf{pk})$
$\mathbf{return}$ $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \ \wedge\ \neg\mathsf{bad}_1 \ \wedge\ \neg\mathsf{bad}_2$
```

Figure A.2: Game_2 played by an adversary $\mathcal{A}$. $\mathsf{SigCache}$ maps $(\mathsf{M\_digest}, \mathsf{salt})$ pairs to the triple $(\mathbf{t}, \mathsf{sig}, \mathsf{fromOracle})$. $\mathcal{H}'$ is the same as in Game_1 but is omitted for brevity.

```
$\mathsf{Game}_3(\mathcal{A})$

    $\mathcal{S}(\mathsf{M},\mathsf{R})$

    $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
    $(\mathsf{seed_{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{L}_i\}_{i\in m}) \leftarrow \mathrm{decodeEsk}\ \mathsf{sk}$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}(M)$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed'_{sk}})$

    **if** $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ **then**
        **if** $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle}$ **then**
            $\mathsf{bad}_2 \leftarrow \mathrm{True}$
        **return** $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$

    $\mathsf{ctr} \leftarrow 0, \mathbf{v} \leftarrow \mathrm{None}$
    **while** $\mathsf{ctr} \leq 255 \wedge \mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank **do**
        $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
        $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$

    $\mathbf{o} \xleftarrow{\$} O^k$
    $\mathbf{t} \leftarrow \mathcal{P}^\star(\mathbf{v} + \mathbf{o})$
    $\mathsf{fails} \xleftarrow{\$} \mathsf{random}$

    $\mathsf{sig} \leftarrow$ **if** $\mathsf{fails}$ **then** None **else** $(\mathbf{v} + \mathbf{o}) \parallel \mathsf{salt}$
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
    **return** $\mathsf{sig}$

    $\mathcal{J}'(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$

    **if** $\mathsf{seed} = \mathsf{seed_{sk}}$ **then**
        $\mathsf{bad}_3 \leftarrow \mathrm{True}$
    **return** $\mathcal{J}(\mathsf{M\_digest}, \mathsf{salt}, \mathsf{seed}, \mathsf{ctr})$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed'_{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot),\mathcal{G},\mathcal{H}',\mathcal{I}',\mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
**return** $\mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G},\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge \neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2 \wedge \neg\mathsf{bad}_3$
```

Figure A.3: Game_3 played by an adversary $\mathcal{A}$. $\mathbf{v}$ and $\mathbf{o}$ now sampled uniformly at random. $\mathcal{H}'$ and $\mathcal{I}'$ are the same as in Game_2 but is omitted for brevity.

```
$\mathsf{Game}_4(\mathcal{A})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$

    $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
    $(\mathsf{seed}_{\mathsf{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in m}, \{\mathbf{L}_i\}_{i\in m}) \leftarrow \text{decodeEsk } \mathsf{sk}$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}'(M)$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$

    **if** $(\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache}$ **then**
        **if** $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle}$ **then**
            $\mathsf{bad}_2 \leftarrow \mathsf{True}$
        **return** $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$

    $\mathsf{ctr} \leftarrow 0, \mathbf{v} \leftarrow \text{None}$
    **while** $\mathsf{ctr} \leq 255 \land \mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank **do**
        $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
        $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$

    $\mathbf{o} \xleftarrow{\$} O^k$
    $\mathbf{t} \leftarrow \mathcal{P}^\star(\mathbf{v} + \mathbf{o})$

    $\mathsf{sig} \leftarrow$ **if** $\mathcal{P}^\star(\mathbf{v} + \cdot)$ is not full rank **then** None **else** $(\mathbf{v} + \mathbf{o}) \,\|\, \mathsf{salt}$
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathrm{t} : \mathbf{t}, \mathrm{sig} : \mathsf{sig}, \mathrm{fromOracle} : \mathsf{False})$
    **return** $\mathsf{sig}$

    $\mathcal{G}'(\mathsf{M})$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}(\mathsf{M})$
    **if** $\mathsf{M\_digest} \in Q_h \land Q_h[\mathsf{M\_digest}] \neq \mathsf{M}$ **then**
        $\mathsf{bad}_4 \leftarrow \mathsf{True}$
    $Q_h[\mathsf{M\_digest}] \leftarrow \mathsf{M}$
    **return** $\mathsf{M\_digest}$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$; $Q_h \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$; $\mathsf{bad}_4 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot), \mathcal{G}', \mathcal{H}', \mathcal{I}', \mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
**return** $\mathsf{M}^* \notin Q_s \land \mathsf{MAYO}_{\mathcal{G}', \mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \land \neg\mathsf{bad}_1 \land \neg\mathsf{bad}_2 \land \neg\mathsf{bad}_3 \land \neg\mathsf{bad}_4$
```

Figure A.4: Game_4 played by an adversary $\mathcal{A}$. The game is lost if there is a hash collision on $\mathcal{G}'$. $\mathcal{H}'$, $\mathcal{I}'$, and $\mathcal{J}'$ are the same as in Game_3 and are omitted for brevity.

```
$\mathsf{Game}_6(\mathcal{A})$

    $\mathcal{S}(\mathsf{M}, \mathsf{R})$

    $Q_s \leftarrow Q_s \cup \{\mathsf{M}\}$
    $(\mathsf{seed}_{\mathsf{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i \in m}, \{\mathbf{L}_i\}_{i \in m}) \leftarrow \mathrm{decodeEsk}\ \mathsf{sk}$

    $\mathsf{M\_digest} \leftarrow \mathcal{G}'(M)$
    $\mathsf{salt} \leftarrow \mathcal{H}(\mathsf{M\_digest}, \mathsf{R}, \mathsf{seed}'_{\mathsf{sk}})$

    $\textbf{if } (\mathsf{M\_digest}, \mathsf{salt}) \in \mathsf{SigCache} \textbf{ then}$
        $\textbf{if } \mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{fromOracle} \textbf{ then}$
            $\mathsf{bad}_2 \leftarrow \mathsf{True}$
        $\textbf{return } \mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})].\mathsf{sig}$

    $\mathbf{v} \xleftarrow{\$} \mathbb{F}_q^{k(n-o)}$
    $\textbf{if } \mathcal{P}^\star(\mathbf{v} + \cdot) \text{ is not full rank } \textbf{then}$
        $\mathsf{bad}_5 \leftarrow \mathsf{True}$
    $\mathbf{o} \xleftarrow{\$} O^k$
    $\mathbf{t} \leftarrow \mathcal{P}^\star(\mathbf{v} + \mathbf{o})$

    $\mathsf{sig} \leftarrow (\mathbf{v} + \mathbf{o}) \parallel \mathsf{salt}$
    $\mathsf{SigCache}[(\mathsf{M\_digest}, \mathsf{salt})] \leftarrow (\mathbf{t}, \mathsf{sig}, \mathsf{False})$
    $\textbf{return } \mathsf{sig}$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{MAYO.Gen}()$
$\mathsf{seed}'_{\mathsf{sk}} \xleftarrow{\$} \mathcal{B}^{8\mathsf{sk\_seed\_bytes}}$
$Q_s \leftarrow \{\}$; $\mathsf{SigCache} \leftarrow \{\}$; $Q_h \leftarrow \{\}$
$\mathsf{bad}_1 \leftarrow \mathsf{False}$; $\mathsf{bad}_2 \leftarrow \mathsf{False}$; $\mathsf{bad}_3 \leftarrow \mathsf{False}$; $\mathsf{bad}_4 \leftarrow \mathsf{False}$; $\mathsf{bad}_5 \leftarrow \mathsf{False}$
$(\mathsf{M}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot,\cdot), \mathcal{G}', \mathcal{H}', \mathcal{I}', \mathcal{J}'}.\mathsf{Forge}(\mathsf{pk})$
$\textbf{return } \mathsf{M}^* \notin Q_s \wedge \mathsf{MAYO}_{\mathcal{G}',\mathcal{I}}.\mathsf{Verf}(\mathsf{pk}, \mathsf{M}^*, \mathsf{sig}^*) \wedge$
                $\neg\mathsf{bad}_1 \wedge \neg\mathsf{bad}_2 \wedge \neg\mathsf{bad}_3 \wedge \neg\mathsf{bad}_4 \wedge \neg\mathsf{bad}_5$
```

Figure A.5: Game_6 played by an adversary $\mathcal{A}$.