Post-quantum cryptography
PCMI 2022 - USS
Supplementary notes

Christelle Vincent

July 25, 2022

Contents

The purpose of this handout is to present some of the main post-quantum ciphers that are currently being studied.

1 Post-quantum cryptography

In addition to giving an algorithm to factor any integer N in quantum-polynomial time, Shor's 1994 article *Algorithms for quantum computation: discrete logarithms and factoring* also presented an algorithm to compute so-called discrete logarithms in quantum-polynomial time. Together, these two problems (the factoring problem and the discrete logarithm problem) form the basis of practically every single public-key cryptosystem that is currently used on the internet. From Shor's work it therefore follows that as soon as quantum computers that can compute with 1000 to 2000 qbits are deployed, the very very vast majority of internet traffic, internet-enabled devices, and encrypted information will no longer be secure.

Though we do not currently know when such computers will be available, the National Institute of Standards and Technology (NIST) is preparing for such a device to exist around 2030. To be clear, at first only large entities and corporations (countries such as China, Russia, and the United States, and companies like Google, Microsoft, and IBM, for example) will have the resources to own and run quantum computers large enough to break encryption.

Nevertheless there is a need to protect our communications and data from these large actors, and for this reason fundamental research is currently underway to design and analyze new hard problems on which to base cryptographic systems that are not vulnerable to attacks from either classical computers or quantum computers.

Post-quantum cryptography refers to cryptographic schemes that can be implemented on classical (non-quantum) computers, but that are based on such problems; problems which are currently believed to only admit exponential-time solutions on both classical and quantum computers. There are currently four main families of problems considered for this purpose. We present them in decreasing order of mathematical sophistication (which is not to say that the algorithms are less sophisticated; we simply mean the amount of mathematics necessary to understand the statement of the problem, and of the encryption and decryption or signature algorithm(s)):

1.1 Isogeny-based cryptography

The background for this section would take us much too far afield, so these notes actually will not discussed isogeny-based cryptography. We note though that one isogeny-based algorithm is going on to the fourth round of NIST post-quantum standardization competition.

1.2 Lattice-based cryptography

The security of lattice-based cryptography is based on the hardness of finding the shortest nonzero vector in a lattice. This problem is often abbreviated SVP for Shortest Vector Problem. This is what we will study for the rest of our time together so we do not say more here.

2022 note: Three of the four algorithms to be standardized by NIST for post-quantum cryptography are lattice-based.

1.3 Code-based cryptography

In mathematics, a *binary code of length n and dimension k*, which we'll denote $\mathcal{C}$, is a set containing 2^k strings of length n containing only the characters 0 and 1, such that if c_1 and c_2 are two strings in $\mathcal{C}$, then the bitwise sum of c_1 and c_2 is also in $\mathcal{C}$. In fancier words, we can say that a binary code of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_2^n$, where $\mathbb{F}_2$ is the field with two elements.

For example, the binary code $\mathcal{C} = \{000, 111\}$ has length 3 and dimension 1. Since we have that $000 + 000 = 000$, $000 + 111 = 111$, and $111 + 111 = 000$ when we perform the bitwise sum, we see if $c_1, c_2 \in \mathcal{C}$, then $c_1 + c_2 \in \mathcal{C}$ as well, so $\mathcal{C}$ does respect the sum condition. (We say that $\mathcal{C}$ is **closed under addition** when this is the case.)

Our interest in codes is that since a code contains only 2^k vectors from within $\mathbb{F}_2^n$, if k is smaller than n there can some **distance** between two elements of the code. The most common and simple notion of distance is the **Hamming distance**, which just counts the number of bits in which two strings differ: For example, the Hamming distance between 000 and 111 is three since the two string differ in all three positions.

We can leverage this distance in the following way: Suppose that we agree that 0 corresponds to the codeword 000 and 1 corresponds to the codeword 111. In this way there is a larger distance between the codewords than between the original strings.

This distance allows us to better distinguish between 0 and 1. This can be important, for example in case there is noise on a communication channel. If someone sends the codeword 000, we might receive the string 001, which is not a codeword. However, we know that 001 is closer to the codeword 000 than to the codeword 111. Therefore it's perhaps more likely that the original codeword sent was 000, and we decode the string 001 to correspond with the codeword 000 (and so the message 0; notice that encoding and decoding are **not** inverses of each other).

For our purposes, we will need to assume that our codes have an efficient **decoding algorithm**. This is a map from all of $\mathbb{F}_2^n$ back to the original set of 2^k strings of length n that form our code. Usually, this map sends an element of $\mathbb{F}_2^n$ to the codeword $c \in \mathcal{C}$ that is closest (in the Hamming distance) to it. When n is large, computing the nearest c can be a very long computation, but mathematicians have tricks to create codes in clever ways so that decoding is fast.

Now to be clear, coding theory (the study of codes) has nothing to do with cryptography. Its original purpose was to study the amount of information in data, how to compress data, and how to transmit data over noisy channels. However, in 1978 McEliece suggested a scheme that uses the error-correcting properties of certain codes as the basis for the decryption of messages. While this scheme, which is now called "Classic McEliece" was not very popular at the time, it is now a finalist in NIST's post-quantum cryptography competition (2022 note – it was not chosen to be standardized at this time, but it is going on to a fourth round of competition; two more code-based schemes were also chosen to go on to the fourth round), and there is renewed interest in developing codes that are fast and secure to use with McEliece's scheme.

The scheme goes as follows: Suppose that it is possible to create a code such that its encoding algorithm can be made public while its decoding algorithm remains secret. Then to receive messages, A picks such a code and publishes the encoding algorithm. A keeps the decoding algorithm private. If B wants to send A a message m, B first encodes m into a codeword c. Then B chooses a small noise vector $r \in \mathbb{F}_2^n$ at random, and computes and communicates publicly the sum $c + r$.

If r is small enough, the sum $c + r$ should decode to the original codeword c. Since the decoding algorithm is private, the vector $c+r$ can be sent publicly, because only A can decode this accurately to the codeword c (it is then easy to get the message m from the codeword c using the inverse of the encoding algorithm). Hence the security of the cryptographic scheme is as strong as it is difficult to decode a code given its encoding algorithm. For a general linear code this is known to be NP-hard, but in practice to have an efficient decoding algorithm we use special codes (the "tricks" mentioned above), which may admit special attacks.

To fix ideas, here is how the repetition code could be used to transmit a message: Suppose that the code chosen by A is the repetition code $\mathcal{C} = \{000, 111\}$, and B wants to send the

codeword 000 (which everyone agrees encodes the value 0). Then B chooses a small noise vector $r = 010$ at random, and sends the vector $000 + 010 = 010$. Our assumption then says that only A can efficiently decode this to the correct codeword 000. A then knows that the message was 0.

1.4 Multivariate cryptography

The security of multivariate cryptography relies on the difficulty of finding a solution for a general system of polynomial equations in n variables. This is known in general to be NP-hard, however in practice mathematicians use tricks to generate systems of equations which they can solve. These more structured instances of the problem might have special attacks.

The current schemes proposed for the NIST post-quantum cryptography competition both use quadratic polynomials (polynomials of degree less than or equal to 2), and are digital signature schemes. (2022 note: None of the multivariate ciphers proposed to NIST have been chosen for standardization or to go on to the fourth round of competition.) They both work in the following way: B wants to send messages and sign them to certify that they were really sent by B and no other person. B then publishes a public system of quadratic equations

$$\begin{aligned} x_1x_2 + x_2x_3 + x_1 + x_4 &= y_1 \\ x_2x_4 + x_3x_4 + x_1 + x_2 + x_3 &= y_2 \\ x_1x_3 + x_2x_3 + x_3x_4 + 1 &= y_3 \end{aligned}$$

as their public key. The private key is the ability to solve this system of equations for any values y_1, y_2, y_3.

Later on, if B wants to sign a document d, B first computes a hash of d of length 3, say $hash(d) = 111$. Then B forms the system of equations

$$\begin{aligned} x_1x_2 + x_2x_3 + x_1 + x_4 &= 1 \\ x_2x_4 + x_3x_4 + x_1 + x_2 + x_3 &= 1 \\ x_1x_3 + x_2x_3 + x_3x_4 + 1 &= 1 \end{aligned}$$

and solves it using their private key. The signature is a solution to the system. The recipient of the message can quickly check that the signature of the message is a solution to B's public set of equations with the hash of the message as vector of constants. The recipient is therefore certain that B sent the message since only they could solve this equation.

1.5 Hash-based signature schemes

These are a family of schemes to sign messages whose security relies on the security of cryptographic hash functions. In this case, we require the cryptographic hash function to be preimage resistant, by which we mean that given a string h, it is hard to find m such that

$h = hash(m)$. One of the algorithms which is to be standardized by NIST at this time for post-quantum cryptography is hash-based.

We illustrate how hash-based signature schemes work by giving the example of the Lamport signature scheme. This is a one-time signature scheme, which means that after signing one message the public key must be thrown out and never reused. The Lamport signature scheme can be coupled with a Merkle tree to allow a person to sign multiple messages with a single public key.

Suppose that we have a hash function that outputs strings of length 3. Then to sign messages, B would do the following: First B chooses 3 pairs of numbers

$$(a_1, b_1), (a_2, b_2), (a_3, b_3).$$

These pairs of numbers are B's secret key. B's public key is

$$(hash(a_1), hash(b_1)), (hash(a_2), hash(b_2)), (hash(a_3), hash(b_3)).$$

To sign a document d, B computes $hash(d)$. This has length 3, and suppose for clarity that $hash(d) = 011$. Then B's signature for the document d would be a_1, b_2, b_3. Here the rule is that B chooses the first element a_i of the pair (a_i, b_i) when the ith bit of the hash is 0 and the second element b_i of the pair if the ith bit of the has is 1.

To verify the signature, the recipient computes the hash of the message, and verifies that the values given by B do hash to the correct values given in B's public key. In this way, B has demonstrated knowledge of the secret key that generated the public key, and no one else could have provided these preimages.