Project 1: Analyzing a cryptosystem based on the finite fields isomorphism problem

Clémence Duplat r0977801

April 5, 2024

1 Task 1

We will implement an algorithm to find an isomorphism between two finite fields represented by irreducible polynomials over a finite field $\mathbf{F}_p$.
To do so, we need to find the roots of the polynomial $f(z)$ in $\mathbf{Y}$. We can thus implement the root finding algorithm, which is a special case of the complete factoring algorithm.

We will use the fact that the polynomial $f(z)$ split completely over $\mathbf{Y}$ i.e. in degree 1 factors. Therefore, there is no need to implement a complete polynomial factoring algorithm because every polynomial in this factorization has degree 1*.
Using repeated squaring in $Y[z]/f(z)$, we will simply compute :

$$h = gcd(z^q - z, f(z))$$

this will give us the common factor between $z^q - z$ and $f(z)$.
If this common factor is equal to 1, f will have to roots in $\mathbf{Y}$. Otherwise, we will call the Cantor-Zassenhaus algorithm (equal-degree factorisation) on h to find factors of $x - u_i$ for i=1,...,deg(h) with u_i the roots of f over $\mathbf{F}_q$.

The Cantor-Zassenhaus algorithm is a probabilistic algorithm that is used for factoring polynomials over a finite field. It will compute 2 other polynomials, that will serve for identifying non-trivial factors of a given polynomial over a finite field.

We will define a random polynomial $r(z)$, of one degree less than d (degree of f(z)), defined in the same finite field. The randomness of this polynomial is crucial for the probabilistic nature of the algorithm and will increase the probability of finding all factors and thus the roots.

We will then compute a second polynomial using the square-and-multiply method: $r_2(z) = r(z)^{(q^d-1)/2} \mathrm{mod} f(z)$. Note that q is the size of the extension field $\mathbf{Y}$ and that d is the degree of the irreducible polynomial defining $\mathbf{Y}$, but we can consider that $d = 1$ as said before*. We know that in a finite field every element is either a quadratic square or quadratic non-square. Computing $r_2(z)$ will identify which components of $f(x)$ are quadratic squares in the field. This will help us to find polynomials that divides f(z) only partially and thus lead to a factorization of f(z). We can then compute the GCD of $r_2(z) - 1$ and $f(z)$. If this new polynomial computed with the GCD is equal to $f(z)$, it failed to find a splitting. On the other hand, if is it not equal to $f(z)$ and not equal to 1, then the GCD is a non-trivial factor of f(z), which indicates that roots of f(z) in Y have been found.

In other words, this algorithm has split $f(z)$ into smaller and more manageable pieces that will eventually lead to its complete factorization.

Once at least one root of $f(z)$ in $\mathbf{Y}$ is found, we can map the generator of $\mathbf{X}$ (defined as the polynomial ring of $\mathbf{F}_p$) to the root of $f(z)$ in $\mathbf{Y}$. This mapping will extend linearly to the whole field $\mathbf{X}$ because both fields are vector spaces over $\mathbf{F}_p$ of the same dimension.
We are in the context of finite fields, so once we have one isomorphism, computing other isomorphisms is simply a matter of applying powers of the Frobenius automorphism to the known isomorphism, because the finite field is cyclic.

The complexity of our algorithm is dominated by the root finding algorithm and is defined by $O(M(n)log(n)log(nq))$ where n is the degree if f(z), q the size of the extension field **Y** and M(n) the complexity of multiplying two polynomials of degree n in **Y**.
For the inverse isomorphism algorithm, we will need to find a polynomial t(z) such that any y in **Y**, s(t(y)) = y mod g(y). We start by calculating the the roots of g(z) in **X**. For each root found, we construct a polynomial t in $F_p[z]$ that correspond to the root coordinate in **X**. If evaluating s(t) at this polynomial is the identity element in X, it found an inverse isomorphism polynomial.
We can report the timings of our 2 algorithms asked in question 1.*b* and 1.*c* with $p = 2^{16} + 7$ and $n = 2^k$ for $k = 2, ...6$.

k	Isomorphism	Inverse Isomorphism
2	0.0	0.0
3	0.015	0.016
4	0.203	0.250
5	4.172	4.000
6	81.610	71.390

Table 1: Timings for the algorithms determining the (inverse) isomorphism of two irreducible polynomials, expressed in seconds.

Both algorithm are tested to see if it correctly translate elements from one field representation to the other and back again. This is done in Magma.

2 Task 2

All the functions here are implemented in Magma and well documented. They are tested seeing if encrypting a message and then decrypting it recovers the original message. To do so, it take on average 3.094 seconds.

3 Task 3

3.1 Task 3.a and 3.b

We need to devise an attack using simple linear algebra to recover the isomorphism ϕ represented by the polynomial $s(z)$. To do so, we will formulate a system of linear equations where the unkown will be the coefficients of the polynomial $s(z)$. The linear system will be as follow :

$$AS = B$$

The matrix A is initialized the over a polynomial ring in order to be able to populate the matrix with values derived from evaluating polynomials at specific points. If we suppose a set of n polynomials over a ring, where p_n is a polynome of degree n, the matrix A can be constrcuted as follow:

$$A = \begin{bmatrix} p_1(X!(m_1+2r_1)) & p_2(X!(m_1+2r_1)) & \cdots & p_n(X!(m_1+2r_1)) \\ p_1(X!(m_2+2r_2)) & p_2(X!(m_2+2r_2)) & \cdots & p_n(X!(m_2+2r_2)) \\ \vdots & \vdots & \ddots & \vdots \\ p_1(X!(m_n+2r_n)) & p_2(X!(m_n+2r_n)) & \cdots & p_n(X!(m_n+2r_n)) \end{bmatrix}$$

The vector B can be constructed at the hand of the ciphertexts

$$B = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

Once we have the system of equation, we can find a solution using linear algebra techniques such as Gaussian elimination. In magma, we will use the function *Solution(A,B)* in order to find the matrix S containing the coefficients of the polynomial $s(z)$.
However, when implementing it in Magma, I noticed that I hadn't direct access to f(z) (which is logic) and thus no access to **X**. Therefore, I used the extension field generated by g to do so, which I know is not entirely correct. Indeed,

my isomorphism is always correct, which does not correspond to the mathematic probability of failure of this algorithm. Our algorithm will work if our matrix is of full rank or in other words of rank n.If we encounter linear dependencies between the rows, we would not have a matrix of full rank anymore.
If we have less than n triples available, our system could be undetermined.

The function *KnownPlaintextNoiseAttack* is implemented in Magma and well commented. We can test our attack with $n = 32$ and $p = NextPrime(2^k)$ for k = 1, ... 8. With my current algorithm I was unfortunately not able to perform the timings with a n that is equal to 32. I stayed hours and hours on this part but did not manage to find a correct solution. Due to this, I didn't found the time to finish point 3c and 3d.

4 task 4

4.1 Task 4.a

To explain why the trace indeed takes values in $\mathbf{F}_p$, we will use the hint and compute $Tr(a)^p$.

$$Tr(a)^p = (a + ... + a^{p^{(n-1)}})^p = a^p + a^{p^2} ... + a^{p^n}$$

For the last part of the equality, we used the Freshman's Dream theorem that states, only if $\mathbf{F}$ is a prime number and $n = p$, that $(x+y)^n = x^n + y^n$ for all x,y in the field.
We know that $\mathbf{F}_{p^n}$ is closed under the operations, we know that applying the p^n (which is the degree of the field extension) power to an element will give us back the element itself. Indeed, it is equivalent to applying the Frobenius automorphism n times, corresponding to the identity over $\mathbf{F}_{p^n}$. Therefore, we have that $a^{p^n} = a$ and this is known as the Fermat's little theorem. We can proceed and see that

$$Tr(a)^p = (a + ... + a^{p^{n-1}})^p = a^p + a^{p^2} + a^{p^n} = a^p + a^{p^2} ... + a^{p^{n-1}} + a = Tr(a)$$

So $Tr(a)^p = Tr(a)$ and by Fermat's little theorem it shows $Tr(a)$ behaves like an element of $\mathbf{F}_p$.

Now we will explain why for any field isomorphism

$$\phi : X \to Y$$

we have $\mathrm{Tr}(\phi(a)) = \mathrm{Tr}(a)$ for all $a \in X$. We can start by computing:

$$\mathrm{Tr}(\phi(a) = \phi(a) + ...\phi(a)^{p^{n-1}}$$

Here, we will use some properties of the field isomorphism ϕ. The first one is the fact that it preserves operations in F_{p^n} and thus that $\phi(a)^p = \phi(a^p)$. Therefore, we have

$$(\mathrm{Tr}(\phi(a)) = \phi(a) + ...\phi(a^{p^{n-1}})$$

The second important property of the field isomorphism ϕ is the linearity.

$$\phi(a) + ..\phi(a^{p^{n-1}}) = \phi(a + ..a^{p^{n-1}}) = \phi(Tr(a))$$

We know that $Tr(a)$ is in $\mathbf{F}_p$ (only takes elements in $\mathbf{F}_p$) and that ϕ is an isomorphism which fixes $\mathbf{F}_p$ and thus $\phi(Tr(a)) = Tr(a)$. Finally we recover that :

$$Tr(\phi(a)) = Tr(a)$$

4.2 task 4.b

We want to show that the trace is $\mathbf{F}_p$ linear, or in other words that in the following sense: for all $\lambda, \mu \in \mathbb{F}_p$, and all $a, b \in X$ we have $\mathrm{Tr}(\lambda a + \mu b) = \lambda \mathrm{Tr}(a) + \mu \mathrm{Tr}(b)$ To do so, we need to prove that the additivity and homogeneity properties hold.

We have for all $\lambda, \mu \in \mathbb{F}_p$, and all $a, b \in X$:

$$Tr(a+b) = (a+b) + ...((a+b)^{p^{n-1}}$$

$$Tr(a+b) = (a+...a^{p^{n-1}}) + ...(b+...b^{p^{n-1}}) = Tr(a) + Tr(b)$$

and that

$$Tr(\lambda a) = (\lambda a) + ...(\lambda a)^{p^{n-1}}$$

By using fermat's little theorem on λ, we get the following equality : $\lambda = \lambda^p$. We thus get that :

$$Tr(\lambda a) = \lambda((a) + ...(a)^{p^{n-1}}) = \lambda Tr(a)$$

We can proceed with the same reasoning for $Tr(\mu a) = \mu Tr(a)$.

Finally we can thus find the following equality

$$\mathrm{Tr}(\lambda a + \mu b) = \lambda \mathrm{Tr}(a) + \mu \mathrm{Tr}(b)$$

which proves that the trace is $\mathbf{F}_p$-linear.

4.3 task 4.c

We proved before that the trace is $\mathbf{F_p}$-linear, allows us to get the following:

$$Tr(m+2r) = Tr(m) + 2 * Tr(r)$$

This will allows us to bound $Tr(m)$ and $Tr(r)$ separately. In our context, m is represented as a small element in $\mathbf{X}$ and r is the noise term also in $\mathbf{X}$.