Multivariate Cryptography

Introduction to Quantum-Safe Cryptography (IBM Zurich),
Programming assignment
July 1 - July 5, 2024

Remarks:

- You will need Python and Sagemath for completing this exercises
- You can follow the instructions at `https://sagemanifolds.obspm.fr/install_ubuntu.html` to install the latest version of Sagemath and use it within a Jupyter notebook (you can also use Sagemath in a console, if you prefer)

Goals: After completing the exercises in this programming assignment you will have a solid understanding of how UOV works and what are the currently best algorithms for cryptanalysis. You will further have an implementation of both the scheme and the attacks.

1. **(UOV design)** In this first exercise ywe will implement UOV. To make it easier for you, we will guide you in the process and provide you with steps you can follow and turn them into an actual implementation.

   (a) The easiest way to implement multivariate cryptography is to use a matrix representation of the (homogeneous) quadratic maps. Therefore it is usefull to create helper functions for

   - Generating random invertible matrices
     `RandomInvertible(n)`
   - Turning a square matrix to upper-diagonal
     `SquareToUpper(M)`
   - Turning an upper diagonal matrix to symmetric
     `UpperToSymmetric(M)`
   - Evaluation of multivariate maps.
     `MQeval(F,x,y)`

   (b) Create a function for key generation that returns the public key and the secret key
   `Keygen(q,n,m)`

   (c) Create a function for signing a message m that returns the message and the signature
   `Sign(message, private_key)`
   You will need to hash the message first, for which you can use `hashlib`

   (d) Create a function for verification of a signature
   `Verify(message, signature, public_key)`

2. **(UOV key compression)** The previous exercise is not concerned with the sizes of the keys. We did not even bother to store them well. In this exercise we will apply an optimization on UOV, that helps reduce the key sizes. We will use the equivalent keys technique to generate the keys

   (a) Update the key generation algorithm from the previous exercise to implement the equivalent keys technique for generating the public and the private key. Make sure to use a public and a private seed. In the interest of time, you don't have to implement the seed generation in a cryptographically secure manner, but you can try this at home.

   (b) In order to be able to use the keys for signing and verification, you will need to decompress the keys in the form used in the signing and verification procedure.
   `Decompress(pk_compressed, sk_compressed)`

3. **(UOV attacks)** Finally we implement (some of) the state-of-the-art algorithms against UOV:

   (a) Direct signature forgery attack

   - For this attack you simply need to run an algebraic solver for a given signature value. Sage has a (not so fast) implementation of Faugere's F4 algorithm that you can use. Recall that since UOV gives an underdetermined system of equations, you need to first fix a certain number of variables.

   (b) Reconciliation attack

   - For educational purposes, you can start with the simplest iterative description of the attack, where in each step you find one oil vector.
   - Then notice how many vectors you will need to look for at once in the first iteration.
   - Afterwards notice, and implement the remaining steps as solving a linear system of equations

   (c) Intersection attack

   - The intersection attack looks for one intersection vector that maps to two different oil vectors that can be used in the reconciliation attack. Thus you can reuse the code from the previous point. You only need to implement an assitional part for forming the intersection vector.

   (d) Rectangular MinRank attack

   - This attack consists of two parts:
     - Changing "the view" on the public matrices (basically turning them around for 90 degrees)
     - Implementing a MinRank attack

     These can be implemented as two separate functionalities. You are free to use any MinRank model, but we recommend the simple Kipnis-Shamir model, that nicely corresponds to direct retrieval of the secret input change of basis (the secret $\mathcal{S}$).