POLITECNICO DI TORINO

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Matematica

TESI DI LAUREA

Zero-Knowledge Multivariate Public-Key Cryptography

Supervisor

Prof. Carlo SANNA

Candidate

Alice COLOMBATTO

Summary

Origin and objectives of the thesis

A quantum computer is a computer that exploits quantum mechanical phenomena: for this reason, it is able to efficiently perform tasks that a classical computer would never complete in human time, including solving problems on which cryptographic systems are currently based. Post-quantum cryptography deals with studying new quantum-resistant problem on which to rely in order to be able to run secure signature protocols. In this context, we can observe that the digital signatures can be based on different problems, more or less complex: the following thesis arise from the interest in analysing one of these, the multivariate quadratic problem, in all its facets, from its definition to its use in zero-knowledge protocols, which in turn can fall into different types.

In particular, I focused on the most recent signatures belonging to the state of the art, the one presented by Ming-Shing Chen, called MQDSS and based on an identification scheme, and the one presented by Thibauld Feneuil, based on the zero-knowledge protocol MPC-in-the-head; moreover I have made implementations of both signatures, both in order to be able to demonstrate the theoretical treatments and to sample results regarding their performance.
At the end, in order not to burden the structure of the thesis, I have included some supplementary material, which may be necessary for the understanding of some of the concepts discussed, dealing with the world of quantum and post-quantum, but also more algebraic and cryptographic concepts.

Structure of the thesis

Chapter 1 provides a brief introduction to quantum computing and its formalism. In Chapter 2 are given the main definitions of the different protocols and are explained the main instruments, such as the Fiat–Shamir transform and the MPC-in-the-head protocol. Chapter 3 provides an exhaustive treatment of the quadratic

multivariate problem, starting with the identification schemes from which, by applying the Fiat–Shamir transform, is obtained the digital signature scheme. Chapter 4 discusses existing signatures and considers the possible attacks (and their costs, both computational and in memory) on the quadratic multivariate problem. Finally, the appendix A shows the additional material to integrate some concepts explained in the thesis, while appendix B shows the code that implements what has been studied.

Acknowledgements

"Therefore keep watch,
because you do not know the day or the hour."

Matthew, ch. 25, 1-13

The conclusion of the path of these years, the culmination of which is expressed in this thesis work, marks an important turning point in my life, thus closing a long and important training period and opening one where it will be required to put into practice what I have learned, and not only culturally speaking.

First of all I would like to thank my parents, to whom this thesis is dedicated, those who most of all have had to be patient in this sometimes too long time, who have had to find the strength even when I no longer had any, who they have always followed regardless and who have been close to me, in all the meanings that this term can carry with it.
I thank my brother, who in these difficult years had a sister who was sometimes too absent and too absorbed in her problems to quickly notice those of others.

From the Polytechnic environment I would like to thank my supervisor, Professor Carlo Sanna, for his great patience and availability towards me, without whom this thesis would not exist. It is also my duty to mention Professors Danilo Bazzanella and Nadir Murru, who were the first to introduce me to and fascinate me in the world of cryptography years ago. Andrea Gangemi, P.h.D., first my teacher and then my friend, was also a very important figure for me, with whom I shared both more beautiful nd carefree moments than less, but nonetheless all very important; he has really given me a lot of support, without him my wavering might have led me not to the point that I am able to reach now.

I thank Luca, a fundamental and ever-present support, who with his closeness gave me strength to get to the end this last period, making it infinitely better through smiles.

Moreover I want to thank all my Friends, from those who are my age (or a little less) to those much older than me, both those I met at the Polytechnic and the historical ones, from those I met in my first year up to the more recent ones, including people with whom I bonded a lot in the Conservatory environment: each of them had (and has) an important role, they are an integral part of my life and I love them very much. I hope our bonds are stronger than the distance imposed by the paths of life we will take.

"Vegliate dunque,
perché non sapete né il giorno né l'ora."

Dal Vangelo secondo Matteo, cap. 25, 1-13

La conclusione del percorso di questi anni, il cui apice è espresso in questo lavoro di tesi, segna un importante punto di svolta della mia vita, chiudendo così un periodo formativo lungo ed importante ed aprendone uno dove sarà richiesto di mettere in pratica quanto appreso, e non solo culturalmente parlando.

Ci tengo a ringraziare in primis i miei genitori, ai quali questa tesi è dedicata, coloro che più di tutti hanno dovuto portare pazienza in questo tempo a volte troppo lungo, che hanno dovuto trovare la forza anche quando io non ne avevo più, che mi hanno seguito sempre a prescindere e che mi sono stati vicini, in tutte le accezioni che questo termine può portare con se'.
Ringrazio mio fratello, che in questi anni di difficoltà ha avuto una sorella a volte troppo assente e troppo presa dai suoi problemi per accorgersi celermente di quelli degli altri.

Dell'ambiente del Politecnico tengo a ringraziare il mio relatore, il Professore Carlo Sanna, per la molta pazienza e la disponibilità nei miei confronti, senza il quale questa tesi non esisterebbe. E' doveroso per me citare anche i Professori

Danilo Bazzanella e Nadir Murru, che per primi mi hanno fatto avvicinare ed appasionare al mondo della crittografia anni fa. E' stata inoltre una figura molto importante per me il Dottor Andrea Gangemi, prima mio docente e poi mio amico, col quale ho condiviso sia momenti più belli e spensierati che meno, ma comunque tutti molto importanti; mi ha dato davvero molto supporto, senza di lui i miei vacillamenti avrebbero potuto condurmi non al punto che sono riuscita a raggiungere ora.

Ringrazio Luca, supporto fondamentale e sempre presente, che con la sua vicinanza mi ha dato forza di arrivare in fondo a questo ultimo periodo, rendendolo infinitamente migliore grazie ai sorrisi.

Voglio altresì ringraziare tutti i miei Amici, da quelli che hanno la mia età (o poco meno) a quelli molto più grandi di me, sia quelli conosciuti al Politecnico che quelli storici, da quelli incontrati al primo anno fino ai più recenti, comprese le persone con le quali ho legato molto nell'ambiente del Conservatorio: ciascuno di loro ha avuto (ed ha) un ruolo importante, sono parte integrante della mia vita e voglio loro molto bene. Spero che i nostri legami siano più forti della distanza imposta dalle strade della vita che intraprenderemo.

Table of Contents

List of Tables

List of Figures

Acronyms

PQ
Post-Quantum

QFT
Quantum Fourier Transform

DLP
Discrete Logarithm Problem

MQ
Multivariate Quadratic

MPC
Multi-Party Computation

NP
Nondeterministic Polynomial

ZKPoK
Zero-Knowledge Proof of Knowledge

PPT
Probabilistic Polynomial-Time

DSS
Digital Signature Scheme

EU-CMA
Existential Unforgeability under adaptive Chosen Message Attacks

IDS

Identification Scheme

MPCitH

Multiparty Computation in the Head

MPKC

Multivariate public key cryptosystem

HVZK

Honest Verifier Zero-Knowledge

Chapter 1

Introduction to post-quantum cryptography

1.1 General definitions and main concepts

Governments all around the world are investing heavily in building quantum computers. Society must be prepared for the consequences, including cryptanalytic attacks accelerated by these computers. In particular, Shor's algorithm shatters the foundations for implemented public key cryptography: RSA and the discrete logarithm problem in finite fields and elliptic curves. Long-term confidential documents such as patient health-care records and state secrets need to be secure for many years, but today encrypted information uses RSA or elliptic curves, which one day will no longer do. Post-quantum cryptography is the science that deals with formulating, studying and testing new cryptographic algorithms, implemented on classical computers, that can be secure against cryptanalytic attacks done by a quantum computer.
Let us take a closer look, however, at what quantum mechanics really is, basis for understanding the following treatises.

1.1.1 The postulates of quantum mechanics

Quantum mechanics is a physical theory whose postulates are designed to provide a bridge between the real world and mathematical formalism.

- Postulate I
  Associated with every isolated physical system there is Hilbert space, that is a complex vector space endowed with a inner product and complete with respect to the distance function induced by its inner product: this space is called *the space of states of the system.* The system is completely described by its state

vector - a unitary vector within the space of states and usually denoted by $|\psi\rangle$ using Dirac notation[1] - which contains all the information available.

- Postulate II
  The time evolution of a closed system is deterministic, and is described by a unitary transformation. More precisely, the state of the system at time t_1 is linked to the state of the same at time t_2 through a unitary operator $\mathcal{U}$ dependent solely on the times t_1 and t_2,

$$|\psi\rangle(t_2) = \mathcal{U}(t_1, t_2)|\psi\rangle(t_1)$$

  .

- Postulate III
  Any observable attribute of a physical system can be described by a Hermitian $\mathcal{M}$ acting on the state vector of the system

$$\mathcal{M} : |\psi\rangle \longrightarrow |\psi'\rangle = \mathcal{M}|\psi\rangle.$$

  For each operator $\mathcal{M}$ there are particular states $|\psi\rangle$ such that

$$\mathcal{M}|\psi_m\rangle = m|\psi_m\rangle.$$

  Such states are called eigenstates of the operator $\mathcal{M}$ while the multiplicative constants m are called eigenvalues (relative to the $|\psi\rangle$ eigenstates) of the operator.
  Note that in the finite-dimensional case, this postulate follows from the spectral theorem in C for Hermitian matrices, which asserts that an endomorphism is Hermitian if and only every Hermitian matrix is diagonalizable by a unitary matrix.
  The only possible outcomes of an observable $\mathcal{M}$ are its eigenvalues. In particular, since these are of Hermitian operators, these eigenvalues turn out to be real and the eigenstates form an orthonormal basis of the space of states.
  Quantum measurements are described by a collection M_m of *measurement operators*, where the index m refers to the possible outcomes that may occur during the experiment.
  If the state of the quantum system is $|\psi\rangle$ immediately before the measurement, for what has been seen, it is possible to pose

$$|\psi\rangle = \sum_m c_m|\psi_m\rangle, \qquad c_m \in \mathbb{C}, \tag{1.1}$$

[1] See the additional material in A

and the probability that m occurs as a result is given by

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle = |c_m|^2, \quad \sum_m |c_m|^2 = 1. \tag{1.2}$$

The status after measurement results as

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}, \tag{1.3}$$

where $M^\dagger$ is the conjugate transposed and there is the corrective term in the denominator inserted by normalization.
The latter result most likely makes this postulate the least intuitive among the four: for it is being asserted that if m has been obtained as the result of a measurement, the state of the system *collapses into the eigenstate* of the operator relative to that eigenvalue.

- Postulate IV
  The space of states of a composite physical system turns out to be the tensor product of the spaces of the states of its individual components

  $$\mathcal{H} = \otimes_{a=1}^{n} \mathcal{H}_a.$$

  It follows then that, if system number i is prepared in state $|\psi_i\rangle$, the overall state of the system turns out to be the tensor product of the states of the individual subsystems

  $$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle.$$

  However, it would be wrong to conclude that every state in $\mathcal{H}$ turns out to be defined in this way. Having fixed a basis for each subsystem, we have in fact that the generic state of the composite is writable as a linear combination of the tensor products of these.

1.1.2 Quantum computing

It is now common knowledge that the basis of computation and information classically there is the *bit*. This "unit of information" - assuming only two values, 0 and 1 - can be encoded, for example, as an on/off switch, or as an on/off voltage inside a transistor (as is the case in our laptops). When we enter the realm of atoms and photons, however, we find that these do not assume only configurations comparable to classical on/off, but also super-positions coherent of these. The fundamental unit of quantum information on which stands this new view is called

the *quantum bit*, or *qubit* for short. States analogous to on/off[2] are now denoted by $|0\rangle$ and $|1\rangle$ while the generic coherent state in which the qubit can exist, called *superposition*, is conventionally denoted by

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.4}$$

where a and b are complex numbers.
Unfortunately, quantum mechanics states that it is impossible to know exactly the $|\psi\rangle$ state with a single measurement, or equivalently, to know α and β. When the state of a qubit is measured, it is only possible to obtain $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$.
Clearly, $|\alpha|^2 + |\beta|^2 = 1$, since these are complementary probabilities.
This surprising mismatch between the state of the qubit and the result of the measurement lies at the foundation of quantum computation and information, thus making it counter intuitive and abstract.

Because of what has just been observed, the state of a qubit can be understood as a vector unit in the 2-dimensional space of states. It is therefore possible to rewrite the Equation 1.4 as

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle\right), \tag{1.5}$$

where θ, ϕ and γ are real numbers.
Since the $e^{i\gamma}$ phase has no observable effect [3] it can be removed, thus rewriting the definition as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle, \tag{1.6}$$

where the values of θ and ϕ uniquely define a point on the three-dimensional unit sphere, usually called the *Bloch sphere*, which is a useful tool for understanding some properties of the qubit (unfortunately, there is no generalization for multiple qubits).
Not only that, such a representation also turns out to have interesting practical applications, such as the study of the polarization of a photon: the Bloch sphere in fact provides a geometric representation of the possible polarization configurations of the particle (vertical/horizontal = $|0\rangle/|1\rangle$, resulting in super-positions).

---

[2]These two states are commonly called computational basis states and form a basis orthonormal of the vector space

[3]Recalling Postulate III, the probability that the result m occurs is given by $\langle\psi|M_m^\dagger M_m|\psi\rangle$ and $\langle\psi|e^{-i\gamma}M_m^\dagger M_m e^{i\gamma}|\psi\rangle = \langle\psi|M_m^\dagger M_m|\psi\rangle$.

Of all the coherent super-positions, some turn out to be of greater significance, acquiring particular designations:

$$|\pm\rangle = \frac{|0\rangle \pm |1\rangle}{\sqrt{2}} \qquad and \qquad |\pm_i\rangle = \frac{|0\rangle \pm i|1\rangle}{\sqrt{2}} \tag{1.7}$$

represent, for example, the rotations of $|0\rangle$ and $|1\rangle$ on the x-axis and the y. It is necessary to dwell on one detail: to claim that one can find the qubit in an infinite number of possible states is incorrect: the measurement will provide only $|0\rangle$ or $|1\rangle$ and especially it changes its state, collapsing it from its superposition.

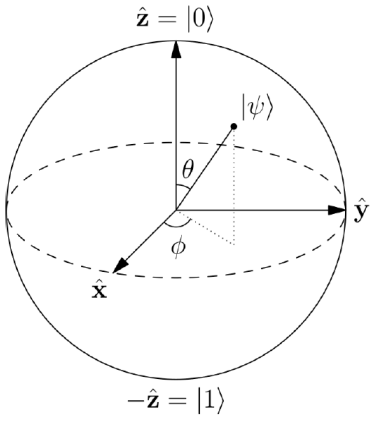


Figure 1.1: Representation of the state of a qubit on the Bloch sphere

1.2 Elements of Quantum algorithms

After explaining what quantum mechanics is, I want to give a hint of the most important problems that quantum computing is therefore able to solve. So let's take for example the *integer factorization*: it is in fact impossible for current computers to factor large numbers that are the product of two prime numbers of almost equal size; the quantum computer with $2n$ q-bits can factor instead numbers with lengths of n bits. Another example is *quantum database search*, for which the search for information in an unordered database takes orders of magnitude of time much smaller than a classical computer. The use in the fields of *simulations of quantum mechanics* is also reported: for example in the context of chemistry, biology, medicine and many others, the quantum computer - unlike the classic one - can calculate millions of variables simultaneously. Finally it's mandatory mentioning the field of *cryptography*, destroyed by the potential arrival of the quantum computer, which is capable of deciphering codes.

1.2.1 Grover's algorithm

An extended class of problems can be categorized as search i.e. *find the elements x for which the statement $f(x)$ is true.*
An unstructured search problem is one in which nothing is known (or no assumptions are used) about the structure of the solution space and the statement f; a structured search problem is one in which one can find information about the search and statement f exploited. Thus for a search in a space of dimension N the search requires an evaluation of elements of order $O(N)$; Grover's algorithm performs a search on an unordered set of $N = 2^n$ elements to find the unique element that satisfies a given condition: it possesses quadratic velocity, using only $O(\sqrt{N})$ operations.

Grover's search algorithm demonstrates how the properties of quantum systems can be used to improve the timing of execution times of classical algorithms. Specifically, the algorithm of Grover exploits the (quantum-only) possibility of superposition of quantum states and phase shifting of the amplitude of the quantum states.

1.2.2 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is the key computational ingredient in many algorithms quantum; in particular, we observe that the QFT:

- performs the transform of quantum amplitudes;
- allow the estimation of the phase, by approximation of the eigenvalues of a unit operator under certain conditions;
- solves the problem of the *hidden subgroup*, a generalization of phase estimation;
- solves the *discrete logarithm problem*, which has no solution in the classical framework.

1.2.3 Quantum Phase Estimation algorithm

The quantum phase estimation algorithm is a quantum algorithm to estimate the phase corresponding to an eigenvalue of a given unitary operator. This algorithm finds one of its main applications for solving the problem of order-finding and factoring. These two problems and their solution are two central routines of Shor's algorithm, that is interesting because it proves that the quantum computers are inherently more powerful than classical computers, and overall are capable of breaking the public-key cryptosystem of RSA.

Chapter 2

Preliminaries and background

2.1 Definitions

In this chapter, there will be given the main instruments and definitions that will be essential to understand the core of this thesis, the *MQ problem*, addressed in the next chapter.

Definition 1 (NP-relation) *An NP-relation is a relation* $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ *such that*

$$(x, w) \in \mathcal{R} \quad \iff \quad R(x, w) = 1 \quad \textit{and} \quad |w| \leq p(|x|)$$

where $R(\cdot, \cdot)$ *is a polynomial-time algorithm and* $p(\cdot)$ *is a polynomial.*

Note that $\{0,1\}^*$ is the space of finite strings in the alphabet of 0 and 1 and $|w|$ is the length of these strings.
In this context, x is called *public key* (often denoted by pk) or problem or instance and w is called *secret key* (often denoted by sk) or solution or witness.
In other words, NP-relations are relations $\mathcal{R}$ for which is computationally easy to determine if, given the solution (x, w), this one belongs (or not) to the relation $\mathcal{R}$, but given x it is usually difficult to find w such that (x, w) is solution (and so $(x, w) \in \mathcal{R}$).

Definition 2 (Problem) *Let* $\mathcal{R}$ *be an NP-relation: then*

given x*, the problem of determining if there exists* w *such that* $(x, w) \in \mathcal{R}$ *is called* decision problem*;*

given x, the problem of finding one w such that $(x, w) \in \mathcal{R}$ is called search problem.

Note that w might not exist.

Definition 3 (Protocol) *A cryptographic protocol is defined as a series of steps and message exchanges between multiple entities in order to achieve a specific security objective.*

The properties of a protocol, generally, are the following:

- Everyone involved in the protocol must know the protocol and all of the steps to follow in advance;
- Everyone involved in the protocol must agree to follow it;
- The protocol must be unambiguous, that is every step is well defined and there is no chance of misunderstanding;
- The protocol must be complete, i.e., there is a specified action for every possible situation.

Note that the protocol may or may not be *interactive*, i.e. it may be required that there is an active exchange of information between the parties or not; in the event of non-interactivity, it is possible to be enabled to prove one's identity or the truthfulness of the information in possession without sharing it.

Definition 4 (PPT) *A Probabilistic Polynomial-Time Problem (said PPT problem) is a problem solvable by a probabilistic Turing machine in polynomial time.*

2.1.1 ZKPoK

A Zero-Knowledge Proof of Knowledge (in short *ZKPoK*) is a method by which one party, the **prover**, can prove to another party, the **verifier**, that s/he (the prover) has knowledge of some secret information, in a way that does not reveal such information to the verifier. In cryptography, the prover has a public key and a secret key; by construction the public key is an instance of a difficult mathematical problem, while the private key is the solution of that instance. The ZKPoK has many applications in cryptography, such as in the identification protocols - the prover identifies himself to the verifier by giving a ZKPoK of his private key - or in the digital signatures - where the prover signs a document by turning it into a ZKPoK of his private key.

For a given NP-relation $\mathcal{R}$ and a publicly known $x \in \{0,1\}^*$, ZKPsoK are formalized as a prover convincing a verifier that s/he knows a w such that $(x, w) \in \mathcal{R}$ (that is, a solution to the search problem).

Zero-Knowledge protocols find use in numerous areas of verification, but in systems designed for ascertainment, it is also necessary to ensure the proof property of knowledge; in particular, three different levels of security are distinguished:

- **Authentication schemes**: the prover can prove to the verifier that he is the prover itself, and no one else can prove to the verifier that he is the real prover.

- **Identification schemes**: The prover can prove to the verifier to be the prover, and the verifier cannot prove to someone else to be the prover.

- **Signature schemes**: the prover can prove to the verifier that he is the prover, and the verifier cannot prove to himself that he is the prover either.

ZKPoK in Graph Theory

One of the most immediate examples that can be offered to understand how ZKPoK works is its application to Graph Theory, so we report the recall of some notions necessary for understanding it.

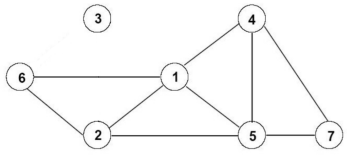


Figure 2.1: A graph.

A graph is a pair (V, E), where V is a finite set and E is a set of unordered pairs of elements of V; the elements of V are called *vertices*, while the elements of E are called *edges*.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs: then a graph *isomorphism* between G_1 and G_2 is a bijective function $f : V_1 \to V_2$ such that

$$\{v, w\} \in E_1 \iff \{f(v), f(w)\} \in E_2.$$

So G_1 and G_2 are isomorphic if there exists a graph isomorphism between them (in such case we write $G_1 \cong G_2$ or $f : G_1 \to G_2$).
Notice that if the functions $f : G_1 \to G_2$ and $g : G_2 \to G_3$ are isomorphism, then the functions $f^{-1} : G_2 \to G_1$ and $(g \circ f) : G_1 \to G_3$ are isomorphism too. It is also important to point out that the permutation, being a bijective function with domain and codomain of equal size, is also an isomorphism.

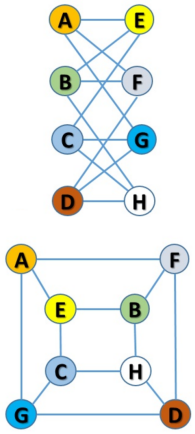
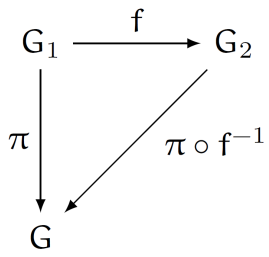


Figure 2.2: Two isomorphic graphs.

The problem of determining if two given graphs are isomorphic is known as **Graph Isomorphism Problem**; given two graphs, both with n vertices, the obvious way to check if they are isomorphic is to try all $n!$ possible f, which is infeasible for large n (the complexity of the most efficient known algorithm[1] grows as $\mathcal{O}(2^{(log\ n)^c})$ where $c = 3$).

In this context, the ZKPoK for Graph Isomorphism requires that there are two publicly known graphs, G_1 and G_2; the prover knows a graph isomorphism $f : G_1 \to G_2$ and wants to convince the verifier of that, but without revealing f. The protocol is the following:

[1]It's a quasi-polynomial-time algorithm that works for all graphs and it's due to László Babai (November 2005; his result was later improved by Helfgott, who proved the value of the constant c.)



1) The prover generates a random permutation $\pi : V_1 \to V_1$ and constructs a graph $G \cong G_1$ by applying π to the vertices of G_1. Then he sends G to the verifier. 2) The verifier chooses a random challenge $ch \in \{1, 2\}$ and sends it to the prover. 3) If the prover received $ch = 1$, then he sends π to the verifier. If instead the prover received $ch = 2$, then he sends $g := \pi \circ f^{-1}$ to the verifier. 4) If $ch = 1$, then the verifier checks that π is a graph isomorphism between G_1 and G. If $ch = 2$, then the verifier checks that g is a graph isomorphism between G_2 and G.

Table 2.1: ZKPoK protocol for Graph Isomorphism

Observations. For $ch = 1$ the verifier obviously accepts because, by construction, π is a graph isomorphism between G_1 and G; for $ch = 2$ the verifier accepts by proving that $g := \pi \circ f^{-1}$ is a graph isomorphism between G_2 and G. In this way, the hypothetical cheater (who does not possess f) cannot know what to give as an answer either in case the challenge is worth 1 or in case it is worth 2. It is also intuitive that no information on f is revealed: in fact, in case of $ch = 1$ the verifier receives π that do not contains any information about f; in case instead of $ch = 2$, the verifier receives g (that is given by $\pi \circ f^{-1}$) but the randomness of the permutation destroy any information on f.
There is a chance, however, that a cheater will convince the verifier that s/he possesses f, constructing G as the prover and receiving as a challenge precisely 1: this happens once in two times; therefore, by repeating the protocol n times, the probability of this happening is lowered to $(\frac{1}{2})^n$.

2.1.2 Sigma Protocols

Let P_1, P_2, V_1 and V_2 be probabilistic-polynomial-time algorithms: a *Sigma Protocol* for an NP-relation $\mathcal{R}$ is an interactive protocol between a prover $P = (P_1, P_2)$ and a verifier $V = (V_1, V_2)$, which follows the steps below.

1. The prover sends to the verifier a *commitment*.
2. The verifier sends to the prover a *challenge* from a finite set of challenges $\mathcal{C}$.
3. The prover computes an appropriate *response* and sends it to the verifier.
4. The verifier checks if the response is correct, in according to the challenge and the commitment. In such a case the verifier accepts, otherwise he rejects.

Table 2.2: Sigma Protocol

Definition 5 (Transcript) *The quadruple that contains the instance x, the commitment* comm*, the challenge* ch *and the response* rsp *is called transcript and it indicates as transcript(x, com, ch, rsp).*

So in other words, given a transcript(x, com, ch, rsp), the execution of its protocol is as reported below:

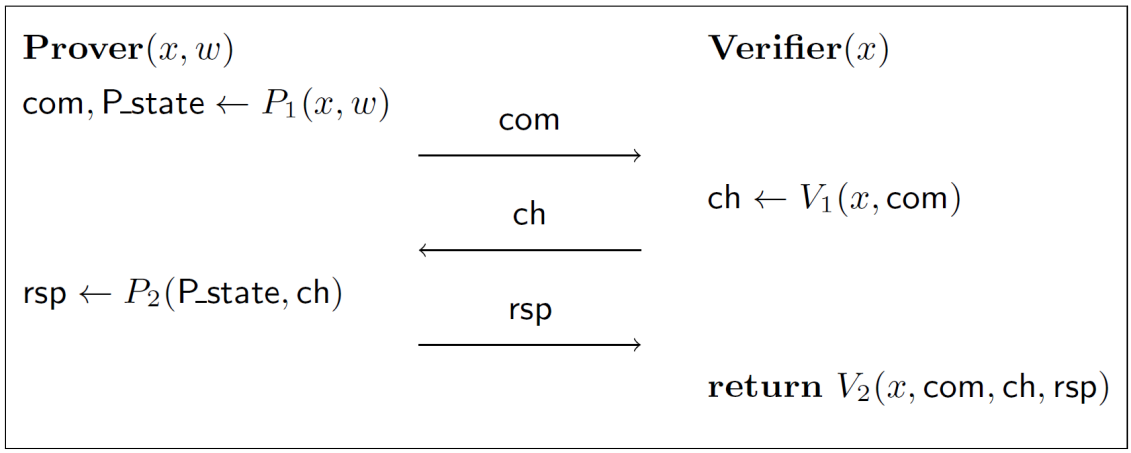


Figure 2.3: Summary of Sigma Protocol

Generally, a cryptographic protocol must respect three properties (that hold for the Sigma Protocol), that are **Completeness**, **Soundness** and **Honest Verifier Zero-Knowledge (HVZK)**.

Definition 6 (Completeness) *If all parties follow the protocol on input $(x, w) \in \mathcal{R}$, then the verifier always accept.*

In other words, the protocol should work if every party is honest.

Definition 7 (Soundness) *There exists a polynomial-time algorithm, called extractor, that takes as input two transcripts,* $(x, \mathsf{com}, \mathsf{ch}_1, \mathsf{rsp}_1)$ *and* $(x, \mathsf{com}, \mathsf{ch}_1, \mathsf{rsp}_1)$*, with* $\mathsf{ch}_1 \neq \mathsf{ch}_2$*, of a honest execution of the protocol on input* $(x, w) \in \mathcal{R}$*, and returns as output a witness* w' *such that* $(x, w') \in \mathcal{R}$*.*

Here w and w' may or may not be equal. In other words, cheating is as difficult as computing the witness.

Definition 8 (Honest Verifier Zero-Knowledge) *There exists a PPT algorithm, called simulator, that takes as input* x *and returns as output a random transcript* $(x, \mathsf{com}, \mathsf{ch}, \mathsf{rsp})$ *having the same probability distribution of transcripts of a honest execution of the protocol on input* $(x, w) \in \mathcal{R}$*.*

In other words, the transcripts of a legit execution of the protocol contain no information on the witness.

Obviously it is necessary also provide security measures, in order to analyze how safe it is to use one protocol compared to another, so for this the following definition is given:

Definition 9 (Soundness error) *The soundness error of a sigma protocol is the probability that a verifier accepts the response of a cheater who does not know the witness.*

In the case of a sigma protocol, if this has a soundness error equal to p, then by doing n rounds (that is, repeating the protocol procedure n times), the probability became equal to p^n. It should be observed that $0 < p < 1$ (in fact we cannot exclude a priori that a cheater is able to give the correct information even only thanks to luck), therefore the more rounds you make, the more the probability of being able to cheat will tend to zero, making the protocol increasingly secure.

The ZKPoK for Graph Isomorphism shown before does indeed satisfy the properties of a sigma protocol.

2.1.3 Schnorr Identification Protocol

The Schnorr Identification Protocol, or Schnorr Identification Scheme, is an identification protocol built from a ZKPoK of the discrete logarithm problem over a cyclic group having as order a prime number.
To fully understand this protocol, it becomes necessary the following definition:

Definition 10 (Identification Protocols based on ZKPoK) *An Identification Protocol is a NP-relation* $\mathcal{R}$ *in which each user has a public key* x *and a private key* w *such that* $(x, w) \in \mathcal{R}$ *and, when a user with public key* x *needs to*

identify himself to another user, he gives a ZKPoK of his key w (being the prover, while the other user is the verifier).

Note that in order to achieve a security level λ, the number of rounds should be at least $\frac{\lambda}{\log \frac{1}{p}}$, where p is the soundness error of the sigma protocol; in this way has the probability of the cardsharper to cheat that is less than $2^{-\lambda}$.

It is also necessary to remember the notions of linear algebra:

Definition 11 (Discrete Logarithm Problem (DLP)) *Let G be a finite multiplicative group generated by $g \in G$. Given $h \in G$, the problem of finding an integer x such that $h = g^x$ is known as the Discrete Logarithm Problem.*

It has been studied that the DLP is a difficult problem when the group G is $\mathbb{Z}_n^*$ or a multiplicative group of a finite field or a cyclic subgroup of the group of an elliptic curve over a finite field: in this case there is no sub-exponential algorithm able to solve it. Note that the choice of group is very important since there are groups for which the DLP is simple to solve, such as $\mathbb{Z}_p$ with p a prime number or, more in general, an additive field $\mathbb{Z}_n$ with n positive integer.

Also in the Schnorr Identification Protocol the group is carefully chosen: in fact the group that usually is used is $G = \langle g \rangle$, where g is given by $g = h^{\frac{p-1}{q}}$ and $h \in \mathbb{Z}_p^*$ such that $h^{\frac{p-1}{q}} \not\equiv 1 \pmod{p}$ (p and q are two primes such that q divides $p-1$); in this context, given G multiplicative cyclic group of q elements and g its generator, that are publicly known, the prover wants to convince the verifier that he knows x (his private key, that is an integer in $[0, q)$), but without revealing x.

1. The prover generates a random $r \in [0, q)$, computes $k = g^r$, and sends k to the verifier.
2. The verifier generates a random challenge $c \in [0, q)$ and sends it to the prover.
3. The prover computes $y = r + cx \bmod q$ and sends it to the verifier.
4. The verifier accepts if $g^y = kh^c$, where h is the public key of the prover, obtained by $h = g^x$. Otherwise, the verifier rejects.

Table 2.3: Schnorr Identification Protocol

Note that The Schnorr Identification Protocol satisfies the properties of a sigma

protocol (the proof is omitted). Furthermore the soundness error of Schnorr Identification Protocol is equal to $\frac{1}{q}$.

2.1.4 ZKPoK for other problems

There are other problems to which the zero-knowledge proof can be applied, such as *Square Root* and *Discrete Logarithm*; for both of them, the obtained protocols satisfies the three properties seen that characterize the sigma protocol.
In the context of the Square Root Problem, there is G (usually $G = \mathbb{Z}_n^*$ with n product of two primes) as finite multiplicative commutative group and $a \in G$: the prover wants to convince the verifier that he knows $x \in G$ such that $a = x^2$, but without revealing x.
In the context of the Discrete Logarithm Problem, instead, there is G multiplicative cyclic group of n elements (G of arbitrary order) and g generator of G, with $h \in G$: the prover wants to convince the verifier that he knows an integer x such that $h = g^x$, but without revealing x.

The protocols are reported here:

1. The prover generates a random $r \in G$, computes $b = r^2$, and sends b to the verifier.
2. The verifier generates a random challenge $\mathsf{ch} \in \{0, 1\}$ and sends it to the prover.
3. The prover computes $c = x^{\mathsf{ch}} r$ and sends it to the verifier.
4. The verifier accepts if $c^2 = a^{\mathsf{ch}} b$. Otherwise, he rejects.

Table 2.4: Square Root Protocol

1. The prover generates a random $r \in [0, n)$, computes $k = g^r$, and sends k to the verifier.
2. The verifier generates a random challenge $c \in \{0, 1\}$ and sends it to the prover.
3. The prover computes $t = cx + r \bmod n$ and sends it to the verifier.
4. The verifier accepts if $h^c k = g^t$. Otherwise, he rejects.

Table 2.5: Discrete Logarithm Protocol

The protocol called *Discrete Logarithm Protocol* is better known and popular as *Fiat–Shamir's identification protocol*. Note that the soundness error of the Discrete Logarithm Protocol is $\frac{1}{2}$, while Schnorr's is $\frac{1}{q}$, which can be much smaller.

2.1.5 The Cut-and-Choose Protocol and the Identification Scheme

The protocols that we will see in the next chapters employ the cut-and-choose approach, so it's given the following:

Definition 12 (Cut-and-Choose Protocol) *A cut-and-choose protocol is a two-party protocol in which one party tries to convince another party that some data he sent to the former was honestly constructed according to an agreed upon method.*

Note that the expression *cut-and-choose* is used in analogy to a popular cake sharing problem: given a complete cake to be shared among two parties distrusting each other. A fair way for them to share the cake is to have one of them cut the cake in two equal shares, and let the other one choose his favourite share. This solution guarantees that it is in the former's best interest to cut the shares as evenly as possible.
In the context of the next protocols, a prover first divides her secret into shares and then proves the correctness of some shares depending on the choice of a verifier without revealing the secret itself.

This paragraph provides the definitions necessary to understand the protocols that will follow; furthermore, in order to standardize the notation (and therefore be consistent with Sakumoto's publication), concepts already introduced in the

previous chapter are rewritten in a concise manner.

Let A and B be two finite sets and let $R \subset A \times B$ be a binary relation; let S be a finite set and let $x \in_R S$ be a random element of this set: then $R(x) := \{s : (x, s) \in R\}$. If $s \in R(x)$ then s is **solution** for the problem x.

Let Setup be an algorithm which takes a security parameter 1^λ and outputs the parameter $param$ (therefore the security level is arbitrary, depending on the λ you pass it); let Gen be a key-generation algorithm which takes $param$, and outputs the couple public key - secret key (pk, sk); let V be a verifier and P be a prover: the quantities $param$ and pk are publicly known between both parties, while sk belongs just to P.

Definition 13 *An* ***Identification Scheme*** *is a tuple of algorithms* $(Setup, Gen, P, V)$*; the protocol* (P, V) *is called* ***Identification Protocol****.*

The **Commitment Scheme** is a scheme that works in two phase:

- The sender computes a commitment value $c \leftarrow \mathsf{Com}(s; \rho)$, where Com is the string commitment function, and sends c to the receiver, where s is a string and ρ is a random string.

- The sender gives (s, ρ) to the receiver and the receiver verifies that $c = \mathsf{Com}(s; \rho)$.

Note that it's required that the string commitment function Com be both statistically hiding and computationally binding.

Definition 14 (Computational Hiding) *For an adversary* $\mathcal{A}$ *we define the advantage for the commitment hiding game for a pair of messages* m, m' *as*

$$Adv^{Hiding}_{\mathsf{Com}}(\mathcal{A}, m, m') =$$

$$\left| \mathop{\mathbb{P}}_{bits \leftarrow \{0,1\}^\lambda}[1 = \mathcal{A}(\mathsf{Com}(bits, m)] - \mathop{\mathbb{P}}_{bits \leftarrow \{0,1\}^\lambda}[1 = \mathcal{A}(\mathsf{Com}(bits, m')] \right|$$

We say that Com is computationally hiding if for all polynomial-time algorithms $\mathcal{A}$, and every pair of messages (m, m') the advantage $Adv^{Hiding}_{\mathsf{Com}}(\mathcal{A}, m, m')$ is a negligible function of the security parameter λ.
Note that in the Commitment Scheme means that at the end of the first phase,

no receiver can distinguish two commitment values generated from two distinct strings even if the receiver is computationally unbounded[2].

Definition 15 (Computational Binding) *For an adversary $\mathcal{A}$ we define its advantage for the commitment binding game as*

$$Adv_{\mathsf{Com}}^{Binding}(\mathcal{A}) = \mathbb{P}[\mathsf{Com}(bits, m) = \mathsf{Com}(bits', m') | (bits, m, bits', m') \leftarrow \mathcal{A}(1^{\lambda})]$$

We say that Com is computationally binding if for all polynomial-time algorithms $\mathcal{A}$, the advantage $Adv_{\mathsf{Com}}^{Binding}(\mathcal{A})$ is a negligible function of the security parameter λ.
Note that in the Commitment Scheme means that no polynomial-time sender can change the committed string after the first phase; please note that in practice the commitment scheme is constructed from a collision-resistant hash function.

2.2 Non-interactive ZKPoK

Following the first publication regarding knowledge-zero, three years later was released the mathematical demonstration of the existence of non-interactive zero-knowledge proofs. In fact, the main defect of normal Zero-Knowledge Protocols is that they are only able to function if the verifier is online and willing to interact with the prover by choosing a random challenge; what is discussed in that papers[3] is the possibility of even disposing of the cyclic interaction between the parties, of the verifier's random choice of challenge, and allow verification of the prover's secret with a single message: all of this is possible if a short random Common Reference String (CRS) is exchanged a priori to be used as key. This translates in mathematical language with the concept of a one-way function, and in cryptography with the concept of a hash function.
In order to allow the key to be exchanged, however, an initial phase is required of communication, called *trusted setup*,in which a *trusted party* generates it publicly. An alternative to the CRS model is to use a *Random Oracle Model*[4].

[2]Unbounded means that there is no a-priori fixed limitation to the amount of space and time that a valid, non-diverging program could take. Unbounded space and time is a hard requirement for a computational model to be Turing-complete, in the sense that there exist functions that take arbitrarily high time and space to be computed; any computational model with a space or time limitation would therefore be unable to compute such functions making it, by definition, not Turing-complete.

[3]See [1] and [2]

[4]A random oracle is a mathematical function that associates with each possible question a truly random answer chosen within its output domain.

2.2.1 Fiat–Shamir transform

The Fiat–Shamir transform is a technique for converting a sigma protocol identification scheme into a digital signature scheme[5].

This heuristic collapse the number of rounds required into a single round, increasing the space used for the challenge from {0,1} into a larger space that allows to control the soundness error (e.g. $\mathbb{Z}_q$), at the cost of making the Honest-Verifier protocol Zero-Knowledge. Furthermore there is no more the verifier that generate challenges, but this is compute using an hash function.

Since this technique completely replaces the interactivity part of the protocol (considering that it reduces the number of rounds to one and the verifier does not even have to answer and to generate the challenge), it can also be seen as converting an interactive ZKPoK protocol into a non-interactive one (Zero-Knowledge is not necessarily required).

The Schnorr's identification scheme is the best example to observe how the Fiat–Shamir transform is applied.

Schnorr Signature

The Schnorr signature scheme is a non-interactive Zero-Knowledge protocol that is also a Proof of Knowledge and a digital signature concept, obtained by applying Fiat–Shamir heuristics to Schnorr's identification protocol seen above. In fact, by including a message M as an optional value, one can obtain a signature on M, which can only be produced by someone who knows the secret key s.
Let $H : \{0,1\} \to \mathbb{Z}_q$ be an hash function, public among the parties: then the protocol revisited for proving the knowledge of the secret s concerning the key public PK_A is:

[5] See [3]

1. The prover generates a random integer $r \in [1, q]$, and s/he hides it by calculating $u = g^r \bmod p$ (commitment).
2. The prover uses $H(\cdot)$ to calculate the challenge $e = H(u||M)$, where u is represented as a bit string, M is the message, and $||$ represents the concatenation operation (challenge).
3. The prover calculates $z = -se + r \bmod q$ and communicates to the verifier the pair (z, e) that constitutes the message signature (response).
4. The verifier only needs to make verifications because of $H(\cdot)$ in common, so he calculates his $u_B = PK_A^e \cdot g^z$ and his $e_B = H(u_B||M)$, and checks whether $e_B = e$. This is valid because if the prover is honest then s/he knows s and can compute z, so we have $u_B = g^{se} \cdot g^{-se+r} \bmod p$ and $u = g^r \bmod p$ (completeness).

Table 2.6: Schnorr's Digital Signature Protocol

Observations. Even in this implementation of the protocol, however, if the same r is used in two separate signatures, it is possible to trace the private key. It is in fact sufficient to subtract the two values of z, such as $z_2 - z_1 = (r_2 - r_1) - s(e_2 - e_1)$. Therefore, if $e_2 \neq e_1$, we have: $s = \frac{z_1 - z_2 + r_2 - r_1}{e_2 - e_1}$. In terms of security, however, it has been shown that Schnorr's signature scheme can be considered as such if the hash function H is defined and modeled as a random oracle.
It has been demonstrated [5] that

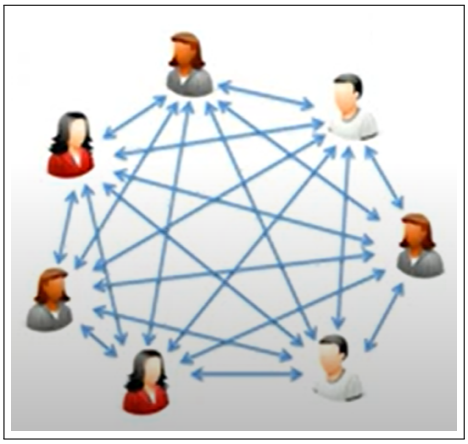
Theorem 1 *Given an identification scheme* Π*, let* Π' *be its signature scheme obtained by applying the Fiat–Shamir transform to* Π*, if* Π *is secure and* H *is modeled as a random oracle, then* Π' *is safe.*

Theorem 2 *If the discrete logarithm problem is difficult relative to the defined problem* (G, q, g)*, then Schnorr's identification protocol can be called safe.*

2.3 The MPC-in-the-Head Paradigm

We've seen that zero-knowledge proofs allow a prover to prove to a verifier about the veracity of a statement without revealing anything beyond the assertion; the so called *MPC protocol*, a secure multiparty computational protocol, allows a set of n mutually distrusting parties to compute a joint function of their private inputs. The security guarantee is that any algorithm that corrupts even all but one of the n parties cannot learn anything about the input of the uncorrupted parties beyond what it can learn from the function output.

Definition 16 (MPC) *A Multi-Party Computation is a distributed protocol between n parties, where every party P_i holds a secret w_i and the output of the protocol is $f(w_1, \ldots, w_n)$; in this setting, the values of w_i remain secret.*



In particular, an MPC protocol is defined by some instructions for each party P_i; furthermore it's defined as $View_i$ of P_i the triplet $(w_i, r_i, (m_1, \ldots, m_j))$ where w_i is the private input (of the user i), r_i is the randomness used by the user i and $(m_1, \ldots, m_j)$ are the messages received by other parties. Two views $View_i$, $View_j$ are consistent if the parties follow the instructions and the set of messages sent by P_i is consistent with the set of messages received by P_j and vice versa.

The function f isn't a generic one but has the particular shape:

$$f(w_1, \ldots, w_n) = \begin{cases} 1 & if \ \ (x, \sum_1^n w_i) \in \mathcal{R} \\ 0 & otherwise \end{cases} \tag{2.1}$$

Note that the quantities x, f and $\mathcal{R}$ are public and the only one that is secret is $w = \sum_{i=1}^n w_i$.
MPC protocols enjoy both *passive security*, that means that the protocol is secure

and correct with respect to honest-but-curious parties, and *t-privacy*, that is that the views of any t parties leak no information about the secret w.

After these definitions, it's possible to explain the **MPC in the head protocol**, that presents as setting the relation $\mathcal{R}$, that is chosen and public, then it's formed $(x, w) \in \mathcal{R}$ and f such that

$$f_x(w) = 1 \iff (x, w) \in \mathcal{R};$$

in this context the MPC in the head protocol consists of the following steps:

1. The prover generates shares of w (i.e. $w = \sum w_i$) and run *in its head* the MPC protocol among n parties, obtaining output shares $f_x(w)_i$ for each party; then commits $View_i$ to C_i for each i and sends them to the verifier; he also sends $f_x(w)_1, \ldots, f_x(w)_n$.
2. The verifier picks t indexes among $\{1, \ldots, n\}$ and sends them to the prover.
3. The prover sends the views of the received indexes.
4. The verifier checks the commitments and that the received views are consistent: if $\sum_i f_x(w)_i = 1$, accepts.

Table 2.7: MPC-in-the-head Protocol

Note that the MPC protocol has the three properties of the sigma protocol, that are completeness (implied by the correctness of the MPC protocol), soundness (given by the fact that if the prover cheats, then some of the views will be inconsistent) and Honest Verifier Zero-Knowledge (given by the t-privacy of the MPC protocol). The soundness error of this protocol is equal to $\frac{1}{\binom{n}{t}}$. This protocol is a perfectly correct MPC that satisfies t-privacy (in fact t = 2 is sufficient); it's possible even extend the protocol to have negligible soundness error without sequential repetition, in this case the steps for this protocol are the following:

1. The prover chooses random $w_1, \ldots, w_n$ subject to the condition that $\oplus_{i \in [n]} w_i = w$ where w is the witness.
2. The prover runs the MPC protocol Π *in his head* (by choosing uniform random coins for each party) to generate the views of each party P_i. Let $View_i$ denote the view of party P_i in the execution of Π.
3. The prover generates commitment to each $View_i$ separately using a statistically binding commitment scheme and sends the commitment to the verifier.
4. The verifier chooses t random $i_1, \ldots, i_t \in [n]$ and sends it to the prover.
5. The prover opens the commitment to the views $View_{i_1}, \ldots, View_{i_t}$.
6. The verifier checks if the openings are correct and if it is the case it checks if the views are consistent. If they are in consistent it outputs 0 if the protocol outputs 0; otherwise, it outputs 1.

Table 2.8: Extended Protocol

Note finally that the security of MPC comes in different flavors depending on *how* the parties are corrupted: if the corruption algorithm follows the protocol description but may try to learn arbitrary information from the protocol transcripts, then it's referred as semi-honest; on the other hand, if the corruption algorithm can deviate arbitrarily from the protocol description, then it's referred as malicious.

Chapter 3

The Multivariate Quadratic Problem

3.1 Introduction to the MQ problem

The problem of solving a system of multivariate quadratic polynomials over a finite field, which is called an *MQ problem*, is a interesting problem in cryptography. The associated decision problem is known to be NP-complete and a random instance of the MQ problem is widely believed to be intractable (there is no known polynomial-time quantum algorithm to solve the MQ problem).
Many studies have been done over time on this type of problem both in the field of symmetric and asymmetric cryptography. A very important milestone, which brings results on which subsequent studies of MQ signature are then based, is the paper by Sakumoto et al. [6], in which is proposed a public-key identification schemes based on the conjectured intractability of the MQ problem under the assumption of the existence of a non-interactive commitment scheme which is statistically-hiding and computationally-binding.

Let $\mathbb{F}_q$ be a finite field (of order q): then we denote by $\mathcal{MQ}(n, m, \mathbb{F}_q)$ a family of functions

$$\left\{ \mathbf{F}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})) \middle| f_l(\mathbf{x}) = \sum_{i,j} a_{l,i,j} x_i x_j + \sum_i b_{l,i} x_i, \right. \\ \left. a_{l,i,j}, b_{l,i} \in \mathbb{F}_q \;\; for \;\; l = 1, \ldots, m \right\} \tag{3.1}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$.

Definition 17 (The MQ function) *We call* $\mathbf{F} \in \mathcal{MQ}(n, m, \mathbb{F}_q)$ *an MQ function.*

In order to use the cut-and-choose protocol, it is necessary to be able to split the secret into several parts: this is possible by exploiting the property of group homomorphism, i.e. using the modular exponentiation or a linear function[1]; however the MQ function does not seem to have such a property. For this reason, new splitting techniques are introduced, which exploit the bilinearity of the following *polar form*:

Definition 18 (Polar form of an MQ function) *The polar form* $\mathbf{G}$ *of the MQ function* $\mathbf{F}$ *is a function* $\mathbf{G}(\mathbf{x_1}, \mathbf{x_2}) = \mathbf{F}(\mathbf{x_1} + \mathbf{x_2}) - \mathbf{F}(\mathbf{x_1}) - \mathbf{F}(\mathbf{x_2})$.

It's known that the function $\mathbf{G} = (g_1, \ldots, g_m)$ has the property of bilinearity, in fact $g_l(\mathbf{x}, \mathbf{y}) = \sum_{i,j} a_{l,i,j}(y_i x_j + x_i y_j)$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$.

The division technique used involves the following steps: let $\mathbf{s}$ and $\mathbf{v} = \mathbf{F}(\mathbf{s})$ be respectively a secret key and a public key:

- First, the secret key is divided as $\mathbf{s} = \mathbf{r}_0 + \mathbf{r}_1$;
- Then the public key $\mathbf{v} = \mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1)$ is represented as

  $$\mathbf{v} = \mathbf{F}(\mathbf{r}_0) + \mathbf{F}(\mathbf{r}_1) + \mathbf{G}(\mathbf{r}_0, \mathbf{r}_1)$$

  (thanks to the polar form $\mathbf{G}$ of the MQ function $\mathbf{F}$);
- Since there is still the term $\mathbf{G}(\mathbf{r}_0, \mathbf{r}_1)$ that depends on both $\mathbf{r}_0$ and $\mathbf{r}_1$, one should consider that $\mathbf{r}_0$ and $\mathbf{F}(\mathbf{r}_0)$ are further divided as $\mathbf{r}_0 = \mathbf{t}_0 + \mathbf{t}_1$ and $\mathbf{F}(\mathbf{r}_0) = \mathbf{e}_0 + \mathbf{e}_1$, respectively.
- In this case, the public key can be divided into two parts:

  $$\mathbf{v} = (\mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0) + (\mathbf{F}(\mathbf{r}_1) + \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_1)$$

  thanks to the bilinearity of $\mathbf{G}$.
  Each of the two parts is represented by either $(\mathbf{r}_1, \mathbf{t}_0, \mathbf{e}_0)$ or $(\mathbf{r}_1, \mathbf{t}_1, \mathbf{e}_1)$ (no information on $\mathbf{s}$ can be obtained from one of the two tuples).

An intractability assumption for a random instance is defined as follows:

Definition 19 (Intractability of an MQ function) *For polynomially bounded functions* $n = n(\lambda)$, $m = m(\lambda)$ *and* $q = q(\lambda)$, *it is said that* $\mathcal{MQ}(n, m, \mathbb{F}_q)$ *is intractable if there is no polynomial-time algorithm that takes* $(\mathbf{F}, \mathbf{v})$ *generated via* $\mathbf{F} \in_R \mathcal{MQ}(n, m, \mathbb{F}_q)$, $s \in_R \mathbb{F}_q^n$, *and* $\mathbf{v} \leftarrow \mathbf{F}(\mathbf{s})$ *and finds a preimage*[2] $\mathbf{s}' \in \mathbb{F}_q^n$ *such that* $\mathbf{F}(\mathbf{s}') = \mathbf{v}$ *with non-negligible probability* $\epsilon(\lambda)$.

[1] See the additional material in A

[2] *See additional material in A*

Note that, in general, a function f is said *polynomially bounded* if $|f(x)| \leq p(x)$ for some real polynomial p.

Definition 20 (The MQ problem) *For* $\mathbf{F} \in \mathcal{MQ}(n, m, \mathbb{F}_q)$*, we define a binary relation* $R_{\mathbf{F}} = \{(\mathbf{v}, \mathbf{x}) \in \mathbb{F}_q^m \times \mathbb{F}_q^n : \mathbf{v} = \mathbf{F}(\mathbf{x})\}$*. Given an instance* $\mathbf{F} \in \mathcal{MQ}(n, m, \mathbb{F}_q)$ *and a vector* $\mathbf{v} \in \mathbb{F}_q^m$*, the* $\mathcal{MQ}$ *problem is finding a solution* $\mathbf{s} \in R_{\mathbf{F}}(\mathbf{v})$*.*

3.2 3-pass identification scheme of Sakumoto [6]

The two identification schemes, both the 3-step and the 5-pass one, are made up of the sequential composition and the parallel composition of the identification protocols.
In this part, assuming the existence of the non-interactive commitment scheme Com described as before (so statistically hiding and computationally binding), it's given the identification scheme, consisting of a 3-pass statistical zero-knowledge argument of knowledge for $R_{\mathbf{F}}$:

Prover's input: $((\mathbf{F}, \mathbf{v}), \mathbf{s})$		Verifier's input: $(\mathbf{F}, \mathbf{v})$
1. Pick $\mathbf{r}_0$, $\mathbf{t}_0 \in_R \mathbb{F}_q^n$, $\mathbf{e}_0 \in_R \mathbb{F}_q^m$ 2. $\mathbf{r}_1 \leftarrow \mathbf{s} - \mathbf{r}_0$, $\mathbf{t}_1 \leftarrow \mathbf{r}_0 - \mathbf{t}_0$ 3. $\mathbf{e}_1 \leftarrow \mathbf{F}(\mathbf{r}_0) - \mathbf{e}_0$ 4. $c_0 \leftarrow \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0)$ 5. $c_1 \leftarrow \mathsf{Com}(\mathbf{t}_0, \mathbf{e}_0)$ 6. $c_2 \leftarrow \mathsf{Com}(\mathbf{t}_1, \mathbf{e}_1)$	$\xrightarrow{(c_0, c_1, c_2)}$	
	$\xleftarrow{\mathsf{Ch}}$	Pick $\mathsf{Ch}_R \in \{0, 1, 2\}$
If $\mathsf{Ch} = 0$, $\mathsf{Rsp} \leftarrow (\mathbf{r}_0, \mathbf{t}_1, \mathbf{e}_1)$ If $\mathsf{Ch} = 1$, $\mathsf{Rsp} \leftarrow (\mathbf{r}_1, \mathbf{t}_1, \mathbf{e}_1)$ If $\mathsf{Ch} = 2$, $\mathsf{Rsp} \leftarrow (\mathbf{r}_1, \mathbf{t}_0, \mathbf{e}_0)$	$\xrightarrow{\mathsf{Rsp}}$	• If $\mathsf{Ch} = 0$, parse $\mathsf{Rsp} = (\mathbf{r}_0, \mathbf{t}_1, \mathbf{e}_1)$ and check $c_1 \stackrel{?}{=} \mathsf{Com}(\mathbf{r}_0 - \mathbf{t}_1, \mathbf{F}(\mathbf{r}_0) - \mathbf{e}_1)$ $c_2 \stackrel{?}{=} \mathsf{Com}(\mathbf{t}_1, \mathbf{e}_1)$ • If $\mathsf{Ch} = 1$, parse $\mathsf{Rsp} = (\mathbf{r}_1, \mathbf{t}_1, \mathbf{e}_1)$ and check $c_0 \stackrel{?}{=}$ $\mathsf{Com}(\mathbf{r}_1, \mathbf{v} - \mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \mathbf{e}_1)$ $c_2 \stackrel{?}{=} \mathsf{Com}(\mathbf{t}_1, \mathbf{e}_1)$ • If $\mathsf{Ch} = 2$, parse $\mathsf{Rsp} = (\mathbf{r}_1, \mathbf{t}_0, \mathbf{e}_0)$ and check $c_0 \stackrel{?}{=} \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0)$ $c_1 \stackrel{?}{=} \mathsf{Com}(\mathbf{t}_0, \mathbf{e}_0)$

Table 3.1: 3-pass identification protocol

It has got a soundness error[3] of $\frac{2}{3}$.
The protocol shown in the table 3.1 is written succinctly; in more detail, the process to be carried out is as follows:

- The beginning part, that is the *key generation*, is done with the previously described algorithm, Setup and Gen, given in order to formulate the definition 13; note that in order for $\mathbf{F} \in \mathcal{MQ}$ to be intractable one has passed the arbitrary parameter λ to have $n = n(\lambda)$, $m = m(\lambda)$ and $q = q(\lambda)$ polynomially bounded.

[3]The soundness error is the difference between the known (estimated) output of the system and the actual achieved output

After choosing a random vector $\mathbf{s} \in_R \mathbb{F}_n^q$, Gen computes $\mathbf{v} \leftarrow \mathbf{F}(\mathbf{s})$, then outputs $(pk, sk) = (\mathbf{v}, \mathbf{s})$.

- In the very first step described in 3.1, the prover choose randomly the three values $\mathbf{r}_0$, $\mathbf{t}_0$ and $\mathbf{e}_0$;
- Then the prover split the secret (aka the secret key): the first time, in point 2, calculating $s := r_0 + r_1$ (so having r_0 and s, he calculate r_1), and the second time in point 3, calculating $r_0 := t_0 + t_1$, in such a way to not have $G(r_0, r_1)$ that depends both on r_0 and r_1.
- In points 4, 5 and 6 the prover calculates the three commitments[4] that it sends to the verifier: depending on the challenge that the verifier will send to the prover, the prover will in turn send a specific tuple that will allow two of the three Commitments to be checked for correctness. This is done in this way because the three Commitments depend differently on $\mathbf{t}_0$, $\mathbf{t}_1$, $\mathbf{e}_0$, $\mathbf{e}_1$, and $\mathbf{r}_1$: thus, by revealing only the tuples constructed in such a way that no further quantities can be computed, it is in no way possible to trace any information regarding the private key $\mathbf{s}$.
- Upon receiving the three Commitments, (c_0, c_1, c_2), the verifier creates a random challenge, drawing a value between 0,1 and 2; he then communicates the value of the challenge to the prover.
- Depending on what it has received, the prover sends *only one* of the following three tuples to the verifier:

$$(\mathbf{r}_0, \mathbf{t}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{t}_1, \mathbf{e}_1), (\mathbf{r}_1, \mathbf{t}_0, \mathbf{e}_0)$$

- Knowing the two relation $\mathbf{s} - \mathbf{r}_0 = \mathbf{r}_1$ and $\mathbf{r}_0 - \mathbf{t}_0 = \mathbf{t}_1$, the verifier has to check the correctness of the equations

$$G(t_0, r_1) + e_0 = v - F(r_1) - G(t_1, r_1) - e_1$$

in fact the only more elaborate verification is that of c_0:

$c_0 = \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0) =^5 \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{r}_0 - \mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^6$

$\mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{r}_0, \mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^7$

[4]Note that the random string ρ in Com is not written explicitly but is implied; moreover note that the "," means concatenation between the different strings.

[5]$\mathbf{t}_0 = \mathbf{r}_0 - \mathbf{t}_1$

[6]Bilinearity of the function polar form $\mathbf{G}$

[7]Definition of $\mathbf{G}$ as the polar form of $\mathbf{F}$: see 18

$$\mathsf{Com}(\mathbf{r}_1, \mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1) - \mathbf{F}(\mathbf{r}_0) - \mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^{8}$$

$$\mathsf{Com}(\mathbf{r}_1, \mathbf{v} - \mathbf{e}_0 - \mathbf{e}_1 - \mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) = \mathsf{Com}(\mathbf{r}_1, \mathbf{v} - \mathbf{e}_1 - \mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1))$$

It is easy to see that the verifier always accepts an interaction with the honest prover: thus the 3-pass scheme has *perfect correctness*. The properties enjoyed by the protocol are as follows:

Theorem 3 *The 3-pass protocol is statistically zero knowledge when the commitment scheme* Com *is statistically hiding.*

Theorem 4 *The 3-pass protocol is argument of knowledge for* $R_{\mathbf{F}}$ *with knowledge error* $\frac{2}{3}$ *when the commitment scheme* Com *is computationally binding.*

Note that a modified version of this protocol was discussed, which involves using a hash function H that is collision-resistant, calculating $c = H(c_0, c_1, c_2)$ instead of sending the three commitment values.[9]

3.3 5-pass identification scheme of Sakumoto [6]

Also in this protocol, we assume the same assumption as in the previous one is valid, namely that there exists the non-interactive commitment scheme Com that is statistically hiding and computationally binding. The identification scheme is here summarized:

[8] $\mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1) = \mathbf{F}(\mathbf{s}) = \mathbf{v}$; moreover $\mathbf{F}(\mathbf{r}_0) = \mathbf{e}_0 + \mathbf{e}_1$, as presented after the definition 18

[9] See [9].

Prover's input: $((\mathbf{F}, \mathbf{v}), \mathbf{s})$		Verifier's input: $(\mathbf{F}, \mathbf{v})$
1. Pick $\mathbf{r}_0$, $\mathbf{t}_0 \in_R \mathbb{F}_q^n$, $\mathbf{e}_0 \in_R \mathbb{F}_q^m$ 2. $\mathbf{r}_1 \leftarrow \mathbf{s} - \mathbf{r}_0$ 3. $c_0 \leftarrow \mathsf{Com}(\mathbf{r}_0, \mathbf{t}_0, \mathbf{e}_0)$ 4. $c_1 \leftarrow \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0)$	$\xrightarrow{(c_0, c_1)}$	
	$\xleftarrow{\alpha}$	Pick $\alpha \in_R \mathbb{F}_q$
$\mathbf{t}_1 \leftarrow \alpha\mathbf{r}_0 - \mathbf{t}_0$ $\mathbf{e}_1 \leftarrow \alpha\mathbf{F}(\mathbf{r}_0) - \mathbf{e}_0$	$\xrightarrow{(\mathbf{t}_1, \mathbf{e}_1)}$	
	$\xleftarrow{\mathsf{Ch}}$	Pick $\mathsf{Ch} \in_R \{0,1\}$
If $\mathsf{Ch} = 0$, $\mathsf{Rsp} \leftarrow \mathbf{r}_0$ If $\mathsf{Ch} = 1$, $\mathsf{Rsp} \leftarrow \mathbf{r}_1$	$\xrightarrow{\mathsf{Rsp}}$	• If $\mathsf{Ch} = 0$, parse $\mathsf{Rsp} = \mathbf{r}_0$ and check $c_0 \stackrel{?}{=} \mathsf{Com}(\mathbf{r}_0, \alpha\mathbf{r}_0 - \mathbf{t}_1, \alpha\mathbf{F}(\mathbf{r}_0) - \mathbf{e}_1)$ • If $\mathsf{Ch} = 1$, parse $\mathsf{Rsp} = \mathbf{r}_1$ and check $c_1 \stackrel{?}{=} \mathsf{Com}(\mathbf{r}_1, \alpha(\mathbf{v} - \mathbf{F}(\mathbf{r}_1)) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \mathbf{e}_1)$

Table 3.2: 5-pass identification protocol

It has got a soundness error of $\frac{1}{2} + \frac{1}{2q}$; this error is smaller than the one in the previous protocol when $p \geq 4$. As before, let us analyze the steps described in the table 3.2:

- The Setup algorithm and the *key-generation* algorithm of this scheme are identical to those of the scheme of the previous chapter, the 3-pass one.
- In step 1, the prover choose randomly $\mathbf{r}_0$, $\mathbf{t}_0$ and $\mathbf{e}_0$;
- In point 2, the prover calculate the quantity $\mathbf{r}_1$ following the split of the secret $\mathbf{s} := \mathbf{r}_0 + \mathbf{r}_1$; the second split will be done later, after the interaction with the verifier.
- In steps 3 and 4, the prover calculate the Commitments c_0 and c_1 and send them to the verifier;
- Once received (c_0, c_1), the verifier choose randomly the value α that sends to the prover;
- The prover computes the values $\mathbf{t}_1$ and $\mathbf{e}_1$ through the formulas $\alpha\mathbf{r}_0 = \mathbf{t}_0 + \mathbf{t}_1$ and $\alpha\mathbf{F}(\mathbf{r}_0) = \mathbf{e}_0 + \mathbf{e}_1$. Note that for more than one choice of $\alpha \in \mathbb{F}_q$, an

impersonator cannot response both of verifier's challenges $\mathsf{Ch} = 0$ and $\mathsf{Ch} = 1$ unless the impersonator has a solution $\mathbf{s}$ for $\mathbf{v}$.
Then he send to the verifier the couple $(\mathbf{t}_1, \mathbf{e}_1)$ just calculated.

- At this point the verifier choose the challenge Ch and sends it to the prover;
- According to what he received, the prover sends *only one* of this values, or $\mathbf{r}_0$ or $\mathbf{r}_1$.
- If $\mathsf{Ch} = 0$, the check of the verifier consists only of using the inverse formulas of those used by the prover; instead if $\mathsf{Ch} = 1$, he has to check if $c_1 = \mathsf{Com}(\mathbf{r}_1, \alpha(\mathbf{v} - \mathbf{F}(\mathbf{r}_1)) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \mathbf{e}_1)$: in fact

$$c_1 = \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\mathbf{t}_0, \mathbf{r}_1) + \mathbf{e}_0) =^{10} \mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\alpha\mathbf{r}_0 - \mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^{11}$$

$$\mathsf{Com}(\mathbf{r}_1, \mathbf{G}(\alpha\mathbf{r}_0, \mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^{12}$$

$$\mathsf{Com}(\mathbf{r}_1, \alpha\mathbf{G}(\mathbf{r}_0, \mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^{13}$$

$$\mathsf{Com}(\mathbf{r}_1, \alpha\Big(\mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1) - \mathbf{F}(\mathbf{r}_0) - \mathbf{F}(\mathbf{r}_1)\Big) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =$$

$$\mathsf{Com}(\mathbf{r}_1, \alpha\mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1) - \alpha\mathbf{F}(\mathbf{r}_0) - \alpha\mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =^{14}$$

$$\mathsf{Com}(\mathbf{r}_1, \alpha\mathbf{v} - \mathbf{e}_0 - \mathbf{e}_1 - \alpha\mathbf{F}(\mathbf{r}_1) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) + \mathbf{e}_0) =$$

$$\mathsf{Com}(\mathbf{r}_1, \alpha\Big(\mathbf{v} - \mathbf{F}(\mathbf{r}_1)\Big) - \mathbf{G}(\mathbf{t}_1, \mathbf{r}_1) - \mathbf{e}_1)$$

It's easy to see that the verifier always accepts an interaction with the honest prover: thus the 5-pass scheme has *perfect correctness*. The properties enjoyed by the protocol are as follows:

Theorem 5 *The 5-pass protocol is statistically zero knowledge when the commitment scheme* Com *is statistically hiding.*

---

[10] $\mathbf{t}_0 = \alpha\mathbf{r}_0 - \mathbf{t}_1$

[11] Bilinearity of the function $\mathbf{G}$

[12] Bilinearity of the function $\mathbf{G}$

[13] Definition of $\mathbf{G}$ as the polar form of $\mathbf{F}$.

[14] $\mathbf{F}(\mathbf{r}_0 + \mathbf{r}_1) = \mathbf{F}(\mathbf{s}) = \mathbf{v}$; moreover $\alpha\mathbf{F}(\mathbf{r}_0) = \mathbf{e}_0 + \mathbf{e}_1$

Theorem 6 *The 5-pass protocol is argument of knowledge for $R_{\mathbf{F}}$ with knowledge error $\frac{1}{2} + \frac{1}{2q}$ when the commitment scheme* Com *is computationally binding.*

Security and Efficiency

These identification schemes, both the 3-pass and the 5-pass one, are secure against impersonation under *active* attack and *passive* attack, respectively; note that requiring security against impersonation under active attack is stronger than under passive attack.
Below are two tables comparing various parameters that identify the efficiency of the identification schemes - 3 and 5 pass (such as bits occupied, memory required, rounds needed and arithmetic operations performed) between Sakumoto's [6] ones and those of other authors, specified below:

	SD [18] [19]	CLE [20]	PP [21]	IDS [6]
round	52	52	73	52
system parameter (bit)	122 500	4 608	28 497	285 600
public key (bit)	350	288	245	80
secret key (bit)	700	192	177	84
communication (bit)	59 800	45 517	100 925	29 640
arithmetic ops. (times/field)	$2^{24}/\ \mathbb{F}_2$	$2^{16}/\ \mathbb{F}_{257}$	$2^{22}/\ \mathbb{F}_{127}$	$2^{26}/\ \mathbb{F}_2$
permutations (times/size)	$2/\ S_{700}$	$2/\ S_{24}$	$2/\ S_{161}, S_{177}$	NO
hash function (times)	4	4	8	4
best known key-recovery attack	2^{87}	2^{84}	$> 2^{74}$	2^{80}

Table 3.3: Comparison of 3-pass schemes on 80-bit security against key-recovery attack when the impersonation probability is less than 2^{-30}

The first column report the parameters of the SD identification scheme, based on the Syndrome Decoding problem, elaborated by Stern in multiple works [18] [19]; in the second column, instead, there is the CLE identification scheme, based on Constrained Linear Equations, always elaborated by Stern [20]. The third column refers to the PP identification scheme, based on the Perceptrons Problem, written by Pointcheval [21].

	SD [18] [19]	SD [23]	PK [22]	CLE [20]	PP [21]	IDS [6]
round	31	31	31	31	52	33
system parameter (bit)	122 500	32 768	4 608	4 608	28 497	259 200
public key (bit)	2450	512	384	288	245	120
secret key (bit)	4900	1024	203	192	177	180
communication (bit)	120 652	61 783	27 234	27 528	105 060	26 565
arithmetic ops. (times/field)	2^{23}/ $\mathbb{F}_2$	2^{18}/ $\mathbb{F}_{256}$	2^{15}/ $\mathbb{F}_{251}$	2^{15}/ $\mathbb{F}_{257}$	2^{21}/ $\mathbb{F}_{127}$	2^{22}/ $\mathbb{F}_{2^4}$
permutations (times/size)	8/ S_{700}	2/ S_{128}	3/ S_{48}	4/ S_{24}	2/ S_{161}, S_{177}	NO
hash function (times)	2	2	2	2	5	2
best known key-recovery attack	2^{87}	2^{87}	2^{85}	2^{84}	$> 2^{74}$	2^{83}

Table 3.4: Comparison of 5-pass schemes on 80-bit security against key-recovery attack when the impersonation probability is less than 2^{-30}

The first, fourth, fifth and sixth columns represent the author identification schemes of the previous table; in the second column there is the IDS based on the Syndrome Decoding problem drawn by Cayrel [23], instead in the third column there is the IDS based on the Permuted Kernel (PK) problem, formulated by Shamir [22].

Please note that all the values of both tables of the system parameter can be reduced to 128 bit if a pseudo-random number generator is used. Finally, note that the field *permutations* shows the number of times of computing permutations and the size of the permutation, where S_n means a permutation over $\{1, \ldots, n\}$.

Chapter 4

MQ-Based Signatures

4.1 Introduction

This chapter discusses the possibility of making signatures based on the MQ problem; therefore, the necessary definitions are given first and then the steps to obtain a signature structure will be analyzed.

Definition 21 (Digital Signature Scheme) *A digital signature scheme, in short DSS, is a triplet of polynomial-time algorithms DSS = (*KeyGen*,* Sign*,* Verify*).*

The algorithms that compose the DSS are the following:

§ KeyGen is the *key generation algorithm* and it's a probabilistic algorithm that on input 1^k, where k is the security parameter you've to pass, outputs a key pair (sk, pk) (that is the secret key and the public key);

§ Sign is the *signing algorithm* and it's a possibly probabilistic algorithm that takes in input the secret key sk and a message M, then outputs the signature σ;

§ Verify is the so called *verification algorithm* and is a deterministic algorithm: takes in input the public key pk, a message M and the signature σ, and outputs a bit b where $b = 1$ indicates that the signature is accepted, while $b = 0$ indicates the rejection.

The standard **security** notion for a digital signature scheme is the property of the *Existential Unforgeability under adaptive Chosen Message Attacks*, in short EU-CMA. To arrive at its definition, the construction is as follows:

□ Let $Exp^{eu-cma}_{DSS(1^k)}(\mathcal{A})$ be the notation that identifies the following experiment:

1. The challenger generates a valid pair of keys (sk, pk) and gives pk to the attacker $\mathcal{A}$;
2. The attacker may now repeatedly ask for signatures on chosen messages $(M_1, \ldots, M_q)$ of its choosing, and receives the valid signatures $(\sigma_1, \ldots, \sigma_q)$ in response;
3. In the end of the experiment, the attacker must output a message and signature M^*, σ^* such that:
   i. the message M^* was not one of the messages requested in the previous step
   ii. the message/signature verifies correctly under the public key.

□ In the experiment $Exp^{eu-cma}_{DSS(1^k)}(\mathcal{A})$, the success probability is denoted as

$$Succ^{eu-cma}_{DSS(1^k)}(\mathcal{A}) := \mathbb{P}\Big[Exp^{eu-cma}_{DSS(1^k)}(\mathcal{A}) = 1\Big]$$

□ Let $k \in \mathbb{N}$ be the arbitrary security parameter and DSS a digital signature scheme defined above: it's call **DSS EU-CMA-secure** if $\forall\ Q_s, t = poly(k)$, the maximum success probability $InSec^{eu-cma}\Big(DSS(1^k); t, Q_s\Big)$ is negligible in k, for all the possible adversaries A (running in a time less or equal than t); in symbols

$$InSec^{eu-cma}(DSS(1^k); t, Q_s) := \max_{\mathcal{A}}\ \ Succ^{eu-cma}_{DSS(1^k)}(\mathcal{A}) = negl(k).$$

So finally a signature is called **EU-CMA-secure** if any PPT adversary has only a success probability that is negligible.

At this point it's possible analyze how to apply the Fiat–Shamir transform to an identification scheme that respects certain characteristics: for this reason it is define the $(2n+1)$-*pass scheme* and the *soundness property*.

Definition 22 ((2n+1)-pass IDS) *Let $k \in \mathbb{N}$: then the identification scheme (KeyGen, P, V) is a $(2n+1)$-pass one if it has got n challenge C_j with $0 < j \leq n$.*

This type of scheme is called *canonical* $(2n+1)$*-pass identification scheme* if the prover can be split into $n+1$ subroutines $\mathcal{P} = (\mathcal{P}_0, \ldots, \mathcal{P}_n)$ and the verifier into $n+1$ subroutines $\mathcal{V} = (\mathsf{ChS}_1, \ldots, \mathsf{ChS}_n, \mathsf{Verify})$ such that:

- $\mathcal{P}_0(sk)$ computes the initial commitment *com* and sent it as first message.
- $\forall\ j \leq n$, ChS_j computes the (j-th) challenge message $\mathsf{ch}_j \leftarrow_R C_j$, sampling a random element from the j-th challenge space.

- $\forall\ 0 < i \leq n$, $\mathcal{P}_i(sk, trans_{2i})$ computes the (i-th) response of the prover thanks to the fact that it owns both the private key and $trans_{2i}$, the transcript so far containing the first $2i$ messages.
- $\mathsf{Verify}(pk, trans)$ outputs $\mathcal{V}$'s final decision because he owns both the public key and the total transcript.

It must be observed that this definition of IDS tells us that this scheme is a *public coin*[1] one, and as consequence the challenges are sampled using the uniform distribution.

Definition 23 (Special n-soundness) *A $(2n+1)$-pass IDS satisfies the special soundness property if there exists a PPT algorithm $\mathcal{E}$ such that given any pair of accepting transcripts (with $ch_1 \neq ch_2$) $\mathcal{E}$ can recover sk.*

In other words, this property says that given two transcripts that agree on all messages but the last challenge and possibly the last response, one can extract a valid secret key. The algorithm $\mathcal{E}$ is called the *extractor*, while the two transcripts are $trans = (\mathsf{com}, \mathsf{ch}_1, \mathsf{resp}_1, \ldots, \mathsf{resp}_{n-1}, \mathsf{ch}_n, \mathsf{resp}_n)$ and $trans' = (\mathsf{com}, \mathsf{ch}_1, \mathsf{resp}_1, \ldots, \mathsf{resp}_{n-1}, \mathsf{ch}'_n, \mathsf{resp}'_n)$ (with $\mathsf{ch}_n \neq \mathsf{ch}'_n$).

At this point we observe that a $(2n+1)$-step scheme remains too generic: for this reason we give the following theorem

Theorem 7 *Every canonical $(2n+1)$-pass IDS that fulfills special n-soundness can be turned into a canonical 3-pass IDS, fulfilling special soundness.*

Given an identification scheme identified as $IDS = (\mathsf{KeyGen}, \mathcal{P}, \mathcal{V})$, considering it a canonical $(2n+1)$-pass IDS that enjoys the special n-soundness, then the 3-pass scheme, identified as $IDS' = (\mathsf{KeyGen}, \mathcal{P}', \mathcal{V}')$, is too canonical and fulfills special soundness if the quantities above are given as:

- $\forall\ j \in (0, n)$ (that is all the j but the last challenge generation algorithm) it moves ChS_j from $\mathcal{V}$ to $\mathcal{P}$: in other words $\mathcal{P}'$ computes $\mathsf{com}' = (\mathsf{com}, \mathsf{ch}_1, \mathsf{resp}_1, \ldots, \mathsf{resp}_{n-1}, \mathsf{ch}_{n-1})$ using $\mathcal{P}_0, \ldots, \mathcal{P}_{n-1}$ and $\mathsf{ChS}_1, \ldots, \mathsf{ChS}_{n-1}$.
- After $\mathcal{P}'$ sent com' to $\mathcal{V}'$;

[1] In a public coin protocol the verifier generates the challenge *ch* and sends it to the prover, while in a private coin context the verifier generates *ch*, applies some transformation to it (i.e. in the Graph Isomorphism Problem it's the random permutation π) and sends it to prover. In the private coin protocol, the prover does not see what has been generated by the verifier, but only the output of a function that used it as input. Note that the Fiat–Shamir transform can only be used in the case of public coins.

- then $\mathcal{V}'$ replies with $\mathsf{ch}'_1 \leftarrow \mathsf{ChS}_n(1^k)$;
- $\mathcal{P}'$ computes $\mathsf{resp}'_1 \leftarrow \mathcal{P}_n(sk, trans_{2n})$ and sends it to $\mathcal{V}'$;
- $\mathcal{V}'$ verifies the transcript using Verify.

Graphically, the procedure can be summarized as follows:

Prover $\mathcal{P}'$		Verifier $\mathcal{V}'$
Pick the needed quantities to calculate the $n-1$ commitments		
Calculate $c_i \leftarrow \mathsf{Com}(\cdot,\cdot)$ with $i \in (0,n)$ and let $\mathsf{com}' := (c_0, c_1, \ldots, c_{n-1})$	$\overset{com'}{\longrightarrow}$	
	$\overset{\mathsf{ch}'_1}{\longleftarrow}$	Pick $\mathsf{ch}'_1 \in \mathsf{ChS}_n(1^k)$
Depending on the value of ch'_1, $\mathsf{Rsp}'_1 \leftarrow \mathcal{P}_n(sk, trans_{2n})$	$\overset{\mathsf{Rsp}'_1}{\longrightarrow}$	Check using Verify

Table 4.1: A 3-pass IDS converted from a $(2n+1)$-pass one

At this point it is natural to think of being able to convert all the schemes with an arbitrary (odd) number of steps into 3-step schemes (which therefore enjoy 1-soundness); however, this is false since the extracting algorithms need more than two transcriptions (they must form two pairs, one to agree on ch_1 but not on ch_2 and vice versa).
For this reason it holds the following

Theorem 8 *The 5-pass identification scheme of the section 3.3 does not fulfill special n-soundness if the computational $\mathcal{MQ}$-problem is hard.*

It is therefore necessary to analyze which are the safety parameters that affect this type of schemes; looking at the literature on $(2n+1)$-pass IDSS, it is observed that most of these are 5-steps, therefore we can restrict the field of analysis to only this type, point out what is lost due to this restriction. It is considered a particular type of 5-pass identification protocols, where the length of the two challenges is restricted to q and 2:

Definition 24 ($q2$ - IDS) *Let $k \in \mathbb{N}$: a $q2$-Identification Scheme $IDS(1^k)$ is a canonical 5-pass identification scheme where for the challenge spaces C_1 and C_2 it holds that $|C_1| = q \in \mathbb{Z}^*$ and $|C_2| = 2$.*

Furthermore, the probability that the commitment *com* takes a given value is negligible (in k), where the probability is taken over the random choice of the input and the used randomness.

Moreover, it's said that a $q2$-Identification scheme $IDS(1^k)$ has a **q2-extractor** if there exists a PPT algorithm $\mathcal{E}$ - the *extractor* - that, given a public key pk and four transcripts, outputs a matching secret key sk for pk with non-negligible success probability (in k).
The transcripts in this case are of the form $trans^{(i)} = (\mathsf{com}, \mathsf{ch}_1^{(i)}, \mathsf{resp}_1^{(i)}, \mathsf{ch}_2^{(i)}, \mathsf{resp}_2^{(i)})$ with $0 \leq i \leq 4$, and the challenges are such that

$$\mathsf{ch}^{(1)}{}_1 = \mathsf{ch}^{(2)}{}_1 \neq \mathsf{ch}^{(3)}{}_1 = \mathsf{ch}^{(4)}{}_1$$
$$\mathsf{ch}^{(1)}{}_2 = \mathsf{ch}^{(3)}{}_2 \neq \mathsf{ch}^{(2)}{}_2 = \mathsf{ch}^{(4)}{}_2$$

Obviously these transcripts are valid with respect to pk mentioned before. Note that to facilitate the reading of the quantities, the colors have been inserted in the superscripts and subscripts belonging to the same scheme.

At this point is possible to construct the signature, pointing out some issues; the first thing to observe, for example, is that these schemes possess only a soundness error that is constant (in the sense that does not depend on any $\mathcal{MQ}$'s parameters): therefore it becomes necessary to create a construction which has a minor error at the intermediate step, and this is achieved by making a polynomial-number of repeated rounds (therefore $trans_j = (\mathsf{com}_j, \mathsf{ch}_{1,j}, \mathsf{resp}_{1,j}, \mathsf{ch}_{2,j}, \mathsf{resp}_{2,j})$ indicates the j-th round).

Definition 25 (q2-signature) *The q2-signature scheme* ***q2-DSS***(1^k) *is the triplet of algorithms (*KeyGen*,* Sign*,* Verify*) given by the application of the Fiat–Shamir transform on q2 - IDS.*

Specifically, by analyzing the scheme we point out the salient points:

- ◇ The q2-identification scheme $IDS = (\mathsf{KeyGen}, P, V)$ has got soundness error equal to ξ.
- ◇ the challenge spaces of IDS^r, aka C_1^r and C_2^r, have exponential size in $k \in \mathbb{N}$ (security parameter); note that $IDS^r = (\mathsf{KeyGen}, \mathcal{P}^r, \mathcal{V}^r)$ is the parallel composition of IDS, given by the r rounds such that $\xi^r = negl(k)$.
- ◇ The challenges ch_j here are substituted by the values that are given by the hash functions $H_1 : \{0,1\}^* \to C_1^r$ and $H_2 : \{0,1\}^* \to C_2^r$, where the input is the concatenation of the message to be signed and the all $2(j-1)+1$ messages that have been exchanged so far (the signature just contains the messages sent by $\mathcal{P}$).

◇ The steps to construct the signature specifically are summarized in the table below:

Inputs: $(sk, pk) \leftarrow \mathsf{KeyGen}(1^k)$
Protocol: 1) The prover $\mathcal{P}$ compute $\sigma_0 = \mathcal{P}_0^r(sk)$ (and $\sigma_0 := \mathsf{com}$). 1.1) $\mathcal{P}$ compute $h_1 = H_1(m, \sigma_0)$; 2) $\mathcal{P}$ compute $\sigma_1 = \mathcal{P}_1^r(sk, \sigma_0, h_1)$ (and $\sigma_1 := \mathsf{resp}_1$). 2.1) $\mathcal{P}$ compute $h_2 = H_2(m, \sigma_0, h_1, \sigma_1)$; 3) $\mathcal{P}$ compute $\sigma_2 = \mathcal{P}_2^r(sk, \sigma_0, h_1, \sigma_1, h_2)$ (and $\sigma_2 := \mathsf{resp}_2$). 4) The signature is $\sigma = (\sigma_0, \sigma_1, \sigma_2) \leftarrow \mathsf{Sign}(sk, m)$. 5) $\mathsf{Verify}(pk, m, \sigma)$ use σ to computes the values h_1, h_2 as above and outputs $\mathrm{Vr}(pk, \sigma_0, h_1, \sigma_1, h_2, \sigma_2)$

Table 4.2: q2-DSS

4.2 Security and Efficiency

Let's now analyze the security of the signature, considering that we assume as hypothesis that the q2-IDS is HVZK[2], posses the property of soundness (with constant soundness error) and has got a q2-extractor;it has been shown that the following hold:

Theorem 9 *Let $IDS(1^k)$ be a q2-IDS (with $k \in \mathbb{N}$ as security parameter) that is HVZK, achieves soundness with constant soundness error ξ and has a q2-extractor: then $q2 - DSS(1^k)$, the q2-signature scheme explained in the table 4.2 above, is existentially unforgeable under adaptive chosen message attacks.*

In other words, this signature resists at the CMA[3]. Moreover, this signature is also resistant at the KOA, in fact the following holds:

[2]As describe in the definition 8

[3]The Existentially Unforgivability is the ability to create a valid signature for a message chosen by the attacker: the message itself is not important (nor that it has a sensible content) but it is important to be able to create the couple (signature, message). It contrasts with selective forgery and universal forgery, which can provide a valid signature for any type of message.

Theorem 10 *Let $IDS(1^k)$ (with $k \in \mathbb{N}$) be a q2-IDS that achieves soundness with constant soundness error ξ and has a q2-extractor: then $q2 - DSS(1^k)$ (the q2-signature) is unforgeable under key-only attacks.*

Finally, it's important to point out how this signature also respects EU-CMA security, so for this reason the two following lemmas are given:

Lemma 1 (Forking lemma) *Let $DSS(1^k)$ be a q2-signature scheme with security parameter $k \in \mathbb{N}$: if there exists a PPT adversary $\mathcal{A}$ that can output a valid signature message pair (m, σ) with non-negligible success probability, given only the public key as input, then - with non-negligible probability - rewinding $\mathcal{A}$ a polynomial-number of times (with same randomness) but different oracles, outputs four valid signature message pairs $(m, \sigma = (\sigma_0, \sigma_1^{(i)}, \sigma_2^{(i)}))$ with $1 \leq i \leq 4$, such that for the associated hash values it holds that*

$$h_{1,j}^{(1)} = h_{1,j}^{(2)} \neq h_{1,j}^{(3)} = h_{1,j}^{(4)}, \quad h_{2,j}^{(1)} = h_{2,j}^{(3)} \neq h_{2,j}^{(2)} = h_{2,j}^{(4)}$$

for some round $j \in \{1, \ldots, r\}$.

In other words, this lemma tell us that if an opponent can forge a signature with non-negligible probability, then there is a non-negligible probability that the same opponent with the same random tape could create a second forged signature in an attack with a different random oracle; that is to say that if an adversary, on inputs drawn from some distribution, produces an output that has some property with non-negligible probability, then with non-negligible probability, if the adversary is re-run on new inputs but with the same random tape, its second output will also have the property.

Lemma 2 (Conversion of CMA into KOA) *Let $IDS(1^k)$ be a q2-IDS (with $k \in \mathbb{N}$ as security parameter) that is HVZK: then any PPT adversary $\mathcal{B}$ against the EU-CMA security of $q2 - DSS(1^k)$ can be turned into a key-only adversary $\mathcal{A}$ with the properties described in Lemma 1.*
$\mathcal{A}$ runs in polynomial-time and succeeds with essentially the same success probability as $\mathcal{B}$.

4.3 Signature of Chen

In the literature we can find some attempts to create and implement this signature, only the most important are reported here, those of Chen, Hülsing, Rijneveld et al. [7] and Feneuil [8].
In the work of Chen it has been used a 5-ass IDS in $\mathcal{MQ}(n, m, \mathbb{F}_q)$, using as field $\mathbb{F}_{31}$, that allow to have a smaller values for n and m (and of the rounds r too) with respect to $\mathbb{F}_2$ for which $n = m = 256$ in order for it to be reached the 128 bits of

post-quantum security. The authors called their signature MQDSS (Multivariate Quadratic Digital Scheme Signature).

♣ Parameters.

MQDSS is parameterized by a security parameter $k \in \mathbb{N}$, and $m, n \in \mathbb{N}$ such that the security level of the $\mathcal{MQ}$ instance $\mathcal{MQ}(n, m, \mathbb{F}_2) \geq k$;
the length of the function $\mathbf{F}$ is equal to $\texttt{length}(\mathbf{F}) = m \cdot \frac{n \cdot (n+1)}{2}$. Moreover are fixed the following:

- cryptographic hash functions, that are
  $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$,
  $H_1 : \{0,1\}^{2k} \to \mathbb{F}_{31}^r$ and $H_2 : \{0,1\}^{2k} \to \{0,1\}^r$.
- The two string commitment functions, as
  $Com_0 : \mathbb{F}_{31}^n \times \mathbb{F}_{31}^n \times \mathbb{F}_{31}^m \to \{0,1\}^k$ and $Com_1 : \mathbb{F}_{31}^n \times \mathbb{F}_{31}^m \to \{0,1\}^k$.
- The pseudo-random generators, that are
  $G_{\texttt{seed}} : \{0,1\}^k \to \mathbb{F}_{31}^{\texttt{length}(\mathbf{F})}$ for the $\texttt{seed}$,
  $G_{sk} : \{0,1\}^k \to \mathbb{F}_{31}^n$ for the secret key,
  $G_c : \{0,1\}^{2k} \to \mathbb{F}_{31}^{r \cdot (2n+m)}$ for the challenges.

♢ Key generation.

Then a secret key of k bits is randomly sampled as $sk \leftarrow_R \{0,1\}^k$, and the same is done for the $\texttt{seed}$ as $\texttt{seed} \leftarrow_R \{0,1\}^k$.
Furthermore is selected a (pseudo-random) $\mathbf{F}$ from $\mathcal{MQ}(n, m, \mathbb{F}_{31})$ by expanding the $\texttt{seed}$. Observe that are generated $\text{length}(\mathbf{F}) = m \cdot (\frac{n \cdot (n+1)}{2} + n)$ elements for $\mathbf{F}$, to use as coefficients for both the quadratic and the linear monomials; for this aim it's used the pseudo-random generator $G_{\texttt{seed}}$.
In order to compute the public key, the idea is to use the secret key as input for $\mathbf{F}$, but sk is a k-bit string (instead of a sequence of n elements from $\mathbb{F}_{31}$), so it's necessary to use it as a seed for a pseudo-random generator as well, deriving $sk_{\mathbb{F}_{31}} = G_{sk}(sk)$; then it is possible to compute $pk = \mathbf{F}(sk_{\mathbb{F}_{31}})$. The secret key $sk' = (sk, \texttt{seed})$ and the public key $pk' = (\texttt{seed}, pk)$ require $2 \cdot k$ and $k + 5 \cdot m$ bits respectively, assuming 5 bits per $\mathbb{F}_{31}$ element.

♡ Signing.

- The signature algorithm takes as input a message $m \in \{0,1\}^*$ and a secret key $sk' = (sk, \texttt{seed})$; it's calculated also $\mathbf{F} = G_{\texttt{seed}}(\texttt{seed})$.

- Then, it's derived a message-dependent random value $R = \mathcal{H}(sk||m)$ (|| is the string concatenation);
- using R, we compute the randomized message digest $D = \mathcal{H}(R||m)$. The value R must be included in the signature, so that a verifier can derive the same randomized digest.
- The core of the derived signature scheme essentially consists of iterations of the IDS: let compute $G_c(sk, D)$ to obtain the commitment vectors

$$(\mathbf{r}_{(0,0)}, \ldots, \mathbf{r}_{(0,r)}, \mathbf{t}_{(0,0)}, \ldots, \mathbf{t}_{(0,r)}, \mathbf{e}_{(0,0)}, \ldots, \mathbf{e}_{(0,r)}).$$

- Compute for each round i, the values of the string commitment functions $c_{(0,i)}$ and $c_{(1,i)}$ (as defined in the IDS) as Com_0 and Com_1:

$$c_{(0,i)} = \mathsf{Com}_0(\mathbf{r}_{(0,i)}, \mathbf{t}_{(0,i)}, \mathbf{e}_{(0,i)})$$

$$c_{(1,i)} = \mathsf{Com}_1(\mathbf{r}_{(1,i)}, \mathbf{G}(\mathbf{t}_{(0,i)}, \mathbf{r}_{(1,i)}) + \mathbf{e}_{(0,i)}).$$

  Observe that it is not necessary to include all $2r$ commitments in the transcript.
- On the other hand, calculate an hash over the concatenation of all commitments

$$\sigma_0 = \mathcal{H}(c_{(0,0)}||c_{(1,0)}|| \ldots ||c_{(0,r-1)}||c_{(1,r-1)})$$

  .
- Then are derived the challenges $\alpha_i \in \mathbb{F}_{31}$ (for $0 \leq i < r$) by applying H_1 to $h_1 = (D, \sigma_0)$.
- With these values of α_i, can be computed the following vectors

$$\mathbf{t}_{(1,i)} = \alpha_i \cdot \mathbf{r}_{(0,i)} - \mathbf{t}_{(0,i)}$$

$$\mathbf{e}_{(1,i)} = \alpha_i \cdot \mathbf{F}(\mathbf{r}_{(0,i)}) - \mathbf{e}_{(0,i)}.$$

- Compute $\sigma_1 = (\mathbf{t}_{(1,0)}||\mathbf{e}_{(1,0)}|| \ldots ||\mathbf{t}_{(1,r-1)}||\mathbf{e}_{(1,r-1)})$.
- Calculate h_2 by applying H_2 to the tuple $(D, \sigma_0, h_1, \sigma_1)$; then use it as r binary challenges $\mathsf{ch}_{2,i} \in \{0,1\}$.
- Let $\sigma_2 = (\mathbf{r}_{(\mathsf{ch}_{2,i},i)}, \ldots, \mathbf{r}_{(\mathsf{ch}_{2,i},r-1)}, c_{1-\mathsf{ch}_{2,i}}, \ldots, c_{1-\mathsf{ch}_{2,r-1}})$. Note that here we also need to include the challenges $c_{1-\mathsf{ch}_{2,i}}$ that the verifier cannot recompute.
- The obtained signature is $\sigma = (R, \sigma_0, \sigma_1, \sigma_2)$;
  at 5 bits per $\mathbb{F}_{31}$ element, the size of this signature is $(2+r)\cdot k+5\cdot r\cdot(2\cdot n+m)$ bits.

♠ Verification.

- To check, let take as input the message m, the signature $\sigma = (R, \sigma_0, \sigma_1, \sigma_2)$ and the public key $pk' = (\mathtt{seed}, pk)$.
- As in the signing algorithm, through R and m it's possible to compute D, and derive $\mathbf{F}$ from $\mathtt{seed}$ using $G_{\mathtt{seed}}$.
- As the signature contains σ_0, it's possible compose h_1 and, consequentially, the challenge values α_i for all r rounds, by using H_1.
- Similarly, the values $\mathsf{ch}_{2,i}$ are computed by applying H_2 to $(D, \sigma_0, h_1, \sigma_1)$.
- $\forall\ 0 \leq i \leq r$, the verifier calculate $\mathbf{t}_i$ and $\mathbf{e}_i$[4] from σ_1, while compute $\mathbf{r}_i$ from σ_2.
- At this point half of the commitments can be computed, in fact:
  if $\mathsf{ch}_{2,i} = 0$, $\quad c_{(0,i)} = \mathsf{Com}_0(\mathbf{r}_i, \alpha \cdot \mathbf{r}_i - \mathbf{t}_i, \alpha\mathbf{F}(\mathbf{r}_i) - \mathbf{e}_i)$,
  if $\mathsf{ch}_{2,i} = 1$, $\quad c_{(1,i)} = \mathsf{Com}_1(\mathbf{r}_i, \alpha \cdot (pk - \mathbf{F}(\mathbf{r}_i)) - \mathbf{G}(\mathbf{t}_i, \mathbf{r}_i) - \mathbf{e}_i)$.
- At least the verifier extract the missing commitments $c_{(1-\mathsf{ch}_{2,i},i)}$ from σ_2, and can computes

  $$\sigma_0' = H(c_{(0,0)}||c_{(1,0)}|| \ldots ||c_{(0,r-1)}||c_{(1,r-1)}).$$

  The signature is valid if the verifier finds the equality $\sigma_0' = \sigma_0$.

For the sake of completeness, here are reported the complete signature and verification schemes, written in implementable pseudo-code:

[4]Note that $\mathbf{t}_i = \mathbf{t}_{(1,i)}$ and $\mathbf{e}_i = \mathbf{e}_{(1,i)}$, always.

Algorithm 1: MQDSS(sk, pk, A, b)

for $t = 1 : \tau$ **do**
  $\mathsf{seed}^{(t)} \leftarrow \{0,1\}^{\lambda}$
  $r_0^{(t)}, t_0^{(t)} \leftarrow \mathsf{PRG}(\mathsf{seed}^{(t)}) \in \mathbb{F}_q^n, \qquad e_0^{(t)} \leftarrow \mathsf{PRG}(\mathsf{seed}^{(t)}) \in \mathbb{F}_q^m$
  $r_1^{(t)} \leftarrow sk - r_0^{(t)}$
  $c_0^{(t)} \leftarrow \mathsf{com}(r_0^{(t)}, t_0^{(t)}, e_0^{(t)})$
  $c_1^{(t)} \leftarrow \mathsf{com}(r_1^{(t)}, G(t_0^{(t)}, r_1^{(t)}) + e_0^{(t)})$
  $\sigma_0 \leftarrow ((c_0^{(t)}, c_1^{(t)})_{t\in[\tau]})$
// First challenge
$h_1 \leftarrow Hash_1(msg, \sigma_0) \in \mathbb{F}_q$
// First response
for $t = 1 : \tau$ **do**
  $t_1^{(t)} \leftarrow h_1 \cdot r_0^{(t)} - t_0^{(t)}$
  $e_1^{(t)} \leftarrow h_1 \cdot F(r_0^{(t)}) - e_0^{(t)}$
  $\sigma_1 \leftarrow ((t_1^{(t)}, e_1^{(t)})_{t\in[\tau]})$
// Second challenge
$h_2 \leftarrow Hash_2(msg, \sigma_0, h_1, \sigma_1) \in \{0,1\}$
// Second response
if $h_2 == 0$ **then**
  $\sigma_2 \leftarrow (r_0^{(t)})_{t\in[\tau]}$
else
  $\sigma_2 \leftarrow (r_1^{(t)})_{t\in[\tau]}$
$\sigma \leftarrow (\sigma_0, \sigma_1, \sigma_2)$
return σ

Figure 4.1: Signing algorithm of MQDSS, based on the Chen's 5-pass identification scheme

Algorithm 2: Verif(σ, pk, A, b, msg)

Recompute σ_0, σ_1 and σ_2 from σ
Compute $h'_1 \leftarrow Hash_1(msg, \sigma_0) \in \mathbb{F}_q$
Compute $h'_2 \leftarrow Hash_2(msg, \sigma_0, h_1, \sigma_1) \in \{0, 1\}$
if $mod(h_2, 2) == 0$ **then**
    for $t = 1 : \tau$ **do**
        Compute $F(r_0^{(t)})$ from $\sigma_2 \leftarrow (r_0^{(t)})_{t\in\tau}$
        Consider $\sigma_1 \leftarrow ((t_1^{(t)}, e_1^{(t)})_{t\in[\tau]})$
        $(c'_0)^{(t)} \leftarrow \mathsf{com}(r_0^{(t)}, h'_1 \cdot r_0^{(t)} - t_1^{(t)}, h'_1 \cdot F(r_0^{(t)}) - e_1^{(t)})$
    Consider $\sigma_0 \leftarrow ((c_0^{(t)}, c_1^{(t)})_{t\in[\tau]})$
    if $(c_0^{(t)} == (c'_0)^{(t)})_{\forall t\in[\tau]}$ **then**
        return Accept
    else
        return Reject
if $mod(h_2, 2) == 1$ **then**
    for $t = 1 : \tau$ **do**
        Compute $F(r_1^{(t)})$ and $G(t_1^{(t)}, r_1^{(t)})$ from $\sigma_2 \leftarrow (r_1^{(t)})_{t\in[\tau]}$
        Consider $\sigma_1 \leftarrow ((t_1^{(t)}, e_1^{(t)})_{t\in[\tau]})$
        $(c'_1)^{(t)} \leftarrow \mathsf{com}(r_1^{(t)}, h'_1 \cdot (pk - F(r_1^{(t)})) - G(t_1^{(t)}, r_1^{(t)}) - e_1^{(t)})$
    Consider $\sigma_0 \leftarrow ((c_0^{(t)}, c_1^{(t)})_{t\in[\tau]})$
    if $(c_1^{(t)} == (c'_1)^{(t)})_{\forall t\in[\tau]}$ **then**
        return Accept
    else
        return Reject

Figure 4.2: Verification algorithm of MQDSS, based on the Chen's 5-pass identification scheme

4.3.1 Security of the signature

The authors [7] of signature security analysis provide only asymptotic statement: while this does not suffice to make any statement about the security of a *specific* parameter choice, it provides evidence that the general approach leads a secure scheme.

Theorem 11 (Optimization) *MQDSS is EU-CMA-secure in the random oracle model (ROM in short), if:*

- *the search version of the $\mathcal{MQ}$ problem is intractable,*
- *the hash functions $\mathcal{H}$, H_1 and H_2 are modeled as random oracles (RO),*
- *the commitment functions Com_0 and Com_1 are computationally binding, computationally hiding, and the probability that their output takes a given value is negligible in the security parameter,*
- *the pseudo-random generator $G_{\mathtt{seed}}$ is modeled as random oracle, and*
- *the pseudo-random generators, G_{sk}, and G_c have outputs computationally indistinguishable from random.*

Theorem 12 *Let IDS(1^k) (with $k \in \mathbb{N}$ as security parameter) be a q2-IDS that is HVZK, achieves soundness with constant soundness error ξ and has a q2-extractor: then opt-q2-DSS(1^k), the optimized q2-signature scheme (derived as in the table 4.2) and the optimization explained in the theorem 11, is existentially unforgeable under adaptive chosen message attacks.*

4.4 Signature of Feneuil

As for the signature of Feneuil, the approach that was used is slightly different: it is in fact based on the use of the Multiparty Computation in the Head, or in short MPCitH. We know, from the chapter 2, that a zero-knowledge protocol combined with the Fiat–Shamir transform gives us a digital signature scheme, taking away its interactivity; while Chen [7] used as a Zero-Knowledge protocol one derived from the identification scheme, Feneuil here decided to approach it with an MPC in the version *in the head*, so that the final signature scheme could be more optimized.
For this reason, let's rewrite some concepts, in order to fully understand the process follow by the author [8].
The *MPCitH* is an MPC protocol in which there are N parties, that are denoted by $\mathcal{P}_1, \ldots, \mathcal{P}_N$, that have the task of evaluate a function f in a secure and correct way on a secret input x; in particular, must be respected the following characteristics:

- the secret x is encoded as a sharing $[\![x]\!]$ and each $\mathcal{P}_i$ takes a share $[\![x_i]\!]$ as input;
- the function f outputs `accept` or `reject`;
- the views of t parties leak no information about the secret x.

Definition 26 *Let $[\![x]\!]$ denote an additive sharing of x, i.e. a sharing of x consists of random $x_1, \ldots, x_N \in \mathbb{F}_q$ such that $x = \sum_{i\in[N]} x_i$, where $\mathcal{P}_i$ holds x_i.*

Keep in mind that all communications occurring between the different parties are broadcast. In this context, the protocol is carried out as follows: the prover has to build a random sharing $[\![x]\!]$ of x, to simulate locally (so *in his/her head*) all the parties of the MPC protocol and then send the commitments to each party's view (aka party's input share), and both the secret random tape and messages sent and received to the verifier; finally the prover has to send the output shares $[\![f(x)]\!]$ of the parties (which should correspond to `accept`). At this point the verifier randomly chooses t parties and asks the prover to reveal their views: after receiving them, the verifier checks that they are consistent with an honest execution of the MPC protocol and with the commitments.
Observe that with only t views, the verifier do not possess any information on the complete secret x and the probability on a successfully cheating is less than $\frac{N-t}{N}$; moreover the protocol takes as input an additive sharing of a candidate solution of the studied problem, and eventually an additive sharing of auxiliary data. It is assumed, of course, that the secret x is additive and also that each party performs an arbitrary number of times the three actions *Receiving randomness*, *Receiving hint* and *Computing & broadcasting*, here described:

- in the action of *Receiving randomness* the parties expect to receive a random value ϵ from a randomness oracle $\mathcal{O}_R$.(when calling this oracle, all the parties get the same random value ϵ); this means that the parties get only once a common random value from the oracle.
- in the *Receiving hint* the parties can receive a sharing $[\![\beta]\!]$ (one per each of them) from a hint oracle $\mathcal{O}_H$;
  observe that the hint β can depend on the witness w and the previous random values sampled from $\mathcal{O}_R$.
- Finally in the action of *Computing & broadcasting* the parties can locally compute $[\![\alpha]\!] := [\![\phi(v)]\!]$ from their sharing $[\![v]\!]$ (note that ϕ is an $\mathbb{F}$-linear function), then broadcast all the shares $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ in order to reconstruct $\alpha := \phi(v)$ publicly. The function ϕ can depend on the previous random values $\{\epsilon^i\}_i$ from $\mathcal{O}_R$ and on the previous broadcasted values.

Note finally that is defined a *false positive rate* when the function f outputs `accept` when in reality it shouldn't: this happens with probability at most p, considering the randomness of the oracle.

Applying the MPC-in-the-Head paradigm to this MPC protocol results in a 5-round zero-knowledge proof of knowledge and, as repeatedly said, a signature scheme is obtained after applying the Fiat–Shamir transform: in this particular case, the forgery cost the signature is

$$cost_{forge} := \min_{\substack{r_1,r_2: \\ r_1+r_2=r}} \left\{ \frac{1}{\sum_{i=r_1}^{r} \binom{r}{i} p^i (1-p)^{r-i}} + N^{r_2} \right\} \tag{4.1}$$

where r is the number of the rounds (or parallel executions) and $p = \frac{2}{q^\eta} - \frac{1}{q^{2\eta}}$.

In the author's [8] MPC protocol, it will be necessary to be able to verify the correctness of the product between different matrices, for this reason, before dealing with the scheme, we still give the definition of the protocol that deals with this type of matrix verification:

Definition 27 *The Matrix Multiplication Checking Protocol, denoted by Π^η_{MM}, is a protocol that checks if, given the three matrices X, Y and Z, is satisfied the relation $Z = X \cdot Y$.*

Observe that the protocol Π^η_{MM}, that we allow ourselves to indicate it with Π^η to lighten the notation (MM stands for *Matrix Multiplication*), depends on the parameter η that is an arbitrary dimension involved in the decomposition of the matrix.
Since the matrix multiplication protocol is a variant of the multiplication checking protocol already presented and exposed by Baum and Nof [10], it is reported first, and then the Feneuil's integrative variant.

Let now define the operation that will be used in the next schemes:

- ⊛ The operation `open`($[\![x]\!]$) means that, to reveal the secret x, each party broadcasts its share x_i; upon receiving x_j from each $\mathcal{P}_j$, $\mathcal{P}_i$ sets $x = \sum_{j\in[N]} x_j$.
- ⊛ The operation of the sum $[\![x]\!] + [\![y]\!]$ means that given the two shares x_i and y_i of x and y, each party $\mathcal{P}_i$ defines $x_i + y_i$ as its share of the result.
- ⊛ The operation $\sigma \cdot [\![x]\!]$ means that, given a sharing $[\![x]\!]$ and a public constant σ, each party $\mathcal{P}_i$ defines $\sigma \cdot x_i$ as its share of the product.
- ⊛ The operation $\sigma + [\![x]\!]$ means that, given a sharing $[\![x]\!]$ and a public constant σ, $\mathcal{P}_1$ defines $x_1 + \sigma$ as its new share while other parties' shares remain the same.

So according to Baum, in a MPC context is given in input the triplet $([\![a]\!], [\![b]\!], [\![c]\!])$ (where $c = a \cdot b$) and the sharing $[\![x]\!]$ and $[\![y]\!]$, while the parties have to verify that $[\![z]\!] = [\![x]\!] \cdot [\![y]\!] = [\![xy]\!]$: then

1. the parties compute $[\![\alpha]\!] = [\![x]\!] - [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] - [\![b]\!]$;
2. then each party runs $\texttt{open}([\![\alpha]\!])$ and $\texttt{open}([\![\beta]\!])$ to obtain α and β;
3. the parties compute $[\![z]\!] = [\![c]\!] - \alpha \cdot [\![b]\!] - \beta \cdot [\![a]\!] + \alpha \cdot \beta$.

With this protocol as introduction, the schema Π^η is now given:

Inputs: Each party takes a share of the following sharings as inputs:
$[\![X]\!]$ where $X \in \mathbb{F}_q^{m \times p}$
$[\![Y]\!]$ where $Y \in \mathbb{F}_q^{p \times n}$
$[\![Z]\!]$ where $Z \in \mathbb{F}_q^{m \times n}$
$[\![A]\!]$ where A has been uniformly sampled from $\mathbb{F}_q^{p \times \eta}$
$[\![C]\!]$ where $C \in \mathbb{F}_q^{m \times \eta}$ satisfies $C = XA$.

Protocol:

1. The parties get a random $\Sigma \in \mathbb{F}_q^{n \times \eta}$;
2. The parties locally set $[\![D]\!] = [\![Y]\!]\Sigma + [\![A]\!]$;
3. The parties broadcast $[\![D]\!]$ to obtain $D \in \mathbb{F}_q^{p \times \eta}$;
4. The parties locally set $[\![V]\!] = [\![X]\!]D - [\![C]\!] - [\![Z]\!]\Sigma$;
5. The parties open $[\![V]\!]$ to obtain $V \in \mathbb{F}_q^{m \times \eta}$;
6. The parties outputs `accept` if $V = 0$ and `reject` otherwise.

Table 4.3: MPC protocol Π^η

Note that the check is done on $V = 0$ because $V = XD - C - Z\Sigma = XY\Sigma + XA - XA - XY\Sigma = (XY - XY)\Sigma$, in a lighter notation.
The authors [10] also give demonstration of the following lemma on the cheating probability:

Lemma 3 *If $Z = X \cdot Y$ and if C are genuinely computed, then Π^η always outputs* `accept`*; if $Z \neq X \cdot Y$, then Π^η outputs* `accept` *with probability at most $\frac{1}{q^\eta}$.*

In this scenario, it is necessary to give the definition of Multivariate Quadratic Problem again, in order to be consistent with the notation of Feneuil [8]:

$\mathcal{MQ}$ problem
Let $m \in \mathbb{N}^*$ for which there are $(A_i)_{i \in [m]}$, $(b_i)_{i \in [m]}$, x and y be such that:

- x is uniformly sampled from $\mathbb{F}_q^n$, where $n \in \mathbb{N}^*$ and $\mathbb{F}_q$ is a finite field with q elements;
- $\forall i \in [m]$, A_i is uniformly sampled from $\mathbb{F}_q^{n \times n}$;
- $\forall i \in [m]$, b_i is uniformly sampled from $\mathbb{F}_q^n$
- $\forall i \in [m]$, y_i is defined as $y_i := x^T A_i x + b_i^T x$.

From $((A_i)_{i\in[m]}, (b_i)_{i\in[m]}, y)$, find x.

The protocol based on this hard problem sees the prover as the one that has to convince the verifier that knows $x \in \mathbb{F}_q^n$ such that

$$\begin{cases} y_1 = x^T A_1 x + b_1^T x \\ \vdots \\ y_m = x^T A_m x + b_m^T x \end{cases} \tag{4.2}$$

Let now give the MPC protocol, MQ based:

Public values: The matrices $A_1, \ldots, A_m \in \mathbb{F}_q^{n \times n}$, the vectors $b_1, \ldots, b_m \in \mathbb{F}_q^n$, the outputs $y_1, \ldots, y_m \in \mathbb{F}_q$.
Inputs: Each party takes a share of the following sharings as inputs: $[\![x]\!]$ where $x \in \mathbb{F}_q^n$ $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_{q^\eta}^n$ $[\![c]\!]$ where $c \in \mathbb{F}_{q^\eta}$ satisfies $c = -\langle a, x\rangle$.
Protocol: 1. The parties get a random $\gamma_1, \ldots, \gamma_m \in \mathbb{F}_{q^\eta}$ and a random $\varepsilon \in \mathbb{F}_{q^\eta}$; 2. The parties locally set $[\![z]\!] = \sum_{i=1}^m \gamma_i (y_i - b_i^T [\![x]\!])$; 3. The parties locally set $[\![w]\!] = (\sum_{i=1}^m \gamma_i A_i)[\![x]\!]$; 4. The parties locally set $[\![\alpha]\!] = \varepsilon \cdot [\![w]\!] + [\![a]\!]$; 5. The parties open $\alpha \in \mathbb{F}_{q^\eta}^n$; 6. The parties locally set $[\![v]\!] = \varepsilon \cdot [\![z]\!] - \langle \alpha, [\![x]\!]\rangle - [\![c]\!]$; 7. The parties open $v \in \mathbb{F}_{q^\eta}$; 8. The parties outputs `accept` if $v = 0$ and `reject` otherwise.

Table 4.4: Feneuil's MPC protocol

The main idea under this passages is that instead of checking the m relationships of the system 4.2 separately, it's possible to group them in a linear combination where the coefficients $\gamma_1, \ldots, \gamma_m$ are uniformly sampled (in $\mathbb{F}_{q^\eta}$); in this way the new check to be performed will be

$$\sum_{i=1}^{m} \gamma_i (y_i - x^T A_i x - b_i^T x) = 0.$$

It is possible to make some algebraic steps, to remodel the equation: in fact

$$\sum_{i=1}^{m} \gamma_i (y_i - x^T A_i x - b_i^T x) = 0 \Longrightarrow \sum_{i=1}^{m} \gamma_i (y_i - b_i^T x) = \sum_{i=1}^{m} \gamma_i (x^T A_i x)$$

For the second term of the equation it is possible to write

$$\sum_{i=1}^{m} \gamma_i (x^T A_i x) = x^T \Big(\sum_{i=1}^{m} \gamma_i A_i \Big) x = \langle x, w \rangle$$

where $w := \Big(\sum_{i=1}^{m} \gamma_i A_i \Big) x$.
In this way the initial check became

$$z = \langle x, w \rangle$$

where it has been defined the quantity $z := \sum_{i=1}^{m} \gamma_i (y_i - b_i^T x)$.

Given the MPC in the head protocol, I explicitly report Feneuil's signature and verification scheme, together with the keys generation algorithms:

Algorithm 3: KeyGen()

$\mathsf{seed}_{\mathsf{pk}}, \mathsf{seed}_{\mathsf{sk}} \leftarrow \{0,1\}^\lambda \times \{0,1\}^\lambda$

Generate $(A_i)_{i\in[m]} \in \mathbb{F}_q^{n\times n}$ and $(b_i)_{i\in[m]} \in \mathbb{F}_q^n$ from $\mathsf{seed}_{\mathsf{pk}}$ with a secure PRG.

Generate $x \in \mathbb{F}_q^n$ from $\mathsf{seed}_{\mathsf{sk}}$ with a secure PRG.

$\forall i \in [m], y_i \leftarrow x^T A_i x + b_i^T x$

$\mathsf{pk} \leftarrow ((y_i)_{i\in[m]}, \mathsf{seed}_{\mathsf{pk}})$

$\mathsf{sk} \leftarrow \mathsf{seed}_{\mathsf{sk}}$

return $(\mathsf{pk}, \mathsf{sk})$

Algorithm 4: DecompressPK(pk)

$y_i, \mathsf{seed}_{\mathsf{pk}} \leftarrow \mathsf{pk}$

Regenerate $(A_i)_{i\in[m]}$ and $(b_i)_{i\in[m]}$ from $\mathsf{seed}_{\mathsf{pk}}$

return $(y_i, A_i, b_i)_{i\in[m]}$

Algorithm 5: DecompressSK(sk)

$\mathsf{seed}_{\mathsf{sk}} \leftarrow \mathsf{sk}$

Regenerate $x \in \mathbb{F}_q^n$ from $\mathsf{seed}_{\mathsf{sk}}$

return x

Figure 4.3: Algorithms for the generation, the compression and the decompression of the public and secret keys in the $\mathcal{MQ}$ problem.

Algorithm 1: $\mathsf{Sign}((\mathbf{A}_1,\ldots,\mathbf{A}_m),(\mathbf{b}_1,\ldots,\mathbf{b}_m),(y_1,\ldots,y_m),x,\mathsf{msg})$

$\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$

// Phase 1: Set up the views for the MPC protocol

for $t = 1:\tau$ **do**
  $\mathsf{seed}^{(t)} \leftarrow \{0,1\}^{\lambda}, \quad (\mathsf{seed}_i^{(t)})_{i\in[\mathsf{N}]} \leftarrow \mathsf{TreePRG}(\mathsf{salt},\mathsf{seed}^{(t)})$
  for $i = 1:N-1$ **do**
    $[\![a^{(t)}]\!]_i, [\![x^{(t)}]\!]_i, [\![c^{(t)}]\!]_i \leftarrow \mathsf{PRG}(\mathsf{salt},\mathsf{seed}_{\mathsf{i}}^{(t)})$
    $\mathsf{state}_i^{(t)} \leftarrow \mathsf{seed}_i^{(t)}$
  $[\![a^{(t)}]\!]_N \leftarrow \mathsf{PRG}(\mathsf{salt},\mathsf{seed}_N^{(t)}), \quad [\![x^{(t)}]\!]_N \leftarrow x - \sum_{i=1}^{N-1}[\![x^{(t)}]\!]_i$
  $a \leftarrow \sum_{i=1}^{N}[\![a^{(t)}]\!]_i$
  $[\![c^{(t)}]\!]_N \leftarrow -\langle a,x\rangle - \sum_{i=1}^{N-1}[\![c^{(t)}]\!]_i$
  $\mathsf{aux}^{(t)} \leftarrow ([\![x^{(t)}]\!]_N, [\![c^{(t)}]\!]_N), \quad \mathsf{state}_N^{(t)} \leftarrow (\mathsf{seed}_N^{(t)},\mathsf{aux}^{(t)})$
  $\mathsf{com}_i^{(t)} \leftarrow \mathrm{Hash}_0(\mathsf{salt},t,i,\mathsf{state}_i^{(t)}) \quad \forall\, i = 1:N$

// Phase 2: First challenges

$h_1 \leftarrow \mathrm{Hash}_1(\mathsf{msg},\mathsf{salt},(\mathsf{com}_i^{(t)})_{i\in[N],t\in\tau})$
$(\gamma_1,\ldots,\gamma_m,\varepsilon)^{(1)},\ldots,(\gamma_1,\ldots,\gamma_m,\varepsilon)^{(\tau)} \leftarrow \mathsf{PRG}(h_1)$

// Phase 3: Simulation of the MPC protocol

for $t = 1:\tau$ **do**
  Compute $[\![z^{(t)}]\!] \leftarrow \sum_{j=1}^{m}\gamma_j(y_j - \mathbf{b}_j^T[\![x^{(t)}]\!]), \quad [\![w^{(t)}]\!] \leftarrow \sum_{j=1}^{m}(\gamma_j\mathbf{A}_j)[\![x^{(t)}]\!]$
  Set $[\![\alpha^{(t)}]\!] \leftarrow \varepsilon^{(t)}\cdot[\![w^{(t)}]\!] + [\![a^{(t)}]\!]$
  Open $[\![\alpha^{(t)}]\!]$, aka $\alpha^{(t)} \leftarrow \sum_{i=1}^{N}[\![\alpha^{(t)}]\!]_i$
  Compute $[\![v^{(t)}]\!] \leftarrow \varepsilon^{(t)}\cdot[\![z^{(t)}]\!] - \langle\alpha^{(t)},[\![x^{(t)}]\!]\rangle - [\![c^{(t)}]\!]$

// Phase 4: Second challenges

$h_2 \leftarrow \mathrm{Hash}_2(\mathsf{msg},\mathsf{salt},h_1,([\![\alpha^{(t)}]\!]_i,[\![\mathbf{v}^{(t)}]\!]_i)_{i\in[N],t\in\tau})$
$(i^*)^{(1)},\ldots,(i^*)^{(\tau)} \leftarrow \mathsf{PRG}(h_2)$

// Phase 5: Assembling the signature σ

$\sigma \leftarrow \left(\mathsf{salt},h_1,h_2,[(\mathsf{state}_i^{(t)})\forall i \neq (i^*)^{(t)},\mathsf{com}_{(i^*)^{(t)}}^{(t)},[\![\alpha^{(t)}]\!]_{(i^*)^{(t)}}]_{\forall t=1:\tau}\right)$
return σ

Figure 4.4: Signing algorithm of $\mathcal{MQ}$ in the head, based on the Feneuil's protocol. Please note that the matrices $(\mathbf{A}_1,\ldots,\mathbf{A}_m)$ are in $\mathbb{F}_q^{n\times n}$, the vectors$(\mathbf{b}_1,\ldots,\mathbf{b}_m)$ are in $\mathbb{F}_q^n$, the outputs $(y_1,\ldots,y_m)$ are in $\mathbb{F}_q$ and x is in $\mathbb{F}_q^n$; msg is in $\{0,1\}^*$

Algorithm 2: Verif$((\mathbf{A}_1,\dots,\mathbf{A}_m),(\mathbf{b}_1,\dots,\mathbf{b}_m),(y_1,\dots,y_m),$ msg, $\sigma)$

$(\gamma_1,\dots,\gamma_m,\varepsilon)^{(1)},\dots,(\gamma_1,\dots,\gamma_m,\varepsilon)^{(\tau)} \leftarrow \mathsf{PRG}(h_1)$,
$(i^*)^{(1)},\dots,(i^*)^{(\tau)} \leftarrow \mathsf{PRG}(h_2)$

for $t = 1 : \tau$ **do**
  $\forall i \neq (i^*)^{(t)}$, compute $\mathsf{com}_i^{(t)}$ as in line 11 of Sign()
  $\forall i \neq (i^*)^{(t)}$, compute $[\![\alpha^{(t)}]\!]$ as in lines 15 and 16 of Sign()
  $\alpha^{(t)} \leftarrow \sum_{i=1}^{N} [\![\alpha^{(t)}]\!]_i$
  $\forall i \neq (i^*)^{(t)}$, compute $[\![v^{(t)}]\!]_i$
  $[\![v^{(t)}]\!]_{(i^*)^{(t)}} \leftarrow -\sum_{i\neq(i^*)^{(t)}} [\![v^{(t)}]\!]_i$

$h'_1 = \mathrm{Hash}_1(\mathsf{msg},\mathsf{salt},(\mathsf{com}_i^{(t)})_{i\in[N],t\in\tau})$

$h'_2 = \mathrm{Hash}_2(\mathsf{msg},\mathsf{salt},h_1,([\![\alpha^{(t)}]\!]_i,[\![\mathsf{v}^{(t)}]\!]_i)_{i\in[N],t\in\tau})$

Output **accept** (bit=1) if $h'_1 = h_1$ and $h'_2 = h_2$, otherwise output **reject** (bit=0)

Figure 4.5: Verification algorithm of $\mathcal{MQ}$ in the head, based on the Feneuil's protocol.

4.4.1 Efficiency

It has been demonstrated that this complete MPC protocol posses a cheating probability of

$$\frac{1}{q^\eta} + \Big(1 - \frac{1}{q^\eta}\Big)\frac{1}{q^\eta} = \frac{2}{q^\eta} - \frac{1}{q^{2\eta}}$$

Moreover, the soundness error of the resulting protocol is

$$\varepsilon := \frac{1}{N} + \Big(1 - \frac{1}{N}\Big)\Big(\frac{2}{q^\eta} - \frac{1}{q^{2\eta}}\Big).$$

As before, if the protocol is repeated r times, the soundness error goes down to ε^r; note that to obtain a soundness error of λ bits, it's required to impose $r = \lceil \frac{-\lambda}{\log_2 \varepsilon} \rceil$.

Once applied the Fiat–Shamir transform, the resulting signature scheme in this case will have the security cost seen in 4.1, while the communication cost is

$$4\lambda + r \cdot \Big(n \cdot \log_2(q) + n \cdot \eta \cdot \log_2(q) + \eta \cdot \log_2(q) + \lambda \cdot \log_2 N + 2\lambda\Big)$$

where λ is the security level, η is a scheme parameter and r is computed such that the soundness error is of λ bits in the interactive case and such that $cost_{forge}$ is of λ bits in the non-interactive case.

4.5 Signature attacks

In post-quantum cryptography, the $\mathcal{MQ}$ problem constitutes the elementary unit of the Multivariate public key cryptosystems, in short called MPKCs; the cryptosystems of this type, used for signatures, are divided into two categories: the *trapdoor* multivariate signature schemes, built upon a trapdoor multivariate polynomial map, and the *one-way* multivariate signature schemes, based on IDSs like the ones analysed before, for which security is directly based on the difficulty of the $\mathcal{MQ}$ problem itself.
We know that the $\mathcal{MQ}$ problem is known to be NP-complete and appears to be hard on average for a wide range of parameters, but despite its obvious difficulty, there are a considerable number of algorithms that can solve the $\mathcal{MQ}$ problem. So what degree of security do cryptographic systems that are based on this problem possess? Fortunately, the complexity of these solving algorithms depends on several values: hence depending on the variation of these values, let's now analyze some of these algorithms.
Note that the following algorithms will rely only on find a possible solution (if there is any) of the problem, also known as the *search* version of the problem.
Please also note that will be discussed two types of algorithms, depending on whether the system is determined or not.

Definition 28 *A system of equations is said to be determined if it has as many unknowns as equations, so referring to the notation of the system 3.1, it has got $m = n$. If instead $n > m$, the system is called underdetermined.*

In the case of an underdetermined system, one can either fall back to a determined system, or use the appropriate algorithms designed considering the difference between the dimension. Observe that it is possible to transform an underdetermined system into a determined one, so that a solution of the former can be found from one of the latter; two ways usually used are:

- using the algorithm call *fixing variables*, which indeed involves fixing $n - m$ unknowns to obtain a $m \times m$ system;

- using the algorithm call *Thomae–Wolf and Improvements*, which involves fixing a a number of equations such that an equivalent system of $m - \lfloor \alpha \rfloor + a$ unknowns and equations is obtained.

As for the specific algorithms for underdetermined systems, we have:

KPG Kipnis, Patarin, and Goubin, the authors of this algorithm, suggest this procedure that works when the dimensions respect the inequality $n > m(m+1)$: in particular, the goal is to solve the system

$$\sum_{i=1}^{m} a_{i,1} y_i^2 + \sum_{i=1}^{m} y_i L_{i,1}(y_{m+1}, \ldots, y_n) + Q_1(y_{m+1}, \ldots, y_n) = 0$$

$$\vdots$$

$$\sum_{i=1}^{m} a_{i,m} y_i^2 + \sum_{i=1}^{m} y_i L_{i,m}(y_{m+1}, \ldots, y_n) + Q_m(y_{m+1}, \ldots, y_n) = 0$$

where $(y_1, \ldots, y_n) = \mathbf{S}(x_1, \ldots, x_n)$ with $\mathbf{S}$ variable change matrix - non singular - (that is to find), $Q_i(\cdot)$ quadratic maps and $L_{i,j}(\cdot)$ linear maps.
The time complexity is $\mathcal{O}(mn^\omega)$, where ω denotes the exponent in the complexity of matrix multiplication, or in other words there is an algorithm that multiplies two $n \times n$ matrices with $\mathcal{O}(n^\omega)$ operations. The space complexity, instead, is given by $\mathcal{O}(mn^2)$.

MHT This algorithm, suggested by Miura, Hashimoto, and Takagi, works instead for $n \geq \frac{m(m+3)}{2}$; its time complexity is

$$\begin{cases} \mathcal{O}(n^\omega m) & \text{if } q \text{ even} \\ \mathcal{O}(2^m n^\omega m) & \text{if } q \text{ odd} \end{cases}$$

where $q$is the characteristic of the field; moreover the space complexity is dominated by the memory required to store the initial set of polynomials.
It has been proposed a generalization of this algorithm, by Huang and Bao, that works for $n \geq \frac{m(m+1)}{2}$ and has time complexity as

$$\begin{cases} \mathcal{O}(q(\log q)^2 \cdot n^\omega m) & \text{if } q \text{ even} \\ \mathcal{O}(q(\log q)^2 \cdot 2^m n^\omega m) & \text{if } q \text{ odd} \end{cases}$$

CGMT-A Courtois, Goubin, Meier and Tacier proposed their "Algorithm A", which is so structured:

- ⊞ $\forall j \in [1, m]$, the polynomials that characterize the underlying system are rewritten as $f_j(x_1, \ldots, x_n) = g_j(x_1, \ldots, x_k) + h_j(x_{k+1}, \ldots, x_n) + \sum_{i=1}^{k} L_{i,j}(x_{k+1}, \ldots, x_n)x_i$, where $L_{i,j}$ are linear maps randomly chosen in $\mathbb{F}_q$;
- ⊞ $\forall j \in [1, 2k]$ let us rewrite also $g'_j(x_1, \ldots, x_n) = g_j(x_1, \ldots, x_k) + \sum_{i=1}^{k} L_{i,j}(x_{k+1}, \ldots, x_n)x_i$: then one has to compute the q^k vectors of the set $\mathcal{G} = \{-(g'_1(a), \ldots, g'_{2k}(a)) : a \in \mathbb{F}_q^k\}$ along with its corresponding preimage a.
- ⊞ Finally, one has to find the vector $b \in \mathbb{F}_q^{n-k}$ such that $\{L_{i,j}(b) = L_{i,j}(x_{k+1}, \ldots, x_n)\}_{i,j=0}^{k,2k} \wedge (h_1(b), \ldots, h_{2k}(b)) \in \mathcal{G}$.

If $k = min[\frac{m}{2}, \sqrt{\frac{n}{2} - \sqrt{\frac{n}{2}}}]$, then the algorithm has got the time complexity equal to $\mathcal{O}\Big(2k\binom{n-k}{2}q^{m-k}\Big)$, while the space complexity is $\mathcal{O}(2kq^k)$.

Another way to search the solutions of the $\mathcal{MQ}$ problem is to use the *Gröbner basis algorithms*, that are based on the formulation of an appropriate Gröbner basis, knowing that a solution to the $\mathcal{MQ}$ problem $f_1 = \cdots = f_m = 0$ can be efficiently extracted from a Gröbner basis G_{lex}, in the lexicographic order, of the ideal $I = \langle f_1, \ldots, f_m \rangle$ (for a complete definition see A.4). Note that it's assumed that the sequence of polynomials is either regular or semi-regular, for this reason are given the following:

Definition 29 *The degree of regularity of a homogeneous ideal $I \subseteq \mathbb{F}_q[x]$ is the minimum integer d, if any, such that $dim(I_d) = dim(R_d)$, where $I_d = R_d \cap I$, R_d is the set of elements in R of degree d.*

Definition 30 *A homogeneous sequence $\mathcal{F} \in \mathbb{F}_q[x]^m$ is called semi-regular if*

$$\sum_{d \geq 0} dim(R_d/I_d)z^d = \Big[\frac{(1-z^q)^n}{(1-z)^n}\Big(\frac{1-z^2}{1-z^{2q}}\Big)^m\Big]_+$$

where $[H(z)]_+$ means that the series $H(z)$ is cut from the first non-positive coefficient. An affine sequence $\mathcal{G} = (g_1, \ldots, g_m)$ is semi-regular if the sequence $(\widetilde{g_1}, \ldots, \widetilde{g_m})$ does, where $\widetilde{g_i})$ is the homogeneous part of g_i of highest degree.

Through this type of procedure are given some algorithms such as F4 and F5, in which the monomial order in the underlying polynomial ring is the graded reverse lexicographic (grevlex) monomial order, or the algorithm given by Bardet, Faugère, and Salvy, used for homogeneous ideal of polynomials in the grevlex order.

One more algorithm is the one used for the exhaustive search, denoted by FES (aka Fast Exhaustive Search): this performs an exhaustive search over $\mathbb{F}_2^n$ by enumerating the solution space with Gray codes[5]. This algorithm posses a time complexity of $4\log(n)\left(\frac{2^n}{n_{sol}+1}\right)$, where n_{sol} is the number of solutions of a given $\mathcal{MQ}$ instance, while the space complexity is $\mathcal{O}(n^2m)$; the extension in a field $\mathbb{F}_q$ (that use the q-ary Gray codes) has (time) complexity $\mathcal{O}\Big(\log_q(n)\frac{q^n}{n_{sol}+1}$.

The last two categories of algorithms are hybrid algorithms, which combine a partial exhaustive search with other procedures, and probabilistic algorithms, which asymptotically outperform the FES.
As for the part of the hybrid ones, there is, for example, the algorithm so called `BooleanSolve`, that give an estimation of the number of the variables iteratively and check the consistency of the resulting problem: this verification is done by checking the consistency of the system made up by the original one but with $n-k$ variables, by using the Macaulay matrix[6] at large enough degree; its time complexity is $q^k \cdot \{\tilde{\mathcal{O}}\Big(\Big(\frac{n-k+d_{wit}}{d_{wit}}\Big)^\omega\Big)$ where $d_{wit} = deg(\Big[\frac{(1-zq)^{n-k}}{(1-z)^{n-k+1}}\Big(\frac{1-z^2}{1-z^{2q}}\Big)^m\Big]_+) + 1$.
Yet another hybrid algorithm is guess a set of the k variables and then solve the resulting system by applying one of the algorithm F4 or F5: the time complexity is $q^k \cdot \mathcal{O}\Big(\Big(\frac{n-k+d_{reg}}{d_{reg}}\Big)^\omega + nn_{sol}^3\Big)$. Finally there is the `Crossbread` algorithm, which is very similar to the `BooleanSolve` cited, but with more detailed complexity analysis.
For the probabilistic algorithm, instead, are reported just the Lokshtanov et al. algorithm, the very first introduced for this scope: its goal is to verify the consistency (i.e., determining whether or not the system has a solution) and computing a solution can be done iteratively several times. In the worst case, the algorithm solves a square ($m = n$) polynomial the time complexity is $\widetilde{\mathcal{O}}(q^{\delta n})$, for some $\delta < 1$ depending on q and $m = n$. Some variations of this algorithm have been made over time, such as Björklund's or Dinur's, but the complexity remains similar.

[5]The Gray code is an ordering of the binary numeral system such that two successive values differ in only one bit, as for example the numbers 1 and 2 - that in binary would normally be `001` and `010` - while in Gray code are represented as `001` and `011`.

[6]See A.5 for additional material.

Appendix A

Additional material

A.1 Dirac notation

In the field of quantum information, the fundamental unit is the *quantum bit* or *qubit*.The qubit is a vector in a complex vector space, and its two fundamental states, $|0\rangle$ are chosen to form an orthonormal basis of the vector space. The superposition of the states is possible by linear combination linear of the vectors of the basis. A generic vector in this space can be written in the form

$$|\psi\rangle = a|0\rangle + b|1\rangle, \qquad |a|^2 + |b|^2 = 1$$

The most commonly used notation is known as Bra-Ket or notation of Dirac, named after the scientist Paul Dirac who introduced it in his time. This notation makes it possible to describe vectors in a way that is more compact and easily handled from a computational point of view:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Let us then consider a generic vector written using the Dirac notation:

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = |\mathbf{v}\rangle$$

The column vector $|\mathbf{v}\rangle$ is referred to as *ket-v*; the dual vector of $\langle\mathbf{v}|$ is called *bra-v* and is written using the Dirac notation:

$$\langle\mathbf{v}| = \overline{\mathbf{v}^T} = [\overline{v_0} \quad \overline{v_1} \quad \dots \quad \overline{v_n}]$$

where $\overline{v}$ is the complex conjugate of v.
Dirac notation is a convenient way to describe vectors in Hilbert space H, which is the vector space used for the quantum computation. The inner product of two vectors in a Hilbert space is denoted using Dirac notation by $\langle \mathbf{u}|\mathbf{v}\rangle$ and is therefore calculated as the product of the vectors $\mathbf{v}$ and $\overline{\mathbf{u}^T}$ (which is the dual vector of $\mathbf{u}$):

$$\langle \mathbf{u}|\mathbf{v}\rangle = \overline{\mathbf{u}^T\mathbf{v}} = \begin{bmatrix} \overline{u_0} & \overline{u_1} & \dots & \overline{u_n} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = \overline{u_0} \cdot v_0 + \overline{u_1} \cdot v_1 + \dots + \overline{u_n} \cdot v_n$$

Two column vectors $|\mathbf{u}\rangle$ and $|\mathbf{v}\rangle$ of lengths m and n produce a column tensor vector of length $m \cdot n$:

$$|\mathbf{u}\rangle|\mathbf{v}\rangle = |\mathbf{uv}\rangle = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_m \end{bmatrix} \otimes \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_0 \cdot v_0 \\ u_0 \cdot v_1 \\ \vdots \\ u_0 \cdot v_n \\ u_1 \cdot v_0 \\ \vdots \\ u_{m-1} \cdot v_n \\ u_m \cdot v_0 \\ \vdots \\ u_m \cdot v_n \end{bmatrix}$$

Tensor products are important because they describe the interaction between two quantum systems. The vector space describing one quantum system multiplied with a tensor product with the vector space describing another system quantum system is the vector space consisting of linear combinations of all the vectors in the two vector spaces.

A.2 Group Homeomorphism

In topology, a *homeomorphism* (known as isomorphism or bi-continuous function too) is a bijective and continuous function between topological spaces that has a continuous inverse function. In particular, it's given as definition the following:

A function $f : X \to Y$ between two topological spaces is a homeomorphism if:

- $\diamond$ f is a bijective function;

- ◇ f is continuous;
- ◇ the inverse function f^{-1} is continuous.

Homeomorphisms are the isomorphisms in the category of topological spaces: for this reason, the composition of two homeomorphisms is again an homeomorphism, and the set of all self-homeomorphisms $X \to X$ forms the group called the homeomorphism group of X.

For our purpose, the two following functions are given as an example:

- ◦ The modular exponentiation $x \mapsto g^x \bmod p$;
  In fact if $s = r_0 + r_1$, then $g^s = (g^{r_0})(g^{r_1})$.
- ◦ The linear function $x \mapsto Mx$;
  In fact if $s = r_0 + r_1$, then $Ms = Mr_0 + Mr_1$.

A.3 Preimage attack

A preimage attack on a hash functions tries to find a message that has a specific hash value, in fact the hash function should resist attacks on its preimage (that is the set of possible inputs). There are two types of preimage resistance:

- ⋆ The **preimage resistance** tells that for all pre-specified outputs, it has to be computationally infeasible to find any input that hashes to that output;
  for example given y, it is difficult to find an x such that $h(x) = y$.

- ⋆ The **second-preimage resistance** tells instead that for a specified input, it has to be computationally infeasible to find another input which produces the same output;
  for example given x, it is difficult to find a second input $x' \neq x$ such that $h(x) = h(x')$.

The second-preimage resistance can be confronted with the **collision resistance**, for which it has to be computationally infeasible to find any two distinct inputs x, x' that hash to the same output (i.e. such that $h(x) = h(x')$. Note that the collision resistance implies the second-preimage resistance, but does not guarantee preimage resistance; in reverse, a second-preimage attack implies a collision attack.

A.4 Gröbner basis

Given a monomial order $<$ on the polynomial ring R, a subset G of the ideal $I \subset R$ is said to be a Gröbner basis for I with respect to $<$ if it satisfies one of the following properties:

- ∋ the ideal given by the principal terms of the polynomials in the ideal I is itself generated by the principal terms of the basis G;

- ∋ the principal term of each polynomial in I is divisible by the principal term of some polynomial in the base G;

- ∋ the multivariate division of each polynomial in the polynomial ring R by G returns a unique remainder;

- ∋ multivariate division of each polynomial in the ideal I by G returns 0.

Please note that all these properties are equivalent.

A.5 Macaulay matrix

The Macaulay matrix can be viewed as generalisation of the Sylvester Matrix, which is defined for two univariate polynomials:

Definition 31 *The Sylvester matrix associated to two polynomials of degree m and n is the $(n+m) \times (n+m)$ matrix constructed following given steps.*

Let p the and q two non-zero polynomials as $p(z) = p_0 + p_1 z + p_2 z^2 + \cdots + p_m z^m$ and $q(z) = q_0 + q_1 z + q_2 z^2 + \cdots + q_n z^n$, then the steps for the construction of the matrix are:

- if $n > 0$, the first row is $\begin{pmatrix} p_m & p_{m-1} & \cdots & p_1 & p_0 & 0 & \cdots & 0 \end{pmatrix}$;
- the second row is the first row, shifted one column to the right (the first element of the row is zero);
- the following $n-2$ rows are obtained the same way, shifting the coefficients one column to the right each time and setting the other entries in the row to be 0;
- if $m > 0$ the $(n+1)$-th row is $\begin{pmatrix} q_n & q_{n-1} & \cdots & q_1 & q_0 & 0 & \cdots & 0 \end{pmatrix}$;
- the following rows are obtained the same way as before.

Explicitly, if $n = 3$ and $m = 4$, the corresponding Sylvester matrix is The Maculay

$$\begin{pmatrix} p_4 & p_3 & p_2 & p_1 & p_0 & 0 & 0 \\ 0 & p_4 & p_3 & p_2 & p_1 & p_0 & 0 \\ 0 & 0 & p_4 & p_3 & p_2 & p_1 & p_0 \\ q_3 & q_2 & q_1 & q_0 & 0 & 0 & 0 \\ 0 & q_3 & q_2 & q_1 & q_0 & 0 & 0 \\ 0 & 0 & q_3 & q_2 & q_1 & q_0 & 0 \\ 0 & 0 & 0 & q_3 & q_2 & q_1 & q_0 \end{pmatrix}$$

Figure A.1: A Sylvester matrix

matrix is therefore the result of the introduction of multivariate resultants with the goal of finding an explicit solution to solve a systems of polynomial equations, by eliminating variables; the resultant of the system is the determinant of the matrix, obtained from the coefficients of the polynomials.

A.6 Finite fields

This section has been made to explain how calculations are developed in B, where arithmetic other than standard is used. Throughout the body of the thesis, every protocol and every calculation was done on a finite field, which is better analysed and explained here.

A field is defined as a commutative ring in which each of its non-zero elements possesses an inverse; this inverse is shown to be unique. A finite field is a field that contains a finite number of elements; such fields are completely classified:

- every finite field has p^n elements;
- For every prime number p and natural $n \geq 1$, there is only one finite field with p^n elements, barring isomorphism.

This is usually denoted by $\mathbb{F}_{p^n}$ or by $\mathrm{GF}(p^n)$, from Galois Field.
The finite field presents a different structure depending on whether n is 1, and thus the field has precisely p elements, or whether n is greater than 1.

$\mathbb{F}_{p^n}$ *with* $n = 1$

When the finite field has exactly p elements (so $n = 1$), its operations are defined by modular arithmetic modulo p. Hence $\mathbb{F}_p$ is the field of rest classes modulo p, and is also denoted by $\mathbb{Z}/p\mathbb{Z}$. The underlying group in this case is a cyclic group of order p.

$\mathbb{F}_{p^n}$ *with* $n > 1$

When $n > 1$, on the other hand, modular arithmetic modulo p does not yield a field since $\mathbb{F}_{p^n}$ is not isomorphic to the ring of rest classes $\mathbb{Z}/p^n\mathbb{Z}$ (we know that if p is not a prime, $\mathbb{Z}_p$ is not a field).
Given a prime power p^n with p prime and $n > 1$, the field $\mathrm{GF}(p^n)$ may be explicitly constructed choosing an irreducible polynomial P in $\mathrm{GF}(p)[X]$ of degree n (such an irreducible polynomial always exists). Then the quotient ring $\mathrm{GF}(q) = \mathrm{GF}(p)[X]/(P)$ of the polynomial ring $\mathrm{GF}(p)[X]$ by the ideal generated by P is a field of order q. More explicitly, the elements of $\mathrm{GF}(q)$ are the polynomials over $\mathrm{GF}(p)$ whose degree is strictly less than n.
In particular, the operations of the field are then defined by means of polynomial arithmetic and each element of the field is seen as a polynomial whose coefficients belong to $\mathbb{Z}/p\mathbb{Z}$ and whose maximum degree is $n-1$. The operations are performed as follows: the arithmetic on the coefficients is a modular arithmetic modulo p and at the end of each operation the resulting polynomial is divided by a particular

irreducible polynomial in $\mathbb{Z}_p$ of degree n and the remainder is taken. Please note that to verify the irreducibility of a polynomial, you can use the Berlekamp algorithm or the Rabin's irreducibility test.
It is well known that irreducible polynomials are not unique, which is why by convention Conway polynomials, those of the form $X^n + X^k + 1$ where k is the minimum possible, are used to perform such operations. They ensure a certain compatibility between the representation of a field and the representations of its sub-fields. In particular, Conway polynomials were defined with the purpose to provide a standard notation for elements in a finite field GF_{p^n} with p^n elements. Their are defined as follows:

Definition 32 *Let $g(X) = g_n X^n + \cdots + g_0$ and $h(X) = h_n X^n + \cdots + h_0$; then we define $g < h$ iff there is an index k with $g_i = h_i$ for $i > k$ and $(-1)^{n-k} g_k < (-1)^{n-k} h_k$. The Conway polynomial $f_{p,n}(X)$ for GF_{p^n} is the smallest polynomial of degree n with respect to this ordering such that:*

- *$f_{p,n}(X)$ is monic;*

- *$f_{p,n}(X)$ is primitive aka any zero is a generator of the (cyclic) multiplicative group of GF_{p^n};*

- *for each proper divisor m of n we have that $f_{p,m}(X^{\frac{p^n-1}{p^m-1}}) = 0 \ mod \ f_{p,n}(X)$ (that is, the $\frac{p^n-1}{p^m-1}$-th power of a zero of $f_{p,n}(X)$ is a zero of $f_{p,m}(X)$)*

The existence of these polynomials can be shown with the Chinese Remainder Theorem.
Since it is expensive to calculate the Conway polynomials explicitly, there are tables in which they are tabulated as n and p vary.

$\mathbf{f_{2,1}} = X + 1$
$\mathbf{f_{2,2}} = X^2 + X + 1$
$\mathbf{f_{2,3}} = X^3 + X + 1$
$\mathbf{f_{2,4}} = X^4 + X + 1$
$\mathbf{f_{2,5}} = X^5 + X^2 + 1$
$\mathbf{f_{2,6}} = X^6 + X^4 + X^3 + X + 1$
$\mathbf{f_{2,7}} = X^7 + X + 1$
$\mathbf{f_{2,8}} = X^8 + X^4 + X^3 + X^2 + 1$
$\mathbf{f_{2,9}} = X^9 + X^4 + 1$
$\mathbf{f_{2,10}} = X^{10} + X^6 + X^5 + X^3 + X^2 + X + 1$
$\mathbf{f_{2,11}} = X^{11} + X^2 + 1$
$\mathbf{f_{2,12}} = X^{12} + X^7 + X^6 + X^5 + X^3 + X + 1$
$\mathbf{f_{2,13}} = X^{13} + X^4 + X^3 + X + 1$
$\mathbf{f_{2,14}} = X^{14} + X^7 + X^5 + X^3 + 1$
$\mathbf{f_{2,15}} = X^{15} + X^5 + X^4 + X^2 + 1$
$\mathbf{f_{2,16}} = X^{16} + X^5 + X^3 + X^2 + 1$
$\mathbf{f_{2,17}} = X^{17} + X^3 + 1$
$\mathbf{f_{2,18}} = X^{18} + X^{12} + X^{10} + X + 1$
$\mathbf{f_{2,19}} = X^{19} + X^5 + X^2 + X + 1$
$\mathbf{f_{2,20}} = X^{20} + X^{10} + X^9 + X^7 + X^6 + X^5 + X^4 + X + 1$

Figure A.2: Table of the Conway polynomials with $p = 2$.

$\mathbf{f_{3,1}} = X + 1$
$\mathbf{f_{3,2}} = X^2 + 2X + 2$
$\mathbf{f_{3,3}} = X^3 + 2X + 1$
$\mathbf{f_{3,4}} = X^4 + 2X^3 + 2$
$\mathbf{f_{3,5}} = X^5 + 2X + 1$
$\mathbf{f_{3,6}} = X^6 + 2X^4 + X^2 + 2X + 2$
$\mathbf{f_{3,7}} = X^7 + 2X^2 + 1$
$\mathbf{f_{3,8}} = X^8 + 2X^5 + X^4 + 2X^2 + 2X + 2$
$\mathbf{f_{3,9}} = X^9 + 2X^3 + 2X^2 + X + 1$
$\mathbf{f_{3,10}} = X^{10} + 2X^6 + 2X^5 + 2X^4 + X + 2$
$\mathbf{f_{3,11}} = X^{11} + 2X^2 + 1$
$\mathbf{f_{3,12}} = X^{12} + X^6 + X^5 + X^4 + X^2 + 2$
$\mathbf{f_{3,13}} = X^{13} + 2X + 1$
$\mathbf{f_{3,14}} = X^{14} + 2X^9 + X^8 + X^7 + 2X^6 + X^5 + 2X^3 + X^2 + 2$
$\mathbf{f_{3,15}} = X^{15} + 2X^8 + X^5 + 2X^2 + X + 1$
$\mathbf{f_{3,16}} = X^{16} + 2X^7 + 2X^6 + 2X^4 + 2X^3 + 2X^2 + X + 2$
$\mathbf{f_{3,17}} = X^{17} + 2X + 1$
$\mathbf{f_{3,18}} = X^{18} + X^{10} + 2X^8 + 2X^6 + X^5 + 2X^4 + 2X^2 + 2$
$\mathbf{f_{3,19}} = X^{19} + 2X^2 + 1$
$\mathbf{f_{3,20}} = X^{20} + 2X^{13} + X^{11} + X^{10} + X^9 + X^8 + 2X^5 + 2X^4 + 2X^3 + X + 2$

Figure A.3: Table of the Conway polynomials with $p = 3$.

$\mathbf{f_{5,1}} = X + 3$
$\mathbf{f_{5,2}} = X^2 + 4\,X + 2$
$\mathbf{f_{5,3}} = X^3 + 3\,X + 3$
$\mathbf{f_{5,4}} = X^4 + 4\,X^2 + 4\,X + 2$
$\mathbf{f_{5,5}} = X^5 + 4\,X + 3$
$\mathbf{f_{5,6}} = X^6 + X^4 + 4\,X^3 + X^2 + 2$
$\mathbf{f_{5,7}} = X^7 + 3\,X + 3$
$\mathbf{f_{5,8}} = X^8 + X^4 + 3\,X^2 + 4\,X + 2$
$\mathbf{f_{5,9}} = X^9 + 2\,X^3 + X + 3$
$\mathbf{f_{5,10}} = X^{10} + 3\,X^5 + 3\,X^4 + 2\,X^3 + 4\,X^2 + X + 2$
$\mathbf{f_{5,11}} = X^{11} + 3\,X + 3$
$\mathbf{f_{5,12}} = X^{12} + X^7 + X^6 + 4\,X^4 + 4\,X^3 + 3\,X^2 + 2\,X + 2$
$\mathbf{f_{5,13}} = X^{13} + 4\,X^2 + 3\,X + 3$
$\mathbf{f_{5,14}} = X^{14} + X^8 + 4\,X^6 + 4\,X^5 + 2\,X^4 + 3\,X^3 + X + 2$
$\mathbf{f_{5,15}} = X^{15} + 2\,X^5 + 3\,X^3 + 3\,X^2 + 4\,X + 3$
$\mathbf{f_{5,16}} = X^{16} + X^8 + 4\,X^7 + 4\,X^6 + 4\,X^5 + 2\,X^4 + 4\,X^3 + 4\,X^2 + X + 2$
$\mathbf{f_{5,17}} = X^{17} + 3\,X^2 + 2\,X + 3$
$\mathbf{f_{5,18}} = X^{18} + X^{12} + X^{11} + X^{10} + X^9 + 2\,X^8 + 2\,X^6 + X^5 + 2\,X^3 + 2\,X^2 + 2$
$\mathbf{f_{5,19}} = X^{19} + X^3 + 2\,X + 3$
$\mathbf{f_{5,20}} = X^{20} + 3\,X^{12} + 4\,X^{10} + 3\,X^9 + 2\,X^8 + 3\,X^6 + 4\,X^3 + X + 2$

Figure A.4: Table of the Conway polynomials with $p = 5$.

For the sake of explanation, here is an example in which this is explicitly calculated: let's verify that, in GF_{2^5}, $7 \cdot 15 = 8$.
$7 = x^2 + x + 1$
$15 = x^3 + x^2 + x + 1$
$f_{2,5} = x^5 + x^2 + 1$
$(x^2 + x + 1) \cdot (x^3 + x^2 + x + 1) = x^5 + 2x^4 + 3x^3 + 3x^2 + 2x + 1 = x^5 + x^3 + x^2 + 1$
$(x^5 + x^3 + x^2 + 1)/(x^5 + x^2 + 1) = 1 \quad r = x^3$
In fact, $8 = x^3$.
Please note that also $x^5 + x^3 + 1$ is an irreducible polynomial in GF_{2^5} (demonstrable with Berlekamp), but we used the Conway one, with the minor total degree.

Appendix B

Code

B.1 Chen

Here are given the codes written in Python that implement, in a simplistic but functional manner, the key generation - KeyGen(), the signing scheme and the verification scheme - MQDSS(sk, pk, A, b, msg) and Verif(σ, msg, pk) - based on the Chen's Identification Scheme.
Please note the following implementations are valid only for q prime of the field $\mathbb{F}_q$, since is exploited the modular arithmetic.

KeyGen()

```
#KeyGen()

import random
import numpy as np

paramSec = 128
n = 10
m = n+1
q = 13 #Only a prime number
seed_pk = random.getrandbits(paramSec)
seed_sk = random.getrandbits(paramSec)
seed_pk = seed_pk % (2**32 - 1)
seed_sk = seed_sk % (2**32 - 1)

np.random.seed(seed_sk)
sk = np.random.randint(q, size=(n, 1))

np.random.seed(seed_pk)
pk1 = np.zeros((m, 1))
A = np.zeros((m, n, n))
b = np.zeros((m, n, 1))

for i in range(m):
    A[i] = np.random.randint(q, size=(n, n))
    b[i] = np.floor(q * np.random.rand(n, 1))
    pk1[i] = sk.T @ A[i] @ sk + b[i].T @ sk

pk = pk1 % q

np.savez("chiavi.npz", sk=sk, pk=pk, n=n, m=m, q=q,
         A=A, b=b, paramSec=paramSec)
```

```
print(sk.T)
```

```
array([[ 4, 12,  9,  7,  2,  2,  0, 11,  7,  9]])
```

```
print(pk.T)
```

```
array([[ 4,  2,  7,  9,  1,  3, 10,  9,  3, 10,  5]])
```

```
print(A[1])
```

```
array([[ 5,  9,  0,  3,  1,  1,  2,  4,  5,  7],
       [ 5,  8,  2, 11,  0,  8,  5,  7,  1,  2],
       [10, 12,  4,  9, 11, 11,  1,  0,  2, 11],
       [ 1,  5, 12,  6,  8,  4,  8, 10,  0,  9],
       [12,  9,  3, 10,  1,  5, 11,  3,  3,  1],
       [ 0,  8,  2,  3,  6,  0,  7,  1,  7,  9],
       [ 1,  5,  9,  5,  0,  6,  8,  6, 12,  7],
       [11,  1,  1, 10,  5,  2, 12,  4,  9,  9],
       [11,  8,  9,  4,  3, 11,  5, 12,  8,  8],
       [ 1, 12,  5,  7,  5,  3, 11,  7,  0,  8]])
```

```
print(b[:,1])
```

```
array([ 4,  6, 11,  4, 11,  8,  0, 12,  8, 12])
```

Figure B.1: Example of algorithm KeyGen() output, with $n = 10$, $q = 13$

MQDSS(sk, pk, A, b, msg)

```
#MQDSS(sk, pk, A, b, msg)

import numpy as np
import hashlib
from hashlib import shake_128
data = np.load("chiavi.npz")

sk = data['sk']
pk = data['pk']
q = int(data['q'])
n = int(data['n'])
m = int(data['m'])
A = data['A']
b = data['b']
s = sk
v = pk
d=20
M_esteso = "Fiat--Shamir transform"  # Message

tau = 3
sigma0 = []
sigma2_r0 = []
sigma2_r1 = []
vett_t0 = []
vett_e0 = []
F_r0 = []
for t in range(tau):
    np.random.seed(t)
    r0 = np.random.randint(q, size=(n, 1))
    np.random.seed(t+1)
    t0 = np.random.randint(q, size=(n, 1))
    np.random.seed(t+2)
    e0 = np.random.randint(q, size=(m, 1))

    r1 = ((s - r0) % q).astype('int')

    concatenazione = np.concatenate((r0, t0, e0),
                                    axis=0)
    concatenazione_stringa = ''
    for i in range(len(concatenazione.T[0])):
        concatenazione_stringa += str(concatenazione.T[0][i])
    arg = bytes(concatenazione_stringa,'utf-8')
    c0 = hashlib.sha3_256(arg).hexdigest()

    sigma0.append(c0)
    sigma2_r0.append(r0)
    sigma2_r1.append(r1)
    vett_t0.append(t0)
```

```
49      vett_e0.append(e0)
50
51      G_t0_r1 = np.zeros((m, 1), dtype=int)
52      Fr0 = np.zeros((m, 1), dtype=int) #Used in First Response
53      for i in range(m):
54          G_t0_r1[i] = (t0.T @ A[i] @ r1 + r1.T @ A[i] @ t0)%q
55          Fr0[i] = (r0.T @ A[i] @ r0 + b[i].T @ r0 ) % q
56
57      F_r0.append(Fr0)
58
59      concatenazione2 = np.concatenate((r1,
60                       np.mod(G_t0_r1 + e0, q)), axis=0)
61      concatenazione_stringa2 = ''
62      for i in range(len(concatenazione2.T[0])):
63          concatenazione_stringa2 += str(concatenazione2.T[0][i])
64      arg2 = bytes(concatenazione_stringa2,'utf-8')
65      c1 = hashlib.sha3_256(arg2).hexdigest()
66
67      sigma0.append(c1)
68
69
70  #First challenge
71  strSigma0=''
72  for i in range(len(sigma0)):
73      strSigma0 += sigma0[i]
74
75  h1 = int(shake_128((M_esteso + strSigma0)
76                     .encode()).hexdigest(d), 16) % q
77
78
79  #First response
80  sigma1_hash = []
81  sigma1 = np.zeros((m,2*tau), dtype='int')
82  for t in range(tau):
83      t1 = np.mod(h1 * sigma2_r0[t] - vett_t0[t], q)
84      e1 = np.mod(h1 * F_r0[t] - vett_e0[t], q)
85      #Conversione per fare hash
86      t1_0=t1.flatten().astype(int).tolist()
87      t1_1 = ''.join(map(str, t1_0))
88      sigma1_hash.append(t1_1)
89
90      e1_0=e1.flatten().astype(int).tolist()
91      e1_1 = ''.join(map(str, e1_0))
92      sigma1_hash.append(e1_1)
93
94      for i in range(n):
95          sigma1[i,2*t] = t1[i].item()
96      for j in range(len(e1)):
97          sigma1[j,2*t+1] = e1[j]
```

```
98
99  strSigma1=''
100 for i in range(len(sigma1_hash)):
101     strSigma1 += sigma1_hash[i]
102
103
104 #Second challenge
105 h2 = int(shake_128((M_esteso + strSigma0 + str(h1)
106         + strSigma1).encode()).hexdigest(d), 16) % 2
107
108 #Second response
109 if h2 == 0:
110     sigma2 = sigma2_r0
111 else:
112     sigma2 = sigma2_r1
113
114 np.savez("pubblico.npz", sigma0=sigma0, sigma1=sigma1,
115          sigma2=sigma2, v=v, A=A, b=b, M_esteso=M_esteso,
116          d=d, q=q, m=m, n=n, tau=tau)
```

Table B.1: This implementation of the Signature Scheme algorithm follows the description made in the pseudo-code in table 4.1

```
sigma0
```

```
array(['72dbd46c5c4782dec5661e72166195702d4b6454996d8a11c559eed34c97b6a4',
       'c300ba471bcc9f302b6e7ace230dcf0efc7a06e41fd056ccb36dca5adfb4ad30'],
      dtype='<U64')
```

```
e1
```

```
array([[ 3,  6,  0,  4,  0,  0, 12,  0,  3,  9,  4]], dtype=int32)
```

```
sigma2.T
```

```
array([[ 2, 11,  2, 11,  6,  7,  9,  6,  3,  2]])
```

Figure B.2: Example of algorithm MQDSS(sk, pk, A, b, msg) output

Please note that in line 55 I've used the following properties:

$G(x, y) = F(x, y) - F(x) - F(y) =^1$

$(x + y)^T A(x + y) + b^T(x + y) - x^T Ax - b^T x - y^T Ay - b^T y = y^T Ax + x^T Ay.$

The same will occur in the next code, in line 98.

[1]Knowing $F(x) = x^T Ax + b^T x$

Verif(σ, msg, pk)

```
#Verif_MQDSS(sigma, msg, pk)

import hashlib
import numpy as np
from hashlib import shake_128

# Load data of the file
data = np.load("pubblico.npz")
sigma0 = data['sigma0']
sigma1 = data['sigma1']
sigma2 = data['sigma2']
v = data['v']
A = data['A']
b = data['b']
M_esteso = data['M_esteso'].item()
q = int(data['q'])
n = int(data['n'])
m = int(data['m'])
d = int(data['d'])
tau = int(data['tau'])


#Find h1
strSigma0=''
for i in range(len(sigma0)):
    strSigma0 += sigma0[i]

h1_v = int(shake_128((M_esteso + strSigma0)
                .encode()).hexdigest(d), 16) % q

#Recalculate the values of the vectors t1 and e1
sigma1_t1 = np.zeros((n,tau), dtype='int')
sigma1_e1 = np.zeros((m,tau), dtype='int')

for i in range(tau):
    for j in range(n):
        sigma1_t1[j,i] = sigma1[j,2*i]

for i in range(tau):
    sigma1_e1[:,i] = sigma1[:,2*i+1]


#Find h2
sigma1_per_str=[]
for i in range(tau):

    t1_0=sigma1_t1[:,i].flatten().astype(int).tolist()
    t1_1 = ''.join(map(str, t1_0))
```

```
    sigma1_per_str.append(t1_1)

    e1_0=sigma1_e1[:,i].flatten().astype(int).tolist()
    e1_1 = ''.join(map(str, e1_0))
    sigma1_per_str.append(e1_1)

strSigma1=''
for i in range(len(sigma1_per_str)):
    strSigma1 += sigma1_per_str[i]

h2_v = int(shake_128((M_esteso + strSigma0 + str(h1_v) + strSigma1
   ).encode()).hexdigest(d), 16) % 2


c0 = []
c1 = []
if h2_v == 0:
    for t in range(tau):
        Fr0_1 = np.zeros(m, dtype=int)
        for i in range(m):
            Fr0_1[i] = (sigma2[t].T @ A[i] @ sigma2[t]
                        + b[i].T @ sigma2[t]) % q

        conc = np.concatenate((sigma2[t],
                np.mod(h1_v * sigma2[t]-sigma1_t1[:,t].reshape(1,
   -1).T, q),
                       np.mod(h1_v * Fr0_1 - sigma1_e1[:,t], q).
   reshape(1, -1).T),
                       axis=0)

        stringa_conc = ''
        for i in range(len(conc.T[0])):
            stringa_conc += str(conc.T[0][i])
        arg = bytes(stringa_conc,'utf-8')

        c0_hash = hashlib.sha3_256(arg).hexdigest()
        c0.append(c0_hash)

    if sigma0[2*t] == c0[t]:
        bit = True
    else:
        bit = False

else: #h2=1
    for t in range(tau):
        G_t1_r1 = np.zeros(m, dtype=int)
        Fr1 = np.zeros(m, dtype=int)
        for i in range(m):
```

```
94              G_t1_r1[i] = (sigma1_t1[:,t].reshape(1, -1) @ A[i] @
    sigma2[t]
95                          + sigma2[t].T @ A[i] @ sigma1_t1[:,t].
    reshape(1, -1).T) % q
96
97              Fr1[i] = (sigma2[t].T @ A[i] @ sigma2[t] + b[i].T @
    sigma2[t] ) % q
98
99          conc2 = np.concatenate((sigma2[t],
100                 np.mod(h1_v * (v.T - Fr1) - G_t1_r1 - sigma1_e1[:,
    t], q)
101                 .astype('int').reshape(-1, 1)),axis=0)
102
103
104         stringa_conc_2 = ''
105         for i in range(len(conc2[0])):
106             stringa_conc_2 += str(conc2[0][i])
107         arg2 = bytes(stringa_conc_2,'utf-8')
108
109         c1_hash = hashlib.sha3_256(arg2).hexdigest()
110         c1.append(c1_hash)
111
112     if sigma0[2*t+1] == c1[t]:
113         bit = True
114     else:
115         bit = False
116
117 print(bit)
```

Table B.2: This implementation of the Verification algorithm follows the description made in the pseudo-code in table 4.2

B.2 Feneuil

Here are given the codes written in Python that implement, in a simplistic but functional manner, the key generation - KeyGen(), the signing scheme and the verification scheme - DSS(sk, pk, A, b, msg) and Verif(σ, msg, pk) - based on the Feneuil's protocol.
Please note that first are given the implementations valid only for q prime of the field $\mathbb{F}_q$, since is exploited the modular arithmetic, then are given the codes valid for every q, prime and power of a prime (in this case is exploited the polynomial arithmetic, which is why the Pyfinite library is used).

KeyGen()

```
import random
import numpy as np

#Faccio i seed con il modulo random
paramSec = 256
n = 9
m = n+1
q = 23 #solo un numero primo
seed_pk = random.getrandbits(paramSec) % (2**32 - 1)
seed_sk = random.getrandbits(paramSec) % (2**32 - 1)

#Campiono col modulo numpy
np.random.seed(seed_sk)
sk = np.random.randint(q, size=(n, 1))

np.random.seed(seed_pk)
pk1 = np.zeros((m, 1))
A = np.zeros((m, n, n))
b = np.zeros((n, m))

for i in range(m):
    A[i] = np.random.randint(q, size=(n, n))
    b[:, i] = np.random.randint(q, size=n)

    pk1[i]=np.dot(sk.T, np.dot(A[i], sk))
    + np.dot(b[:, i].T, sk)

pk = pk1 % q

np.savez("chiavi.npz", sk=sk, pk=pk, n=n, m=m,
         q=q, paramSec=paramSec, A=A, b=b)
```

Table B.3: This implementation of the Key Generation algorithm follows the description made in the pseudo-code in table 4.3

```
print(sk.T)
```

```
array([[12, 19, 10,  4, 17,  8, 10,  8,  3]])
```

```
print(pk.T)
```

```
array([[19,  7,  0,  4,  9, 21,  5, 17, 22, 21]])
```

```
print(A[4])
```

```
array([[20, 12,  7, 18, 11,  5, 22, 13,  0],
       [ 3, 21, 12,  7,  7, 13, 10, 16, 18],
       [ 9, 15, 12,  3,  9, 19, 19, 11, 22],
       [ 4,  6, 12, 15,  3,  5,  4, 20, 19],
       [ 4,  9, 17, 21,  5, 12,  9,  1, 14],
       [ 9, 18,  8, 17, 10, 13,  8, 12,  0],
       [ 0,  8, 10,  4, 12, 11, 11,  3, 18],
       [13, 18,  9, 22,  6,  8, 21,  9, 10],
       [ 3,  7,  6,  8,  3, 16,  6, 22,  4]])
```

```
print(b[:,4])
```

```
array([ 2,  0,  5,  3, 17, 22,  1,  6, 17])
```

Figure B.3: Example of algorithm KeyGen() output, with $n = 9$, $m = n + 1$, $q = 23$.

Sign()

```
import numpy as np
import random
import hashlib
data = np.load("chiavi.npz")

paramSec = data['paramSec']
sk = data['sk']
pk = data['pk']
q = int(data['q'])
n = int(data['n'])
m = int(data['m'])
A = data['A']
b = data['b']
numParties = 7
tau = 269
eta = 1
msg = "Fiat--Shamir transform"  # messaggio
random.seed(0)
salt = random.getrandbits(2*paramSec)  % (2**32 - 1)

#Fase1
com=[]
a = np.zeros((tau, n, numParties), dtype='int')
x = np.zeros((tau, n, numParties), dtype='int')
c = np.zeros((tau, 1, numParties), dtype='int')
state = np.zeros((tau, numParties+n+1), dtype='int')

for t in range(tau):
    random.seed(t+1)
    seed1_t = (random.getrandbits(paramSec)) % (2**32 - 1)
    np.random.seed([salt, seed1_t])
    seed2_t = np.random.randint(seed1_t, size=numParties)
    for i in range(numParties):
        state[t,i] = seed2_t[i]

    for j in range(numParties):
        np.random.seed([salt,seed2_t[j]])
        x[t,:,j]=np.random.randint(q,size=n)
        a[t,:,j]=np.random.randint(q**eta,size=n)
        c[t,0,j]=np.random.randint(q**eta)
    x[t,:,numParties-1]=(sk.T-np.sum(x[t,:,:-1], axis=1)) % q

    a_tot = np.sum(a[t], axis=1) %q**eta
    c_tot= (- np.dot(a_tot,sk)) %q**eta
    c[t,:,numParties-1] = (c_tot - np.sum(c[t,:,:-1])) %q**eta

    for i in range(numParties,numParties+n):
        state[t,i] = x[t,i-numParties,numParties-1].T
```

```
49
50      state[t,numParties+n] = c[t,:,numParties-1].item()
51
52      for i in range(numParties-1):
53          res = np.concatenate([np.array([salt]),
54                np.array([t]), np.array([i+1]),
55                np.array([state[t,i]])])
56          com.append(hashlib.sha3_256(res.tobytes()).hexdigest())
57      #com_N
58      res = np.concatenate([np.array([salt]), np.array([t]),
59            np.array([numParties]), state[t,numParties-1:]])
60      com.append(hashlib.sha3_256(res.tobytes()).hexdigest())
61
62  #Fase2
63  comRes=''
64  for i in range(numParties*tau):
65      comRes+= com[i]
66  stringa=msg + str(salt) + comRes
67  o=bytes(stringa,'utf-8')
68  h1 = hashlib.shake_128(o).hexdigest(4)
69
70  np.random.seed(int(h1,16))
71  gamma = np.zeros((tau,1,m), dtype='int')
72  vareps = np.zeros((1,tau), dtype='int')
73
74  for t in range(tau):
75      gamma[t] = np.random.randint(q**eta, size=m)
76      vareps[:,t] = np.random.randint(q**eta)
77
78  #Fase3
79  z=np.zeros((tau, 1,numParties), dtype='int')
80  w=np.zeros((tau, n,numParties),dtype='int')
81  alpha1 = np.zeros((tau, n, numParties), dtype='int')
82  openAlpha1 = np.zeros((tau, n, 1), dtype='int')
83  v1 = np.zeros((tau, 1, numParties), dtype='int')
84
85  for t in range(tau):
86
87      #Calcolo shares di z
88      a2=np.zeros(m,dtype='int')
89      for i in range(m):
90          a1=(pk[i] - np.inner(b[:,i].T,x[t,:,0]))%q
91          a2[i]=(gamma[t,0,i]*a1.item())%q
92      z[t,0,0]= (np.sum(a2)%q)
93
94      for j in range(1,numParties):
95          a1=np.zeros(m,dtype='int')
96          for i in range(m):
97              a1[i] = (- gamma[t,0,i]*(np.inner(b[:,i].T,
```

```
                                                x[t,:,j])))%q
        z[t,0,j]= (np.sum(a1)%q)

    #Calcolo shares di w
    B=np.zeros((m,n,n),dtype='int')
    for i in range(m):
        B[i]=gamma[t,0,i]*A[i]
    C=(np.sum(B, axis=0)%q)

    for j in range(numParties):
        w[t,:,j]=(np.matmul(C,x[t,:,j])%q)

    #Calcolo alpha
    for j in range(numParties):
        alpha1[t,:,j] = vareps[0,t]*w[t,:,j] + a[t,:,j]
    alpha = alpha1 % q**eta

    for j in range(numParties):
        openAlpha1[t,:,0] += alpha[t,:,j]
    openAlpha = openAlpha1 % q**eta

    #Calcolo v
    for j in range(numParties):
        v1[t,0,j]= vareps[0,t]*z[t,0,j]
        - np.inner(openAlpha[t,:, 0], x[t,:,j]) - c[t,0,j]
    v = v1%q**eta

#Fase4
coppiaRes=[]
for t in range(tau):
    for i in range(numParties):
        res=np.concatenate([alpha[t,:,i],np.array([v[t,0,i]])])
        coppiaRes += res.tolist()
hash0 = ''
for el in coppiaRes:
    hash0 += str(el)

stringa2 = msg + str(salt) + h1 + hash0
o2 = bytes(stringa2, 'utf-8')
h2 = hashlib.shake_128(o2).hexdigest(4)

np.random.seed(int(h2,16))
istar = np.zeros((1,tau), dtype='int')

for t in range(tau):
    istar[:,t] = np.random.randint(1,numParties+1)


#Fase5
```

```
147 stateN = np.zeros((tau, numParties+n), dtype='int')
148 for t in range(tau):
149     if istar[0,t] != numParties:
150         stateN[t] = state[t, [jdx
151             for jdx in range(state.shape[1])
152             if (jdx != (istar[0,t]-1))]]
153     else:
154         for i in range(numParties-1):
155             stateN[t,i] = state[t,i]
156 sigma = []
157 sigma.append(salt)
158 sigma.append(h1)
159 sigma.append(h2)
160 for t in range(tau):
161     sigma.append(stateN[t])
162     sigma.append(com[numParties*t+istar[0,t]-1])
163     sigma.append(alpha[t,:,istar[0,t]-1])
164
165 import pickle
166 with open('pubblico.pkl', 'wb') as file:
167     pickle.dump({'sigma': sigma, 'pk': pk, 'A':A, 'b':b,
168                  'msg':msg, 'numParties':numParties,
169                  'tau':tau, 'q':q, 'n':n, 'm':m, 'eta':eta},
170                  file)
```

Table B.4: This implementation of the Digital Signature Scheme follows the description made in the pseudo-code in table 4.4

```
x[0:2]
```

```
array([[[ 3,  4,  3,  7, 18,  3, 10],
        [19,  1, 19,  8, 11, 19,  6],
        [17, 12, 18,  9, 14, 17,  8],
        [19, 12, 12, 12, 10, 19, 13],
        [17,  4,  0,  6,  0, 17,  9],
        [18, 16, 20,  4,  5, 18,  7],
        [16,  3,  3, 15,  0, 16,  8],
        [14, 10, 20, 16, 10, 14, 20],
        [13, 12,  6,  3, 10, 13, 12]],

       [[11,  4, 11, 17,  3,  2,  0],
        [13,  1, 13,  3,  2,  0,  5],
        [ 0,  9,  0, 16,  5,  7, 12],
        [ 9,  1,  9, 20, 21, 14,  0],
        [17,  5, 17, 10, 10, 18, 22],
        [22,  3, 22,  9, 12,  5, 15],
        [17,  2, 17,  7,  3, 19, 19],
        [11,  0, 11,  7, 15, 21, 16],
        [ 8, 22,  8,  3, 17,  8,  3]]])
```

```
state
```

```
array([[ 1,  3,  0,  4,  2,  1,  7, 10,  6,  8, 13,  9,  7,  8, 20, 12,
         5],
       [12, 26, 12,  8, 41, 46,  9,  0,  5, 12,  0, 22, 15, 19, 16,  3,
        20],
       [14,  6,  6,  1,  2,  7,  6,  8, 18,  8, 22,  5,  2,  7, 13,  3,
        16],
       [14,  6,  6,  1,  2,  7,  6,  8, 18,  8, 22,  5,  2,  7, 13,  3,
        16],
       [ 5,  3,  6, 38, 37, 30, 36, 17, 17,  2,  2, 14,  8, 15,  8,  9,
         9],
       [ 8, 31,  7, 27, 27,  1,  5, 15, 13, 12,  3,  7,  9,  6, 12, 20,
        10],
       [ 2, 15,  8, 18,  4, 16, 12, 19, 20,  3,  4,  1, 15, 20, 14, 12,
        11],
       [ 5, 12,  4,  1,  5,  0,  0,  0, 17, 18, 22,  1, 17, 15,  5,  1,
         0]])
```

```
com[0:15]
```

```
['998d243de0517573a838314138daa8321b6241a9883b6aba96809d7f8e573c78',
 'e79743edd16a8652db12630bf7a95196d2ff8eedc53f08c5520d322d51ab071b',
 '19f0d9e8eb267014544a4438c0f7d601e404b6e1dfb7dfaa51939c57c0e440e4',
 'e6b0c0e7a6c4d681c84653978be28b0cc0e7f177a722076bdfce061a2f0ba79d',
 '9c58d4b48e16c87649a93a25fc259003b9d053dc874b063d2b58090666cd6b13',
 'e1f6b98d5186124ae81fe638f869c9f9e85f2ec2897e5d11d9cfb84b2fbe31fa',
 '0024fe3d51d4cc892e5f80a229e56b9e0b95dc489a378ea8cd8bb481bbae5b34',
 '81ed2c0e0a86ab2e2ea85f3f1d8cc1490d4c72be810dd8a5e4f6c3a8acccdb6f',
 '9e8f85fc51cc21c2ac0f6d5edd5cdbd5254474d31c31956fb7dfd9594f3884f3',
 'd02bfbf6608804e318b0c6b0f4082569c70fdc5936db25cccca1f0caf79390d3',
 '33aa8f2ecc0b8762a11d04d88240b69adcde5f1a9a34a4299781020e5790b543',
 '218a35987522e7ef87ac8c6f2bb109c45fa346ad82cc1237717e663b4407d4a3',
 '1b873c760ac2053403eae9f3abc73ec7ed42f1165a64e5e206b0b67286a2390f',
 '3a79f1a5f5e24f157f91a897dd4f4c4cc633c17653c3f4c1a0367dbe457406d4',
 '0c5b4c683d2dac7b4823508213352585fa1dd895a018af50b8cede5ec74343b3']
```

```
h1
```

```
'0c31f476'
```

```
h2
```

```
'244f3775'
```

```
istar
```

```
array([[5, 3, 5, 2, 3, 2, 6, 1]])
```

```
v
```

```
array([[[ 6, 15, 19,  2,  8,  6, 13]],

       [[ 5, 18, 11, 11,  8, 19, 20]],

       [[ 4, 10, 10, 22,  6, 12,  5]],

       [[18, 22, 22, 12,  0, 20, 21]],

       [[ 9, 15, 16, 12, 18,  6, 16]],

       [[ 1,  3,  0, 16, 16, 10,  0]],

       [[ 3,  0,  8,  8,  9,  5, 13]],

       [[ 4, 17,  5, 13, 21, 17, 15]]], dtype=int32)
```

Figure B.4: Example of algorithm Sign() output, with $n = 9$, $m = n + 1$, $q = 23$, number of rounds $\tau = 8$.

Verif()

```
import pickle
import numpy as np
import hashlib
with open('pubblico.pkl', 'rb') as file:
    data = pickle.load(file)
sigma = data['sigma']
pk = data['pk']
A = data['A']
b = data['b']
msg = data['msg'] #"Fiat-Shamir trasform"
numParties = data['numParties']
tau = data['tau']
q = data['q']
n = data['n']
m = data['m']
eta = data['eta']
salt = sigma[0]
h1 = sigma[1]
h2 = sigma[2]

stateN = np.zeros((tau, numParties+n), dtype='int')
com_istar=[]
alpha_istar = np.zeros((tau, n), dtype='int')
for i in range(1,tau+1):
    stateN[i-1] = sigma[3*i] #tutti tranne istar
    com_istar.append(sigma[3*i +1])
    alpha_istar[i-1] = sigma[3*i+2]

np.random.seed(int(h1,16))
gamma = np.zeros((tau,1,m), dtype='int')
vareps = np.zeros((1,tau), dtype='int')

for t in range(tau):
    gamma[t] = np.random.randint(q**eta, size=m)
    vareps[:,t] = np.random.randint(q**eta)

np.random.seed(int(h2,16))
istar = np.zeros((1,tau), dtype='int')
for t in range(tau):
    istar[:,t] = np.random.randint(1,numParties+1)

#Ricostruisco gli stati per fare i com
state_rebuilt = np.zeros((tau, numParties+n+1), dtype='int')
for t in range(tau):
    if istar[0,t] == numParties:
        for i in range(numParties+n):
            state_rebuilt[t,i] = stateN[t,i]
    else:
```

```
        for i in range(istar[0,t]-1):
            state_rebuilt[t,i] = stateN[t,i]
        for i in range(istar[0,t], numParties+n+1):
            state_rebuilt[t,i] = stateN[t,i-1]

com=[]
for t in range(tau):

    for i in range(0,istar[0,t]-1):
        res = np.concatenate([np.array([salt]),
              np.array([t]), np.array([i+1]),
              np.array([state_rebuilt[t,i]])])
        com.append(hashlib.sha3_256(res.tobytes()).hexdigest())

    com.append(com_istar[t])

    for i in range(istar[0,t], numParties-1):
        res = np.concatenate([np.array([salt]),
              np.array([t]), np.array([i+1]),
              np.array([state_rebuilt[t,i]])])
        com.append(hashlib.sha3_256(res.tobytes()).hexdigest())

    if istar[0,t]!= numParties:
        #com_numParties lo devo trattare a parte
        res = np.concatenate([np.array([salt]),
              np.array([t]), np.array([numParties]),
              state_rebuilt[t,numParties-1:]])
        com.append(hashlib.sha3_256(res.tobytes()).hexdigest())

comRes=''
for i in range(numParties*tau):
    comRes+= com[i]
stringa=msg + str(salt) + comRes
o=bytes(stringa,'utf-8')
h1_1 = hashlib.shake_128(o).hexdigest(4)

seed2 = np.zeros((tau,numParties), dtype='int')

for t in range(tau):
    if istar[0,t] != numParties:
        for i in range(0,istar[0,t]-1):
            seed2[t,i] = stateN[t,i]

        for i in range(istar[0,t],numParties):
            seed2[t,i] = stateN[t,i-1]
    else:
        for i in range(numParties-1):
            seed2[t,i] = stateN[t,i]

```

```
x = np.zeros((tau,n,numParties), dtype='int')
a = np.zeros((tau,n, numParties), dtype='int')
c = np.zeros((tau,1, numParties), dtype='int')
for t in range(tau):
    for j in range(numParties):
        np.random.seed([salt,seed2[t,j]])
        x[t,:,j]=np.random.randint(q,size=n)
        a[t,:,j]=np.random.randint(q,size=n)
        c[t,0,j]=np.random.randint(q)
    if istar[0,t] != numParties:
        x[t,:,numParties-1] = stateN[t,
                              numParties-1:len(stateN[t])-1]
        c[t,:,numParties-1] = stateN[t,len(stateN[t])-1]

z=np.zeros((tau, 1,numParties), dtype='int')
w=np.zeros((tau, n,numParties),dtype='int')
alpha1 = np.zeros((tau, n, numParties), dtype='int')
openAlpha1 = np.zeros((tau, n, 1), dtype='int')
v1 = np.zeros((tau, 1, numParties), dtype='int')

for t in range(tau):
    #Calcolo shares di z
    a2=np.zeros(m,dtype='int')
    for i in range(m):
        a1=(pk[i] - np.inner(b[:,i].T,x[t,:,0]))%q
        a2[i]=(gamma[t,0,i]*a1.item())%q
    z[t,0,0]= (np.sum(a2)%q)

    for j in range(1,numParties):
        a1=np.zeros(m,dtype='int')
        for i in range(m):
            a1[i] = (- gamma[t,0,i]*(np.inner(b[:,i].T,
                                              x[t,:,j])))%q
        z[t,0,j]= (np.sum(a1)%q)


    #Calcolo shares di w
    B=np.zeros((m,n,n),dtype='int')
    for i in range(m):
        B[i]=gamma[t,0,i]*A[i]
    C=(np.sum(B, axis=0)%q)

    for j in range(numParties):
        w[t,:,j]=(np.matmul(C,x[t,:,j])%q)
        alpha1[t,:,j] = vareps[0,t]*w[t,:,j] + a[t,:,j]
    alpha1[t,:,istar[0,t]-1] = alpha_istar[t,:]
    alpha = alpha1 % q**eta

    for j in range(numParties):
```

```
147         openAlpha1[t,:,0] += alpha[t,:,j]
148     openAlpha = openAlpha1 % q**eta
149 
150     #Calcolo v
151     for j in range(numParties):
152         v1[t,0,j]= vareps[0,t]*z[t,0,j]
153         - np.inner(openAlpha[t,:, 0], x[t,:,j]) - c[t,0,j]
154     temp = v1[t,0,istar[0,t]-1]%q
155     res = np.sum(v1[t]) %q
156     v1[t,0,istar[0,t]-1] = -(res-temp) %q
157 
158     v = v1%q**eta
159 
160 
161 coppiaRes=[]
162 for t in range(tau):
163     for i in range(numParties):
164         res=np.concatenate([alpha[t,:,i], np.array([v[t,0,i]])])
165         coppiaRes += res.tolist()
166 hashn = ''
167 for el in coppiaRes:
168     hashn += str(el)
169 
170 stringa2 = msg + str(salt) + h1_1 + hashn
171 o2 = bytes(stringa2, 'utf-8')
172 
173 h2_1 = hashlib.shake_128(o2).hexdigest(4)
174 
175 if h1_1==h1 and h2_1==h2:
176     print(True)
177 else:
178     print(False)
```

Table B.5: This implementation of the Verification Scheme of the previous signature follows the description made in the pseudo-code in table 4.5

Bibliography

[1] Manuel Blum, Paul Feldman, and Silvio Micali. «Non-Interactive Zero-Knowledge and Its Applications». In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 329–349. ISBN: 9781450372664. URL: https://doi.org/10.1145/3335741.3335757 (cit. on p. 18).

[2] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. «Non-interactive Zero-Knowledge». In: *SIAM Journal on Computing* 20.6 (1991), pp. 1084–1118. DOI: 10.1137/0220068. eprint: https://doi.org/10.1137/0220068. URL: https://doi.org/10.1137/0220068 (cit. on p. 18).

[3] Amos Fiat and Adi Shamir. «How To Prove Yourself: Practical Solutions to Identification and Signature Problems». In: *Advances in Cryptology — CRYPTO' 86*. Ed. by Andrew M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194. ISBN: 978-3-540-47721-1 (cit. on p. 19).

[4] David Pointcheval and Jacques Stern. «Security Arguments for Digital Signatures and Blind Signatures». In: *Journal of Cryptology* 13 (Oct. 2001). DOI: 10.1007/s001450010003.

[5] Jonathan Katz and Yehuda Lindell. «Introduction to Modern Cryptography (2nd ed.)» In: Chapman and Hall/CRC, 2014. Chap. https://doi.org/10.1201/b17668 (cit. on p. 20).

[6] Koichi Sakumoto. «Public-Key Identification Schemes Based on Multivariate Cubic Polynomials». In: *Public Key Cryptography – PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 172–189. ISBN: 978-3-642-30057-8 (cit. on pp. 25, 27, 29–31, 33, 34).

[7] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. *From 5-pass MQ-based identification to MQ-based signatures*. Cryptology ePrint Archive, Paper 2016/708. https://eprint.iacr.org/2016/708. 2016. URL: https://eprint.iacr.org/2016/708 (cit. on pp. 41, 47).

[8] Thibauld Feneuil. *Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP*. Cryptology ePrint Archive, Paper 2022/1512. https://eprint.iacr.org/2022/1512. 2022. URL: https://eprint.iacr.org/2022/1512 (cit. on pp. 41, 47, 49, 50).

[9] Jacques Stern. «A new paradigm for public key identification». In: *IEEE Trans. Inf. Theory* 42 (1996), pp. 1757–1768. URL: https://api.semanticscholar.org/CorpusID:13963515 (cit. on p. 30).

[10] Carsten Baum and Ariel Nof. «Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography». In: *Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors,* volume 12110 of LNCS (PCK 2020, Part I), pp. 495–526 (cit. on pp. 49, 50).

[11] Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, and Javier Verbel. «An Estimator for the Hardness of the MQ Problem». In: *Progress in Cryptology - AFRICACRYPT 2022*. Ed. by Lejla Batina and Joan Daemen. Cham: Springer Nature Switzerland, 2022, pp. 323–347.

[12] Gora Adj, Luis Rivera-Zamarripa, and Javier A. Verbel. «MinRank in the Head - Short Signatures from Zero-Knowledge Proofs». In: *Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings*. Ed. by Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne. Vol. 14064. Lecture Notes in Computer Science. Springer, 2023, pp. 3–27. DOI: 10.1007/978-3-031-37679-5_1. URL: https://doi.org/10.1007/978-3-031-37679-5%5C_1.

[13] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. *Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature*. Cryptology ePrint Archive, Paper 2021/1576. https://eprint.iacr.org/2021/1576. 2021. DOI: 10.1007/s10623-022-01116-1. URL: https://eprint.iacr.org/2021/1576.

[14] Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. *Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection*. Cryptology ePrint Archive, Paper 2022/223. https://eprint.iacr.org/2022/223. 2022. URL: https://eprint.iacr.org/2022/223.

[15] Thibauld Feneuil and Matthieu Rivain. *Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head*. Cryptology ePrint Archive, Paper 2022/1407. https://eprint.iacr.org/2022/1407. 2022. URL: https://eprint.iacr.org/2022/1407.

[16] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. «Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs». In: *Advances in Cryptology – CRYPTO 2022*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Cham: Springer Nature Switzerland, 2022, pp. 541–572.

[17] Daniel Kales and Greg Zaverucha. «An Attack on Some Signature Schemes Constructed From Five-Pass Identification Schemes». English. In: *Cryptology and Network Security - 19th International Conference, CANS 2020, Vienna, Austria, December 14–16, 2020, Proceedings*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 19th International Conference on Cryptology And Network Security, CANS 2020 ; Conference date: 14-12-2020 Through 16-12-2020. Springer, Dec. 2020, pp. 3–22. ISBN: 9783030654108. DOI: 10.1007/978-3-030-65411-5_1. URL: https://cans2020.at/.

[18] Jacques Stern. «A new identification scheme based on syndrome decoding». In: *Advances in Cryptology — CRYPTO' 93*. Ed. by Douglas R. Stinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 13–21. ISBN: 978-3-540-48329-8 (cit. on pp. 33, 34).

[19] Jacques Stern. «A new paradigm for public key identification». In: *IEEE Transactions on Information Theory* 42.6 (1996), pp. 1757–1768 (cit. on pp. 33, 34).

[20] Jacques Stern. «Designing Identification Schemes with Keys of Short Size». In: *Advances in Cryptology — CRYPTO '94*. Ed. by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 164–173. ISBN: 978-3-540-48658-9 (cit. on pp. 33, 34).

[21] David Pointcheval and Guillaume Poupard. «A New $\mathcal{NP}$ -Complete Problem and Public-Key Identification». In: *Designs, Codes and Cryptography*. 2003, pp. 5–32. URL: https://doi.org/10.1023/A:1021835718426 (cit. on pp. 33, 34).

[22] Adi Shamir. «An Efficient Identification Scheme Based on Permuted Kernels (extended abstract)». In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 606–609. ISBN: 978-0-387-34805-6 (cit. on p. 34).

[23] Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. «A Zero-Knowledge Identification Scheme Based on the q-ary Syndrome Decoding Problem». In: *Selected Areas in Cryptography*. Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–186. ISBN: 978-3-642-19574-7 (cit. on p. 34).