1 Overview of Course

Cryptography is about *communicating* and *computing* securely in the presence of *malicious* behavior. Its use goes back to antiquity, but only in the past few decades has it become a rigorous discipline. Central to the modern approach are precise mathematical *models* and strong *definitions* of security, as well as rigorous *proofs* based on well-formed (and often mild) assumptions.

Our inquiry will be centered around several core themes:

- **"Perfect" security.** The ideal and its limitations.
- **Computational hardness.** What it means for a problem to be "hard." The kinds of hardness needed for cryptographic applications. Where we might hope to find such hardness.
- **Indistinguishability and pseudorandomness.** How very different objects can appear "essentially the same," and how hardness can be used to achieve this. Applying these concepts to secret communication (encryption).
- **Authentication.** How to ensure that a message came from an expected source, not an impersonator. How to identify yourself remotely.
- **Interaction and knowledge.** Proving that a statement is true, without giving any reason why. Keeping a secret even if pieces of it are revealed. Running a program in public without revealing its inputs.
- **Advanced encryption.** Secrecy even given a decryption oracle. Your name as your public key.
- **Special topics.** Querying a database privately. Implications of quantum computers. Lattice-based cryptography. The frontier.

See the Course Information handout for policies and procedures.

2 Hidden Writing and Perfect Secrecy

Suppose a sender (call her Alice) wants to send a secret message to a receiver (Bob), but she doesn't want a potential eavesdropper (Eve) to be privy to its contents. Can this be achieved?

In addressing this question (and almost all others in this course!), we will apply the *cryptographic methodology*:

1. Form a realistic *model* of the problem (adjusting as necessary to allow for the possibility of a solution).
2. Next, *precisely define* the desired functionality and security properties of a potential solution.
3. Finally, *construct* and *analyze* a solution, (ideally) proving that it satisfies all the desired properties.

2.1 A First Attempt at a Model

In any scientific discipline, one of the earliest tasks in addressing a question is to form a precise *model* of the problem. This is one of the most important steps of the process, because all else depends on it: we need the model both to admit a useful solution to our problem, and to be as close to "reality" as possible (even though it will necessarily only be an approximation).

Let's try to define a model for the above "hidden writing" problem. In cryptography, we generally model all the potential actors in the system by *algorithms*. These algorithms have precisely defined interfaces, i.e., what inputs they take and what outputs they produce. In this case, we might choose the following model:

- The sender Alice is represented by an algorithm $A(\cdot)$ that takes as input a "plaintext" m from some (finite) set of possible messages $\mathcal{M}$, and outputs some "ciphertext" c from some (finite) set $\mathcal{C}$.
- The receiver Bob is an algorithm $B(\cdot)$ that takes as input a ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$.
- The eavesdropper Eve is some algorithm $E(\cdot)$ that takes as input (i.e., is allowed to see) a ciphertext c, and outputs... what, exactly? Because E is an adversary, we have little control over what it does, so let's not impose any constraint on the form of its output.

  Note, however, that we have specified (though somewhat imprecisely at this point) Eve's privileges in attacking the system: she gets to see a ciphertext c, and nothing else.

So far, so good. Notice that we have so far defined only the *interfaces* of the algorithms, but not any of the properties we would like them to have. One obvious property we'd want is that Bob should correctly recover (i.e., output) the message that Alice intended. More precisely: for every $m \in \mathcal{M}$, we should have $B(A(m)) = m$. (For jargon lovers, this is often called the "completeness" property.)

Our next task is to try to precisely define the desired *security* property we want our scheme to have. This is usually one of the most difficult and subtle tasks to get right, perhaps because it is a "negative goal:" we want to say that Eve *cannot* do something — but what, exactly? For the moment, we can certainly agree that even a minimally secure scheme should *at least* prevent Eve from always discovering Alice's message m. But a moment's thought reveals a problem: if the eavesdropper simply runs Bob's algorithm B (i.e., if $E = B$), then by the completeness property, E will *always* output the correct m. The problem is with our model — it is too strong to allow for a meaningful solution to the problem. We need to change it.

2.2 Fixing the Model

To avoid the problem described above, we need to introduce something that distinguishes Bob (and perhaps Alice as well) from Eve. One immediate idea is to make Bob's algorithm B *secret*, so that Eve cannot run it (often called "security by obscurity"). This turns out to be a *terrible idea*: the history of cryptography (and security in general) is littered with the discarded remains of "secret" algorithms/mechanisms that were anything but, or were broken even without discovering the mechanism at all. Furthermore, it is impossible to evaluate the security or effectiveness of an algorithm without knowing what it is! Therefore, a central tenet of modern cryptography is that

the system should be secure even if all its algorithms are public.

(This maxim is often called Kerckhoff's Law.) Of course, we need not go out of our way to disclose our algorithms to our enemies, but we should play it safe and assume that they will somehow learn what they are. And in practice, designing security mechanisms to be public often "keeps us honest" and ends up leading to better solutions.

OK, back to the problem at hand. Instead of using a secret algorithm, to distinguish Bob from Eve we use a secret *input*, typically called a "key" (by analogy to a lock mechanism that only the key can open). We augment our model above with an additional algorithm Gen that creates a key, which then becomes an

input to Alice and Bob, but not to Eve.[1] For more evocative notation, we also represent Alice by Enc (for "encrypt") and Bob by Dec (for "decrypt"). Our new model is as follows:

- Gen is a *randomized* algorithm that takes no input, and outputs a key k in some (finite) set $\mathcal{K}$.
- $\mathsf{Enc}_k(m) = \mathsf{Enc}(k, m)$ takes a key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$.
- $\mathsf{Dec}_k(c) = \mathsf{Dec}(k, c)$ takes a key $k \in \mathcal{K}$ and ciphertext $c \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$.

Notice that Gen *cannot* be deterministic (i.e., it must be able to "flip coins"), or else we would have gained nothing: Eve could just run Gen herself and learn the key. Also note that our model is still realistic, though somewhat less usable than before: Alice and Bob both need to get ahold of Gen's output, so they may need to meet in advance or have some trusted communication path to obtain the key without Eve intercepting it.

For completeness (pun not intended), let's update our correctness condition: for every $k \in \mathcal{K}$ and $m \in \mathcal{M}$, we should have $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

2.3 Shannon / Perfect Secrecy

Now we have a model that (hopefully) allows for a secure solution. But what does "secure" *mean*? We can imagine many desirable properties:

- Eve should not learn the key.
- Eve should not be able to output the message, given the ciphertext.
- The ciphertext should look like "random gibberish" without the key.
- ...

These all seem nice enough, but where does the list end? And how do we define them in a precise, mathematical way?

Let's go back and consider what we really want our scheme to accomplish: *it should conceal the message*. (The key and the ciphertext are only means to this end, so our security notion should not be "about" them.) Ideally, we would like the ciphertext to convey *nothing* to the adversary, i.e.,

seeing the ciphertext should be no better than *seeing nothing at all!*

This principle is a crucial insight that will appear again and again (in various guises) throughout our study of cryptography.

In his seminal 1949 work on information theory and cryptography, Claude Shannon precisely expressed the above principle in the language of probability theory, giving the following definition.

Definition 2.1 (Shannon secrecy)**.** A shared-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is *Shannon-secret with respect to a probability distribution D* over $\mathcal{M}$ if for all $\bar{m} \in \mathcal{M}$ and all $\bar{c} \in \mathcal{C}$,

$$\Pr_{m \leftarrow D,\, k \leftarrow \mathsf{Gen}}[m = \bar{m} \mid \mathsf{Enc}_k(m) = \bar{c}] = \Pr_{m \leftarrow D}[m = \bar{m}]. \tag{2.1}$$

The scheme is *Shannon-secret* if it is Shannon-secret with respect to every distribution D over $\mathcal{M}$.

[1] This is called the "shared-key" or "symmetric-key" model, for obvious reasons. Later in the course we will see other models that can be both more flexible to use, and allow for amazing functionality and security properties.

Let's consider this definition. First, the distribution D represents how Alice chooses her message, and can be arbitrary — but we (conservatively) imagine that the distribution itself is publicly known. The right-hand side of Equation (2.1) is simply the *a priori* probability that Alice chooses the message $\bar{m}$; this represents what Eve already knows about the message *without seeing the ciphertext*. The left-hand side is the *a posteriori* probability that Alice chose the message $\bar{m}$, *conditioned* on the fact that the ciphertext (which Eve gets to see) was $\bar{c}$. The definition says that the two probabilities are exactly the same, or in other words, that no matter the value of the ciphertext, it reveals nothing about the underlying message that was encrypted.

Despite its intuitive appeal, Shannon secrecy is often cumbersome to work with due to the arbitrary distribution D and the conditional probability expression. Here we give a simpler definition, which says that the ciphertext's distribution (solely over the random choice of the key) is exactly the same, no matter what message is encrypted.

Definition 2.2 (Perfect secrecy)**.** A shared-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is *perfectly secret* if for all $m_0, m_1 \in \mathcal{M}$ and all $\bar{c} \in \mathcal{C}$,

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m_0) = \bar{c}] = \Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m_1) = \bar{c}]. \tag{2.2}$$

Despite their very different syntactic forms, it is reasonably straightforward to prove (using elementary probability) that the two definitions are *equivalent*! That is, a scheme is Shannon-secret if and only if it is perfectly secret, so for all purposes we can use whichever definition we prefer. (We will not do the proof now, but a good one can be found in the Pass-shelat notes.) This robustness also gives us further confidence in the "rightness" of our approach and choice of definitions. Later on we will see other examples where seemingly very different security definitions end up being equivalent.

2.4 A Perfectly Secret Scheme: The One-Time Pad

Now that we have a model and a (pair of) good definition(s), it's time to move to the third step of our methodology: constructing and analyzing a scheme.

Interestingly, the encryption scheme we use predates Shannon's definition by 30 years! (Nowadays, it is less common for a scheme to end up proving secure according to a definition formulated later on, but those were simpler times.) Today the scheme is is called the *one-time pad*, or sometimes the *Vernam cipher* after its inventor. The intuition is that the key is used to completely "randomize" the message, by "shifting" each of its symbols by a random amount (independently of all the others).

Definition 2.3 (One-Time Pad)**.** Let $n \geq 1$ be an integer. The key, message, and ciphertext spaces are each the set of n-bit strings: $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$. The scheme is defined as follows:

- Gen outputs a uniformly random $k \leftarrow \{0,1\}^n$.
- $\mathsf{Enc}_k(m)$ outputs $c = m \oplus k \in \{0,1\}^n$, where $\oplus$ denotes the bitwise exclusive-or.
- $\mathsf{Dec}_k(c)$ outputs $m = c \oplus k \in \{0,1\}^n$.

Theorem 2.4. *The one-time pad is a perfectly secret shared-key encryption scheme.*

Proof. First, the one-time pad is well-defined as a shared-key encryption scheme: we have defined the sets $\mathcal{K}$, $\mathcal{M}$, $\mathcal{C}$, and the algorithms Gen, Enc, Dec in a manner consistent with our model.

We now prove completeness. Observe that for any $m \in \mathcal{M}$ and $k \in \mathcal{K}$, we have

$$\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0^n = m,$$

as required. (Here we have used standard facts about the $\oplus$ operation, such as associativity.)

Finally, we prove perfect secrecy according to Definition 2.2. Observe that for any $\bar{m} \in \mathcal{M}$ and $\bar{c} \in \mathcal{C}$, we have

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(\bar{m}) = \bar{c}] = \Pr_{k \leftarrow \{0,1\}^n}[\bar{m} \oplus k = \bar{c}] = \Pr_{k \leftarrow \{0,1\}^n}[k = \bar{m} \oplus \bar{c}] = 2^{-n}.$$

It follows that for any $m_0, m_1 \in \mathcal{M}$ and $\bar{c} \in \mathcal{C}$, we have

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m_0) = \bar{c}] = 2^{-n} = \Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}_k(m_1) = \bar{c}],$$

as required by the definition. This completes the proof. □