Week 1

Nadav Kohen

June 22, 2025

Welcome! As you begin working through readings and exercises, if you find yourself uncomfortable with the notation and language around sets and functions, consider coming back to the following optional reading.

Reading 1 (Background)**.** *As needed, read about sets, set notations, functions and function notations in:*
Herstein - Abstract Algebra - pages 3-13.

Our goal this week is to introduce some of the basic constructions from public key cryptography that will be useful moving forward. In particular, we will focus on encryption to motivate our learning of some mathematical background, and then we will also discuss digital signatures in our synchronous time together.

As will be discussed in Reading 4, an asymmetric cipher consists of a public key function, $G : \mathcal{K}_{\text{priv}} \to \mathcal{K}_{\text{pub}}$, an encryption function, $E : \mathcal{M} \times \mathcal{K}_{\text{pub}} \to \mathcal{C}$, and a decryption function, $D : \mathcal{C} \times \mathcal{K}_{\text{priv}} \to \mathcal{M}$, such that $D(E(m, G(k)), k) = m$ (that is, decrypting an encrypting with the correct key yields the original message). In this definition, $\mathcal{M}$ denotes the set of possible messages, $\mathcal{K}_{\text{priv}}$ denotes the set of possible private keys, $\mathcal{K}_{\text{pub}}$ denotes the set of possible public keys, and C denotes the set of possible ciphertexts (a.k.a. encrypted messages). In order to make things more concrete and practical, we will want to replace these abstract

sets with specific finite sets. In particular, we will use the integers modulo a prime, $\mathbb{Z}/p\mathbb{Z}$ also denoted by $\mathbb{Z}_p$ or $\mathbb{F}_p$ depending on the author (or context). In general we will use p to denote a prime integer.

Reading 2 introduces $\mathbb{Z}/m\mathbb{Z}$ and its operations (without requiring that m be prime, as we will be assuming later). In particular, this reading introduces an efficient algorithm for modular exponentiation. This is important because we will be using the following public key function: Let $g \in \mathbb{Z}_p$ be a fixed non-zero number and define $G : \mathbb{Z}_p^* \to \mathbb{Z}_p$ by

$$G(x) = g^x \bmod p = g \cdot g \cdot \ldots \cdot g \bmod p \ (x \text{ times}).$$

We will discuss this further in Reading 5. The problem of trying to compute a private key for a public key with this G is known as the *discrete log problem.*

Reading 2. *Read about modular arithmetic and fast modular exponentiation in:*
Hoffstein, Pipher, Silverman - An Introduction to Mathematical Cryptography - pages 19-22, 24-26.

Exercise 1. *Implement the fast exponentiation algorithm in the language of your choice. This will allow us to compute public keys, and will also be useful in future exercises.*

As we work with numbers in $\mathbb{Z}_p$, it will often be important to compute "modular inverses." That is, if we have $a \in \mathbb{Z}_p$ that is not 0, then there exists a unique number $b \in \mathbb{Z}_p$ such that $ab \equiv 1 \pmod{p}$. We call b the modular inverse of a, and sometimes denote it by a^{-1}. For example, $2 \cdot 3 = 6 \equiv 1 \pmod{5}$ so that 2 and 3 are each others' modular inverses when $p = 5$. Reading 3 describes one method for computing modular inverses, and describes why our modulus, p, needs to be a prime number if we want to guarantee the existence of modular inverses. However, this reading is not necessary for our purposes, since we will describe an alternative method for computing modular inverses in Exercise 3. As such, the

following reading should not be your highest priority, but you can come back to it if you're ever curious!

Reading 3 (Optional Extra Background)**.** *Read about prime factorization, the Euclidean algorithm, and Bézout's identity in either of the following:*
Hamkins - Proof and the Art of Mathematics - pages 15-21.
Hoffstein, Pipher, Silverman - An Introduction to Mathematical Cryptography - pages 10-19.

The most fundamental result when working with modular exponentiation is *Fermat's little theorem.*

Exercise 2. *Find at least one proof of Fermat's little theorem online that you understand (there are many arguments out there). Be prepared to share!*

Since we will be working in $\mathbb{Z}_p$ where p is a prime number, we can leverage Fermat's little theorem, and the fact that if $ab \equiv ac \pmod{p}$ and $a \not\equiv 0 \pmod{p}$, then $b \equiv c \pmod{p}$, to come up with an alternative approach to computing a modular inverse that uses our code from Exercise 1.

Exercise 3. *Use Fermat's little theorem and fast modular exponentiation to come up with an alternative algorithm for computing the multiplicative inverse of an element in $\mathbb{Z}_p^*$. Implement this algorithm in the language of your choice.*

We now have all of the tools we need to introduce the Diffie-Hellman key exchange and the ElGamal cipher! All of the modular arithmetic we have covered so far will also be needed for our later discussion of Schnorr signatures.

We begin with a general discussion of symmetric and asymmetric ciphers in Reading 4.

Reading 4. *Read about symmetric ciphers, encoding messages, and asymmetric ciphers in: Hoffstein, Pipher, Silverman - An Introduction to Mathematical Cryptography - pages 36-47.*

In Reading 5, we look concretely at an asymmetric cipher build for the public key algorithm we have been studying (modular exponentiation). This reading ends by introducing abelian groups, which is the general context where the Diffie-Hellman key exchange and the ElGamal cipher can be implemented. You can think of abelian groups as a trait (or interface, if you prefer the language Java) for which these algorithms can be defined that will later allow us to switch out some of the number system we have been using for Elliptic Curves (which Bitcoin uses) later.

Reading 5. *Read about one-way functions, the discrete log problem for $\mathbb{Z}_p^*$, the Diffie-Hellman key exchange, the ElGamal cipher, and abelian groups in:*
Hoffstein, Pipher, Silverman - An Introduction to Mathematical Cryptography - pages 59-75.

Test your knowledge by utilizing your code from Exercises 1 and 3 to implement the ElGamal cipher!

Exercise 4. *Implement ElGamal for $\mathbb{Z}_p^*$ in the language of your choice.*

Exercise 5. *Once you have completed Exercise 4, describe how ElGamal can be implemented using an arbitrary abelian group. Don't forget to mention how public keys are computed. Explain why using $\mathbb{Z}_p$ with the operation of addition as your abelian group is insecure (Hint: recall Reading 2).*

If you wish to read about the details of elliptic curve computations, see Reading 6. However, knowledge of these details will not be necessary moving forward, since we will work at the abstract level of abelian groups and their operations to avoid entangling ourselves in any complicated EC details.

Reading 6 (Optional Extension)**.** *Read about the group of points on an Elliptic Curve in: Hoffstein, Pipher, Silverman - An Introduction to Mathematical Cryptography - pages 279-289.*