Project 2: Systems of linear disequations

Computer Algebra for Cryptography (B-KUL-H0E74A)

Deadline: 4pm (Leuven time) on 20/05/2025

In this project we will study the not-so-well-known problem of solving systems of linear disequations modulo an integer $N \geq 2$. For well-chosen parameters this is very difficult. We will study connections between this problem and that of solving systems of non-linear equations, we will introduce a candidate signature scheme based on its hardness and, finally, we will break this scheme.

Submission

Your output should consist of three parts: one `.m` file containing all the MAGMA code you wrote during the project, one `.pdf` file containing your report and the AI code of conduct form. Note that the report should be **max 5 pages (not counting references)**. Reports written in LaTeX are preferred, but this is not mandatory. Please follow the numbering of the tasks in this document, i.e., Task 1a and so on.

Make sure that **both** files clearly contain your **name and student number**.

The two files should be submitted via Toledo under Project 2. The deadline is 20/05/2025 at 4pm (Leuven time).

1 Multivariate system solving in Magma

At a couple of points in this project you will face the task of finding solutions to a system of polynomial equations

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0, \\ \qquad\vdots \\ f_m(x_1, \ldots, x_n) = 0 \end{cases}$$

over a finite field $\mathbb{F}_q$. If we want to express that we are only interested in solutions having coordinates in $\mathbb{F}_q$, then we can augment the system with n

so-called field equations, namely

$$x_1^q - x_1 = 0\,, \quad \ldots, \quad x_n^q - x_n = 0\,.$$

Indeed, an element $\alpha \in \overline{\mathbb{F}}_q$, i.e. an element that lives in some field extension $\mathbb{F}_{q^k}$, will be an element of $\mathbb{F}_q$ if and only if $\alpha^q - \alpha = 0$. Adding the field equations thus restricts the set of solutions to those that are defined over $\mathbb{F}_q$ only.

As discussed in the lectures, multivariate systems of polynomial equations can be solved using algorithms for computing Gröbner bases. More precisely, recall that from a Gröbner basis with respect to the lexicographic monomial order, the solutions can be read off by backward substitution. MAGMA provides an optimized combined implementation of all this, accessible through the function `Variety(I::RngMPol) -> SeqEnum`. We remark that this function does not add the field equations by default, so you will need to do that yourself in case you want to incorporate them (see Task 2c).

Example 1. Here is an example usage:

```
> R<x,y,z> := PolynomialRing(GF(3), 3);
> f1 := x^2 + y^2 + x*z + z + 1;
> f2 := x*y + 2*z + 1;
> f3 := x^3 + y^2 + 2;
> I := Ideal([f1, f2, f3]);
> Variety(I);
[ <0, 1, 1>, <0, 2, 1> ]
```

Warning: the command `Variety(I)` will try to list *all* solutions. So it is not a good idea to apply it to largely underdetermined systems.

2 Systems of linear disequations

A system of linear disequations over a commutative ring R is a system of the form

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \neq & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & \neq & b_2, \\ & & & & \vdots & & & & \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \neq & b_m, \end{cases} \tag{1}$$

where the coefficients a_{ij} and constants b_i are elements of R. By solving such a system we usually mean returning the set of all tuples $(x_1, x_2, \ldots, x_n) \in R^n$ satisfying each non-equality. But in some applications we are happy with just one solution, assuming such a solution exists. The system is called *homogeneous* if $b_i = 0$ for all $i = 1, \ldots, m$.

We will always work over rings of the form $R = \mathbb{Z}_N$ (i.e., the ring of integers modulo N) for some integer $N \geq 2$. Throughout this project, we think of N as being fixed and small (in practice we will always have $N \leq 10$). As usual, if N is equal to a prime number p, then we will write $\mathbb{F}_p$ rather than $\mathbb{Z}_p$.

Remark 1. Sometimes, it is a good idea to make this distinction in MAGMA as well. The ring $\mathbb{Z}_N$ can be invoked by `Integers(N)`, but if $N = p$ is a prime number then

```
> Type(Integers(p));
RngIntRes
> Type(FiniteField(p));
FldFin
```

shows that MAGMA makes a distinction between the residue ring $\mathbb{Z}_p$ and the finite field $\mathbb{F}_p$ (somewhat unfortunately). Certain functions, including `Variety`, may deny service when applied to objects (e.g., polynomials, ideals, ...) defined over structures of type `RngIntRes` rather than `FldFin`.

For later use it will be convenient to have a MAGMA function available that generates a random disequation satisfied by a given vector $(s_1, \ldots, s_n) \in \mathbb{Z}_N^n$:

Task 1 (2 marks). *(a) Write a* MAGMA *function*

```
RandomDiseq(s::Tup, homog::BoolElt) -> Tup, RngIntResElt
```

with upon input of a tuple $s = (s_1, s_2, \ldots, s_n) \in \mathbb{Z}_N^n$ and a boolean homog *returns the following output:*

- *if* homog *is false then the output is a uniformly random pair $a = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}_N^n, b \in \mathbb{Z}_N$ such that $a_1 s_1 + a_2 s_2 + \ldots + a_n s_n \neq b$,*
- *if* homog *is true then the output is just a uniformly random $a = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}_N^n$ such that $a_1 s_1 + a_2 s_2 + \ldots + a_n s_n \neq 0$.*

In the homogeneous case it may be assumed that the input tuple s has at least one non-zero entry (otherwise no corresponding output exists).

(b) Give a heuristic reasoning which shows that the expected number of random (not necessarily homogeneous) linear disequations needed for s to be the unique solution is of the form $C_N \cdot n$, where C_N is a constant depending on N only. Give an estimate for C_N.

(c) Is the following assertion true or false (explain your answer)? The number of solutions to a homogeneous system of linear disequations over $\mathbb{Z}_N$ is always a multiple of $\varphi(N)$.

For Task 1a it is useful to know that MAGMA can recover $\mathbb{Z}_N$ and n from s, so there is no need to pass these variables explicitly. E.g., for $\mathbb{Z}_N$ this can be done via `Parent(s[1])`. We also note that $\mathbb{Z}_N^n$ can be constructed from $\mathbb{Z}_N$ and n using the MAGMA intrinsic `CartesianPower`.

Remark 2. Systems of linear disequations were studied by Ivanyos [2] in the context of the *hidden shift problem* for abelian groups $G, +$. This problem is about finding a secret element $s \in G$, called the hidden shift, when being given oracle access to two injective functions

$$f_1 : G \to X, \qquad\qquad f_2 : G \to X$$

with the promise that $f_2(g) = f_1(g + s)$ for all $g \in G$ (here X just denotes some set). Several hard computational problems reduce to an instance of the hidden shift problem. It turns out that for groups of the form $\mathbb{Z}_N^n, +$ there exists an efficient quantum procedure producing random homogeneous linear disequations that are satisfied by the hidden shift $s = (s_1, s_2, \ldots, s_n)$. Therefore, if N is sufficiently small, then any efficient method for solving systems of linear disequations over $\mathbb{Z}_N$ leads to an efficient quantum solution to the hidden shift problem for $\mathbb{Z}_N^n$. One thus sees (see Task 2a below) that in groups of the form $\mathbb{F}_2^n$ the hidden shift problem is very easy to solve by quantum computers. This specific instance of the hidden shift problem is better known as *Simon's problem* (in slight disguise); it is historically the first example of a problem proven to be exponentially harder for classical computers than for quantum computers.

The following is the real starting point for this project:

Task 2 (2 marks). *(a) Show that solving systems of linear disequations over $\mathbb{F}_2$ is equivalent to solving systems of linear equations over $\mathbb{F}_2$, so in this case we can simply proceed by Gaussian elimination.*

(b) More generally: use Fermat's little theorem to write a MAGMA *function*

```
FromDiseqToEq(a::Tup, b::RngIntResElt) -> RngMPolElt
```

which upon input of a tuple $a = (a_1, \ldots, a_n) \in \mathbb{F}_p^n$ and a constant $b \in \mathbb{F}_p$, with p any prime number, returns a polynomial $f \in \mathbb{F}_p[x_1, x_2, \ldots, x_n]$ of degree $p - 1$ such that

$$a_1x_1 + \ldots + a_nx_n \neq b \quad \textit{if and only if} \quad f(x_1, x_2, \ldots, x_n) = 0$$

for all $x_1, x_2, \ldots, x_n \in \mathbb{F}_p$.

(c) Thus: if $N = p$ is prime, every system of linear disequations can be converted into a system of (not necessarily linear) equations. It can therefore be solved by computing a Gröbner basis. Experiment with this strategy for $p = 2, 3, 5, 7$, by selecting a random $s \in \mathbb{F}_p^n$, building a system of (not necessarily homogeneous) linear disequations using the function `RandomDiseq` *from Task 1a, and feeding it to the* MAGMA *intrinsic* `Variety`*. Include a concise report of your experiment in the* `.pdf` *file, where you discuss how the runtimes grow with the number of variables n.*

- *Initially, your system should consist of $C_p \cdot n$ disequations, where C_p is your constant from Task 1b, but you should also analyze the effect of having more disequations, say $r \cdot C_p \cdot n$ for $r = 1, 2, 3, \ldots$ How does this affect the runtimes?*

– *What is the effect of including the field equations $x_1^p - x_1 = 0, \ldots$ in your system?*

(d) *Pinpoint what goes wrong with this strategy when trying to generalize this obervation to non-prime moduli N, using Euler's congruence instead of Fermat's little theorem.*

Despite Task 2d, a similar reduction still applies to moduli N of the form p^k for any prime number p and exponent $k \geq 1$, but this becomes more complicated. The next task handles the case $k = 2$.

Task 3 (4 marks). (a) *Write down a polynomial $W(x, y) \in \mathbb{F}_p[x, y]$ with the property that*

$$W(c,d) = \begin{cases} 0 & \textit{if } \mathrm{Rep}(c) + \mathrm{Rep}(d) < p, \\ 1 & \textit{if } \mathrm{Rep}(c) + \mathrm{Rep}(d) \geq p \end{cases} \tag{2}$$

for all $c, d \in \mathbb{F}_p$. Here Rep *maps an element of $\mathbb{F}_p$ to its unique representant in $\{0, 1, \ldots, p-1\} \subseteq \mathbb{Z}$. In other words, $W(c, d) = 1$ if and only we observe a "wrap-around" modulo p when computing $c + d$.*

Hint: for every (c, d) in the support of (2), find a polynomial mapping (c, d) to 1 and all other points to 0 using Fermat's little theorem; then simply let $W(x, y)$ be the sum of these polynomials.

(b) *Consider the digit-extraction map $\delta : \mathbb{Z}_{p^2} \to \mathbb{F}_p^2 : c_0 + c_1 p \mapsto (c_0, c_1)$ to base p. Implement this as a* Magma *function with the following API:*

```
DigitExtract(c::RngIntResElt) -> FldFinElt, FldFinElt;
```

Also provide code for the converse operation

```
DigitMerge(c0::FldFinElt, c1::FldFinElt) -> RngIntResElt;
```

and prove in your report that $\delta(c + d) = (c_0 + d_0, c_1 + d_1 + W(c_0, d_0))$ for any $c, d \in \mathbb{Z}_{p^2}$, where $\delta(c) = (c_0, c_1)$, $\delta(d) = (d_0, d_1)$.

(c) *Write a* Magma *function*

```
FromDiseqToEq2(a::Tup, b::RngIntResElt) -> RngMPolElt
```

which upon input of a tuple $a = (a_1, \ldots, a_n) \in \mathbb{Z}_{p^2}^n$ and a constant $b \in \mathbb{Z}_{p^2}$ returns a multivariate polynomial $f \in \mathbb{F}_p[x_{11}, x_{12}, x_{21}, x_{22}, \ldots, x_{n1}, x_{n2}]$ having the property that

$$a_1 x_1 + \ldots + a_n x_n \neq b \quad \textit{if and only if} \quad f(\delta(x_1), \delta(x_2), \ldots, \delta(x_n)) = 0$$

for all $x_1, x_2, \ldots, x_n \in \mathbb{Z}_{p^2}$. Hint: first compute polynomials f_0, f_1 such that $\delta(a_1 x_1 + a_2 x_2 + \ldots + a_n x_n - b) = (f_0(x_{11}, \ldots, x_{n2}), f_1(x_{11}, \ldots, x_{n2}))$ where, as a subhint, we note that $a_1 x_1 + a_2 x_2 + \ldots$ can be read as

$$\underbrace{x_1 + x_1 + \ldots + x_1}_{a_1 \text{ times}} \quad + \quad \underbrace{x_2 + x_2 + \ldots + x_2}_{a_2 \text{ times}} \quad + \quad \ldots$$

(d) *Repeat the experiment from Task 2c for $p^2 = 4$ and $p^2 = 9$. Combine the* MAGMA *intrinsic* `Variety` *with your implementation of* `DigitMerge` *so that you can easily verify that s is indeed among the solutions you found.*

(e) *Briefly discuss in your report which problem you think is easier: solving systems of linear disequations over the ring $\mathbb{Z}_{p^2}$, or solving systems of linear disequations over the field $\mathbb{F}_{p^2}$?*

Beyond moduli of the form $N = p^k$, no reduction from systems of linear disequations to systems of equations is known. In this case (e.g., when $N = 6$), we know of no better solution than exhaustively trying all possible options for $s = (s_1, s_2, \ldots, s_n)$, each time verifying whether each disequation in (1) is indeed satisfied or not. The expected runtime of this approach is $\tilde{O}(N^n)$.

On the other hand, in the prime power case where our system does translate into a system of equations, Gröbner basis algorithms will perform better than exhaustive search, at least if we are allowed to assume that this system is *semi-regular* (whatever this means). The reason is that we have $m \approx C_N \cdot n$ equations at our disposal, which is considerably more than the number of unknowns.

Task 4 (1 mark). *Assume that the system is semi-regular, so that we can determine the degree of regularity d_{reg} using the Hilbert series $H(z)$ as on slide 44 of Lectures 6 & 7. Do this for $N = p = 3$ and increasing values of n, and make an extrapolating guess for the asymptotic behaviour of d_{reg} as n tends to infinity. What can we conclude about the asymptotic runtime of the Gröbner basis attack? How does it compare to the exhaustive search method?*

In certain use cases, including Ivanyos' quantum solution to the hidden shift problem discussed in Remark 2, the situation is even better and we have access to arbitrarily many linear disequations. In this case the problem can be solved in polynomial time via a technique called "linearization". In more detail and focusing on the case where $N = p$ is a prime number: by treating every monomial of degree at most $p - 1$ as a new indeterminate, one can just view each non-linear equation as a *linear* equation and solve the resulting linear system via Gaussian elimination.

Task 5 (2 marks). *(a) Give an estimate for the expected number of linear disequations that are needed for this method to succeed, and discuss some subtleties that arise when the solution is not unique.*

(b) *Write a* MAGMA *function*

```
FromDiseqToLinEq(a::Tup, b::RngIntResElt) -> ModMatFldElt, FldFinElt
```

which upon input of a tuple $a = (a_1, \ldots, a_n) \in \mathbb{F}_p^n$ and a constant $b \in \mathbb{F}_p$ returns a row matrix containing the coefficients (including zeroes) of the output of `FromDiseqToEq(a::Tup, b::RngIntResElt)` *at all non-constant monomials of degree at most $p - 1$. As a second output, it returns the coefficient at the monomial 1, i.e., the constant term.*

(c) *Experiment with the linearization technique for $p = 3, 5, 7$, by fixing a random $s \in \mathbb{F}_p^n$ and generating (not necessarily homogeneous) linear disequations using* `RandomDiseq`*, until the corresponding linearized system has full rank (so that there is a unique solution). Does the number of disequations needed match well with your prediction from (a)?*

Here, a useful command is `MonomialsOfDegree(R, d)`, which returns the list of all monomials of degree `d` in a polynomial ring `R`.

3 A candidate Σ-protocol from disequations

From now on we will assume that our system of linear disequations (1) is homogeneous. We will write it in matrix form as

$$\mathbf{A} \cdot \mathbf{x} \not\equiv \mathbf{0},$$

where $\mathbf{A} = (a_{ij})_{1 \leq i \leq m, 1 \leq i \leq n} \in \mathbb{Z}_N^{m \times n}$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is viewed as a column vector. The symbol $\not\equiv$ should thus be read as "differs at *each* entry from".

We consider the following protocol in which a prover, called Peggy, aims at proving knowledge of a solution $\mathbf{s} \in \mathbb{Z}_N^n$ to the system (1) without revealing it. The protocol involves two rounds of communication between Peggy and a verifier named Victor. It is convenient to introduce the notation $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} \in \mathbb{Z}_N^m$, where we note that $\mathbf{t} \not\equiv \mathbf{0}$. Throughout the protocol it is assumed that $m = C_N \cdot n$ with C_N as in Task 1b. For any given matrix $\mathbf{C} \in \mathbb{Z}_N^{m \times m}$ we write $\mathbf{C}^{\mathrm{trunc}(n)}$ for its upper-left $n \times n$ submatrix.

1. **Commitment:** Peggy chooses a uniformly random matrix $\mathbf{C} \in \mathbb{Z}_N^{m \times m}$ such that
   - $\mathbf{C}^{\mathrm{trunc}(n)} \in \mathrm{GL}_n(\mathbb{Z}_N)$, i.e., $\det \mathbf{C}^{\mathrm{trunc}(n)} \in \mathbb{Z}_N^*$,
   - $\mathbf{C} \cdot \mathbf{t} \not\equiv \mathbf{0}$.

   She sends $\mathbf{M} := \mathbf{C} \cdot \mathbf{A} \cdot \mathbf{C}^{\mathrm{trunc}(n)} \in \mathbb{Z}_N^{m \times n}$ to Victor.
2. **Challenge:** Victor replies with a challenge bit $c \in \{0, 1\}$.
3. **Response:** If $c = 0$, then Peggy reponds with $\mathbf{C}$. If $c = 1$ then she responds with $\mathbf{m} := (\mathbf{C}^{\mathrm{trunc}(n)})^{-1} \cdot \mathbf{s}$.
4. **Verification:** If $c = 0$ then Victor accepts the response if $\mathbf{M} = \mathbf{C} \cdot \mathbf{A} \cdot \mathbf{C}^{\mathrm{trunc}(n)}$. If $c = 1$ then he accepts the response if $\mathbf{M} \cdot \mathbf{m} \not\equiv \mathbf{0}$.

If a protocol of this type satisfies certain robustness properties, then it can claim to be a Σ-protocol [1], which is an important type of cryptographic building block that can be used a.o. to construct unforgeable signatures via the Fiat–Shamir transform (see below). The terminology just comes from the zigzag shape of the Greek letter Σ, which visualizes the back-and-forth communication between Peggy and Victor. Informally, the necessary properties are:

- *completeness*: if Peggy and Victor honestly follow the protocol, then Victor always accepts Peggy's response,
- *soundness*: anyone able to give valid responses to both $c = 0, 1$ for the same commitment $\mathbf{M}$ must know a solution to $\mathbf{A} \cdot \mathbf{x} \not\equiv \mathbf{0}$, or can efficiently compute one,
- *zero-knowledge*: executing the protocol does not leak any information about $\mathbf{s}$ (or about any other solution to $\mathbf{A} \cdot \mathbf{x} \not\equiv \mathbf{0}$)

The formal requirements are slightly more technical and are omitted, but see [1].

Task 6 (3 marks)**.** *Show that the protocol is complete and implement it in* Magma *by providing functions with the following APIs:*

(a) `KeyGen(N::RngIntElt, n::RngIntElt) -> ModMatRngElt, ModMatRngElt` *first generates a uniformly random non-zero* $\mathbf{s} \in \mathbb{Z}_N^{n\times 1}$ *and then builds a matrix*
$$\mathbf{A} \in \mathbb{Z}_N^{m\times n} \quad \textit{such that} \quad \mathbf{A} \cdot \mathbf{s} \not\equiv \mathbf{0}$$
using $m = C_N \cdot n$ *invocations of* `RandomDiseq`*. It returns s and A.*

(b) `Commit(s::ModMatRngElt, A::ModMatRngElt) -> ModMatRngElt, ModMatRngElt` *similarly returns a matrix* $\mathbf{C} \in \mathbb{Z}_N^{m\times m}$ *such that* $\mathbf{C}\cdot\mathbf{t} = \mathbf{C}\cdot\mathbf{A}\cdot\mathbf{s} \not\equiv \mathbf{0}$*, where moreover it is ensured that* $\mathbf{C}^{\mathrm{trunc}(n)}$ *is invertible; it then also returns the matrix* $\mathbf{M} = \mathbf{C} \cdot \mathbf{A} \cdot \mathbf{C}^{\mathrm{trunc}(n)}$.

(c) `Response(c::RngIntElt, s::ModMatRngElt, C::ModMatRngElt) -> ModMatRngElt` *responds to the challenge* $c \in \{0, 1\}$*: it just returns* $\mathbf{C}$ *if* $c = 0$*, while it returns* $(\mathbf{C}^{\mathrm{trunc}(n)})^{-1} \cdot \mathbf{s}$ *in case* $c = 1$.

(d) `Verify(c::RngIntElt, A::ModMatRngElt, M::ModMatRngElt, resp:ModMatRngElt) -> BoolElt` *returns* `true` *if* **resp** *is indeed a valid response to challenge bit c; it returns* `false` *if it is not.*

Note that an impersonator – let us call her Eve – has a 50% chance of guessing Victor's challenge bit in advance and act accordingly:

- if she guesses $c = 0$, she just generates some random $\mathbf{C} \in \mathbb{Z}_N^{m\times m}$, computes $\mathbf{M} = \mathbf{C} \cdot \mathbf{A} \cdot \mathbf{C}^{\mathrm{trunc}(n)}$ and sends this to Victor; if Victor then indeed challenges her with $c = 0$, she can simply respond with $\mathbf{C}$ and this will pass Victor's verification,

- if she guesses $c = 1$, then she just generates some random non-zero $\mathbf{m} \in \mathbb{Z}_N^{m\times 1}$, after which she generates a random matrix $\mathbf{M} \in \mathbb{Z}_N^{m\times n}$ such that $\mathbf{M} \cdot \mathbf{m} \not\equiv \mathbf{0}$ and sends it to Victor; if Victor indeed challenges her with $c = 1$, she can just respond with $\mathbf{m}$ and pass the verification.

The idea behind soundness is that in a Σ-protocol with challenge space $\{0, 1\}$, Eve's chances of deceiving Victor are limited to this 50%. Consequently, if the protocol is repeated λ consecutive times, then the probability that Eve's cheating goes unnoticed drops to $1/2^\lambda$, which soon becomes negligible.

We now explain how to turn this into a signature scheme: this is done through a generic transformation known as the Fiat–Shamir transform. Peggy publishes $\mathbf{A}$ as her public key; the secret key is her solution $\mathbf{s}$ to $\mathbf{A} \cdot \mathbf{x} \not\equiv 0$. If she wants to sign a message `M`, she proceeds as follows:

1. She computes λ random matrices $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_\lambda \in \mathbb{Z}_N^{m\times m}$ satisfying the requirements from the commitment phase above, along with the corresponding matrices $\mathbf{M}_i = \mathbf{C}_i \cdot \mathbf{A} \cdot \mathbf{C}_i^{\text{trunc}(n)} \in \mathbb{Z}_N^{m\times n}$ for $i = 1, 2, \ldots, \lambda$.

2. She computes $c := H((\mathbf{A}, \mathtt{M}, \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_\lambda)) \in \{0, 1\}^\lambda$, where H is a secure hash function.

3. The bits in $c = c_1 c_2 \cdots c_\lambda$ now serve as challenge bits for the commitments $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_\lambda$: Peggy's signature is the tuple

$$(\mathbf{M}_1, \mathbf{resp}_1, \mathbf{M}_2, \mathbf{resp}_2, \ldots, \mathbf{M}_\lambda, \mathbf{resp}_\lambda),$$

   where $\mathbf{resp}_i$ are the corresponding responses.

4. Victor can now recompute $c := H((\mathbf{A}, \mathtt{M}, \mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_\lambda))$ and verify all Peggy's responses. If each $\mathbf{resp}_i$ passes, then he accepts the signature.

Observe that the challenge bits no longer come from Victor: they are generated by the hash function and as such the protocol has become *non-interactive* (this is the crux of the Fiat–Shamir transform).

In the following you can use the simple hash function below, which takes as input a string and returns a vector of 160 bits. Note that this hash function is not considered secure anymore due to its use of SHA1.

```
function SimpleHash(s)
    input := IntegerToString(SequenceToInteger(StringToBytes(s), 256),16);
    return Intseq(StringToInteger(SHA1(input),16),2,160);
end function;
```

Task 7 (2 marks)**.** *Implement the above signature scheme for* $\lambda = 160$*, by providing functions with the following APIs:*

(a) `DiseqSign(s::ModMatRngElt, A::ModMatRngElt, mess::MonStgElt) -> Tup` *takes as input a secret key* $\mathbf{s} \in \mathbb{Z}_N^{n\times 1}$*, a public key* $\mathbf{A} \in \mathbb{Z}_N^{m\times n}$*, a message* `mess`*, and returns a corresponding signature, which is a length-*254 *tuple of matrices over* $\mathbb{Z}_N$*.*

(b) `VerifySign(sig::Tup, A::ModMatRngElt, mess:MonStgElt) -> BoolElt` *takes as input a signature output by* `DiseqSign` *along with a public key* $\mathbf{A} \in \mathbb{Z}_N^{m\times n}$ *and a message* `mess`*, and returns a boolean indicating whether or not the signature is accepted.*

Test your functions by generating a private and public key for the parameters $N = 6$ *and* $n = 30$ *using* `KeyGen`*, letting Peggy sign the message*

```
I will pay 10 EUR to Victor. Best wishes, Peggy
```

and having the signature verified by Victor. Include the signing and verification times in your report, and briefly discuss how the sizes of the private key $\mathbf{s}$*, the public key* $\mathbf{A}$ *and the signature grow with* N, n, λ*.*

Remark 3. In Tasks 7a and 7b it is useful to know about the command `Sprint` which converts any Magma object into a string.

4 Attacking the protocol

The candidate Σ-protocol from Task 6 is complete, but let us now zoom in on the other requirements: soundness and zero-knowledge. The latter requirement is not satisfied in case $N = p$ is a prime number (or, more generally, a prime power):

Task 8 (1 mark)**.** *Explain why for* $N = p$ *a prime number, many repeated runs of the protocol eventually allow an eavesdropper to find a solution to* $\mathbf{A} \cdot \mathbf{x} \not\equiv \mathbf{0}$ *using linear algebra. Hint: concentrate on the runs with challenge* $c = 0$*.*

This can be countered by moving away from prime moduli and, e.g., working with $N = 6$ as we already did in Task 7. In this case, the protocol may indeed enjoy the zero-knowledge property. However, it does not suffice that Eve cannot learn anything about Peggy's secret $\mathbf{s}$ for this to be a secure signature scheme! Indeed, the soundness part of our protocol is absolutely problematic:

Task 9 (3 marks)**.** *Show that, in contrast with the discussion following Task 6, Eve can actually impersonate Peggy and compute instances of* $\mathbf{M}, \mathbf{C}, \mathbf{m}$ *that allow her to respond to both* $c = 0$ *and* $c = 1$*, without knowledge of* $\mathbf{s}$ *and only using linear algebra. Implement this and incorporate it into a* Magma *function*

```
DiseqSignForge(A::ModMatRngElt, mess::MonStgElt) -> Tup
```

with the same output as `DiseqSign(s, A, mess)`*, but this time* without *relying on the secret key as input. Test your method by generating a private and public* *key for the parameters $N = 6$ and $n = 30$ using* `KeyGen`*, letting Eve forge a signature to the message*

```
I will pay 10000 EUR to Eve. Best wishes, Peggy
```

using `DiseqSignForge`*, and having the signature verified by Victor using your function* `VerifySign`*.*

References

[1] Ivan Damgård, *On Σ-protocols*, notes of the course Cryptologic Protocol Theory, Aarhus Universitet, available at `https://www.cs.au.dk/~ivan/Sigma.pdf`

[2] Gábor Ivanyos, *On solving systems of random disequations*, Quantum Information and Computation **8**(6&7), pp. 0579–0594 (2008), available at `https://arxiv.org/abs/0704.2988`