In CS 4820 we strive to teach you to distinguish between problems that have efficient algorithms and problems that are computationally hard, either conjecturally (in the case of NP-hard problems) or unconditionally (in the case of undecidable problems). Cryptography is an arena where both types of problems meet up, with fruitful consequences.

For thousands of years, people have attempted to use ciphers for secret communication. The history of cryptography has been a cat-and-mouse game between code makers coming up with creative methods of encoding messages, and code breakers coming up with even more creative ways of decoding them. The past century has been the most eventful one in the history of cryptography, with computers enabling new modes of secure communication that were previously unimaginable.

In order for encryption to be secure, the process of decrypting a message must be easy for a receiver who has the requisite secret key, but it must be hard for an attacker who lacks the key. But how should one interpret the terms "easy" and "hard" in the foregoing sentence? In these lectures we'll talk about two interpretations.

- **Statistical** notions of security formalize the notion that the attacker's inability to break the code stems from *randomness* in the process of generating the secret key and using it to encrypt messages, combined with the attacker's *lack of information* about the key.
- **Computational** notions of security formalize the notion that the attacker's inability to break the code stems from the *computational hardness* of the code-breaking task.

Statistical security guarantees are stronger than computational ones, since they persist even if the attacker applies unlimited computational resources to the challenge of breaking the code. On the other hand, as we'll see, achieving perfect statistical security is often impractical, which motivates the computational definition.

1 Shannon security and one-time pads

Claude Shannon, the founder of information theory, formalized one notion of statistical security as follows.

Definition 1.1. An encryption scheme with message set $\mathcal{M}$ is specified by giving a set of possible ciphertexts (C), a set of possible keys ($\mathcal{K}$), and three (possibly randomized) algorithms $\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$, such that:

1. Gen takes no input and outputs a key $k = \mathsf{Gen}()$;
2. $\mathsf{Enc} : \mathcal{K} \times \mathcal{M} \to C$ takes a key and message, and it outputs the encryption of the message, $c = \mathsf{Enc}(k, m)$.

3. $\mathsf{Dec} : \mathcal{K} \times C \to \mathcal{M}$ takes a key and ciphertext, and it outputs the decryption of the ciphertext, $m = \mathsf{Dec}(k, c)$.

4. The decryption operation with key k inverts the encryption operation with the same key.

$$\forall k \in \mathcal{K}\ \forall m \in \mathcal{M} \qquad \mathsf{Dec}(k,\ \mathsf{Enc}(k, m)) = m.$$

The encryption scheme is *Shannon secure* if the combination of key generation and encryption results in the same distribution over ciphertexts regardless of the message.

$$\forall m_0, m_1 \in \mathcal{M}\ \ \forall c \in C \qquad \Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}(k, m_0) = c] = \Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}(k, m_1) = c].$$

Shannon specified a simple construction that fulfills this security property: the *one-time pad*. One-time pad encryption schemes have the following characteristics.

1. The sets $\mathcal{M}, C, \mathcal{K}$ all have the same number of elements, N.

2. For any fixed $k \in \mathcal{K}$, the operation $m \mapsto \mathsf{Enc}(k, m)$ is a bijection between $\mathcal{M}$ and C, and $\mathsf{Dec}(k, \cdot)$ is the inverse of this bijection.

3. For any fixed $m \in \mathcal{M}$, the operation $k \mapsto \mathsf{Enc}(k, m)$ is a bijection between $\mathcal{K}$ nd C.

4. The key generation operation, Gen, samples k uniformly at random from $\mathcal{K}$.

For example, if $N = 2^n$ then we could take $\mathcal{M} = \mathcal{K} = C = \{0, 1\}^n$, and the encoding and decoding functions could be defined using the bitwise XOR operation:

$$\mathsf{Enc}(k, m) = \mathsf{Dec}(k, m) = k \oplus m.$$

Alternatively, for any N, we could take $\mathcal{M} = \mathcal{K} = C$ to be the set $\mathbb{Z}/(N)$ of integers modulo N, with encoding defined by addition and decoding defined by subtraction:

$$\mathsf{Enc}(k, m) = m + k, \qquad \mathsf{Dec}(k, m) = m - k.$$

Both of these would be considered one-time pad schemes. It is an exercise to verify that one-time pads satisfy Shannon security because the ciphertext is uniformly distributed over C, regardless of the message contents.

The Achilles' heel of Shannon's one-time pad is its enormous key size. Picture a spy satellite relaying terabytes of imagery to Earth. In order for the communication from the satellite to obey Shannon security, it would need to be launched into orbit with a hard drive containing terabytes of secret-key information. Worse yet, once the satellite had relayed an amount of data equal to the size of its secret key, the secrecy of its subsequent transmissions could not be assured. Unfortunately, encryption schemes that are Shannon secure must inevitably use large keys.

Lemma 1.1. *Any encryption scheme satisfying Shannon security must use a key set with at least as many elements as the message set.*

Proof. Consider some key k_0 such that Gen outputs k_0 with positive probability, and some ciphertext c in the range of $\mathsf{Enc}(k_0, \cdot)$. In other words, there is a message m_0 such that $\mathsf{Enc}(k_0, m_0) = c_0$. Then, by Shannon security, for every $m_1 \in \mathcal{M}$ we must have

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Enc}(k, m_1)] = c_0 > 0.$$

Since $\mathsf{Dec}(k, \mathsf{Enc}(k, m_1)) = m_1$, we also have

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathsf{Dec}(k, c_0)] = m_1 > 0.$$

Hence, for every $m_1 \in \mathcal{M}$ there exists some $k \in \mathcal{K}$ such that $\mathsf{Dec}(k, c_0) = m_1$. In other words, there is a surjection from $\mathcal{K}$ to $\mathcal{M}$ defined by $k \mapsto \mathsf{Dec}(k, c_0)$. The existence of a surjection from $\mathcal{K}$ to $\mathcal{M}$ implies that there are at least as many keys as messages. □

To sum up, in this section we've learned two important lessons about secure communication.

- If two parties share an n-bit secret key that they can use as a one-time pad, then one of them can send an n-bit message to the other with ***perfect secrecy***: an attacker with no information about the secret key can ***learn nothing whatsoever*** about the message, even if they intercept the ciphertext and expend unlimited computational resources in trying to decrypt it.
- Conversely, in order to achieve such a strong information-theoretic guarantee of secrecy, it is ***necessary*** to share a secret whose size (in bits) is as large as the message itself. This motivates the study of cryptographic schemes that are secure against weaker, computationally bounded, attackers.

2 One-way functions

A one-way function is a function that is easy to compute but hard to invert. This turns out to be extremely useful for building cryptographic schemes that are secure against computationally bounded attackers. In this section we'll present the basic definition of a one-way function, and we'll present a construction of a family of functions that are believed to satisfy the definition. In later sections we'll see how to use this family of functions to accomplish some cryptographic goals that sound miraculous when first encountered.

- *Key exchange:* Two parties, Alice and Bob, exchanging messages "in the clear" on a public channel visible to attackers, can agree on a shared secret key without (computationally bounded) attackers gaining knowledge of the key.
- *Public-key cryptography:* Alice can publicize an encoding function Enc to be used for sending messages to her. Using a secret key Alice can decode messages encoded using Enc. A (computationally bounded) attacker who doesn't know the secret key gains no information about the messages being sent to Alice.

Definition 2.1. A *one-way function* is a function $f : \{0,1\}^* \to \{0,1\}^*$ satisfying the following properties.

1. f **is easy to compute:** There is a (potentially randomized) algorithm to compute $f(x)$ running in time poly($|x|$), where $|x|$ denotes the number of bits in x.
2. f **is hard to invert:** For any attacker using a (potentially randomized) polynomial-time algorithm $\mathcal{A}$ and any constant $c < \infty$, if:
   - x is sampled uniformly at random from $\{0,1\}^n$;
   - $f(x)$ is computed and sent to the attacker;
   - the attacker attempts to invert f by calculating $x' = \mathcal{A}(0^n, f(x))$;

   then $\Pr[f(x') = f(x)] < 1/n^c$ for all sufficiently large n.

The hard-to-invert property asserts that an attacker who attempts to invert f using a randomized polynomial-time algorithm has only negligible probability of succeeding. Here, the definition of "negligible" is that the probability of success, as a function of the message length x, tends to zero faster than any inverse-polynomial function of n.

There are a couple of things that seem strange about the second part of the definition when one first encounters it.

- *Why focus on* $\Pr[f(x') = f(x)]$ *rather than* $\Pr[x' = x]$*?* The answer is that if the purpose of the one-way function were only to conceal the value of x, a constant function $f(x) = 0$ would trivially do that.
- *Why does the attacker compute a two-variable function* $\mathcal{A}(0^n, f(x))$ *rather than a one-variable function* $\mathcal{A}(f(x))$*?* The unary input 0^n is an artificial construct that just ensures the attacker's algorithm is allowed to run in poly(n) time, even if $|f(x)| \ll n$.

Before presenting an example of a (conjecturally) one-way function, we offer the following remarks.

1. If $\mathsf{P} = \mathsf{NP}$ then every function that is easy to compute is also easy to invert. Given the value $f(x)$ for some $x \in \{0,1\}^n$, the search for an x' such that $f(x') = f(x)$ reduces to solving a sequence of $O(n)$ decision problems, each of the form, "Here is a string $z \in \{0,1\}^k$ for some $k < n$. Does there exist a string $x' \in \{0,1\}^n$ such that $f(x') = f(x)$ and the first k bits of x' match with z?" Each of these is an NP decision problem, so if $\mathsf{P} = \mathsf{NP}$ then we have a polynomial-time algorithm to invert f.
2. Since we don't yet know how to prove $\mathsf{P} \neq \mathsf{NP}$, constructing a *provably* one-way function is way beyond the reach of current techniques.
3. The next-best thing would be a construction that provably satisfies the definition of one-way function under the assumption that $\mathsf{P} \neq \mathsf{NP}$. We still don't know any such construction, but progress in the past five years has made some cryptographers optimistic that the long-sought goal of basing cryptography on NP-hard functions may soon be within reach.

Example 2.1 (Factoring)**.** Our first example of a one-way function could be more accurately classified as an "instructive non-example." Consider the multiplication function, $m(a, b)$, which takes two integers $a, b > 1$ each having, say, $n/2$ binary digits, outputs their product $m(a, b) = a \cdot b$. This function is easy to compute. In fact, earlier in this course you learned an algorithm, based on the Fast Fourier Transform, that computes the product of two $n/2$-bit integers in time $O(n \log^2 n)$. The fastest known multiplication algorithm improves this running time to $O(n \log n)$.

On the other hand, inverting the multiplication function is tantamount to finding a non-trivial factorization of a given composite number. The fastest known integer factorization algorithm on a classical computer requires $2^{O(n^{1/3})}$ time to factor an n-bit integer. (Peter Shor famously found an integer factorization algorithm that runs in polynomial time on a quantum computer. At present, quantum computing technology has not progressed to the point where Shor's algorithm could be run on inputs of non-trivial size.)

Thus, given our present state of knowledge, multiplication is a function that is easy to compute but hard to invert. However, it doesn't satisfy Definition 2.1 because easy-to-factor integers are too plentiful: if we sample $x = (a, b)$ at random and compute $f(x) = m(a, b) = a \cdot b$, it's often easy to find an x' such that $f(x') = f(x)$. In order for Definition 2.1 to be satisfied, we need the probability of finding such an x' to be negligible. The situation is summarized by saying that multiplication is (believed to be) a *weakly one-way function*, but not a one-way function.

Example 2.2 (discrete logarithm)**.** Our next example of a one-way function is based on modular exponentiation. Given positive integers g and N, the exponential function $f(x) = g^x \pmod N$ is easy to compute. Letting $\ell = \lfloor \log_2 x \rfloor$ one first computes $g^1, g^2, g^4, g^8, \ldots, g^{2^\ell} \pmod N$ by repeated squaring. Then, using the binary representation of x to write it as a sum of at most ℓ powers of 2 in the range $\{1, 2, \ldots, 2^\ell\}$, we see that g^x is a product of at most ℓ numbers in the set $\{g^1, g^2, \ldots, g^{2^\ell}\}$. Thus, computing $g^x \pmod N$ reduces to at most 2ℓ instances of integer multiplication, each of which can be computed in near-linear time as noted earlier.

The inverse of the function $f(x) = g^x \pmod N$ is called the *discrete logarithm function* (with base g and modulus N) and is believed to be a genuine one-way function when N is prime and g is a so-called *primitive root*, a number whose first $N - 1$ powers are all distinct, mod N.

2.1 Sophie Germain primes and the discrete logarithm

To continue working with the discrete logarithm, it will be useful to refer to the following definition.

Definition 2.2. The *order* of g modulo N is the smallest positive exponent d such that $g^d \equiv 1 \pmod N$.

Observation 2.1. If d is the order of g modulo N then the powers of $g \pmod N$ form a periodic sequence with period length d. To see that this is the case, observe that the identity $g^{d+x} \equiv g^x \pmod N$ holds, for all $x \geq 0$, by induction on x. The base case $x = 0$ is Definition 2.2, and for the induction step one assumes $g^{d+k} \equiv g^k \pmod N$ and multiplies both sides of the congreuence by g to conclude $g^{d+k+1} \equiv g^{k+1} \pmod N$.

In constructing key exchange protocols and public-key cryptosystems, we will be working with the discrete logarithm problem when N is a large prime number, and the order of g modulo N is also a large prime number. Fortunately, there is a plentiful source of examples.

A *Sophie Germain prime* is a prime number p such that $q = 2p + 1$ is also prime. While there is no known proof, at present, that the set of Sophie Germain primes is infinite, it is conjectured that a random n-bit integer is a Sophie Germain prime with probability at least c/n^2, for some constant $c > 0$ not depending on n. The numerical evidence based on small values of n is consistent with this conjecture.

Lemma 2.1. *When p is a Sophie Germain prime and $q = 2p + 1$, then the order of 4 modulo q equals p.*

Proof. We'll start by proving $4^p \equiv 1 \pmod{q}$. Observe that $4^p = 2^{2p} = 2^{q-1}$. Using the binomial theorem we have

$$2^{q-1} = \frac{1}{2} \cdot 2^q = \frac{1}{2} \sum_{i=0}^{q} \binom{q}{i} = \sum_{i=0}^{p} \binom{q}{i},$$

where the final equation follows because of the identity $\binom{q}{i} = \binom{q}{q-i}$, which reveals that the last $p + 1$ terms of the sum are equal to the first $p + 1$ terms in reverse order. Now, from the formula $\binom{q}{i} = \frac{q!}{i!(q-i)!}$ and the fact that q is prime, we see that $\binom{q}{i} \equiv 0 \pmod{q}$ except when $i \in \{0, q\}$. Hence, only the first term of the sum on the right is non-zero, and $4^p = 2^{q-1} \equiv \binom{q}{0} = 1 \pmod{q}$, as claimed.

Now, let d denote the order of 4 modulo q. We claim, by induction on $k > 0$, that $4^k \equiv 1 \pmod{q}$ if and only if k is divisible by d. The base cases $1 \leq k \leq d$ follow from the definition of d, which requires that $4^d \equiv 1 \pmod{q}$ but that $4^k \not\equiv 1 \pmod{q}$ for $1 \leq k < d$. For the induction step, suppose $k > d$ and use the periodicity relation $4^k \equiv 4^{k-d} \pmod{q}$. By the induction hypothesis, $4^{k-d} \equiv 1 \pmod{q}$ if and only if $k - d$ is divisible by d, which happens if and only if k is divisible by d.

Since we have previously shown that $4^p \equiv 1 \pmod{q}$, we may conclude that the order of 4 modulo q is a divisor of p. But p is prime, so the order of 4 modulo q is either 1 or p. Since $4^1 \not\equiv 1 \pmod{q}$, we know the order of 4 cannot be 1, so it must equal p as claimed. □

3 Diffie-Hellman Key Exchange

Recall the one-time pad encryption scheme of Section 1. If two communicating parties, Alice and Bob, can agree on a shared n-bit secret key without an attacker, Eve, gaining any information about the key, then Alice and Bob can use the secret key as a one-time pad to communicate n bits to one another in perfect secrecy.

Suppose now that Alice and Bob are not able to meet in private to agree on their secret key. Instead, all of their communication must take place on a public channel, with Eve observing every bit of information they send to one another. In this section we will present a protocol by which they can reach agreement on a secret key, yet in order to gain information about their secret key an attacker would have to be able to solve (a lightly strengthened version of) the discrete logarithm problem.

Here is the protocol. Let q be a prime number, and let g be a positive integer such that the order of g modulo q is another prime number, p. If p is a Sophie Germain prime and $q = 2p + 1$, then we have seen that we can take $g = 4$ for example. Finally, let $[p]$ denote the set $\{0, 1, \ldots, p - 1\}$.

1. Alice generates secret key $a \in [p]$, uniformly at random, and sends $A = g^a \pmod q$ to Bob.
2. Bob generates secret key $b \in [p]$, uniformly at random, and sends $B = g^b \pmod q$ to Alice.
3. They both compute the shared secret key $k = g^{ab} \pmod q$.
    - Alice computes k using the formula $k = B^a \pmod q$. This is possible because she knows a, and she learned B from Bob.
    - Bob computes k using the formula $k = A^b \pmod q$. This is possible because he knows b, and he learned A from Alice.

Once Alice and Bob have shared key k, they can use it as a one-time pad in an encryption scheme with

$$\mathcal{M} = \mathcal{C} = \mathcal{K} = \{\text{powers of } g \pmod q\}.$$

Note that the size of the message set is p, by our assumption that g has order p modulo q. Thus, at the cost of performing a set-up phase that involves sending $2\lceil \log_2 q \rceil$ bits, they are able to communicate a secret message of length $\lfloor \log_2 p \rfloor$. When $q = 2p + 1$, this means the communication cost of the set-up phase exceeds the number of message bits transmitted by a factor of 2, plus a small constant number of additional bits.

We have described how to implement the $\mathsf{Gen}()$ functionality of the encryption scheme using communication over a public channel. The encoding function is $\mathsf{Enc}(k, m) = k \cdot m \pmod q$, and the decoding function is $\mathsf{Dec}(k, c) = k^{p-1} \cdot c \pmod q$. Decoding with key k inverts encoding with key k, because if we write $k = g^j$ for some non-negative integer j then

$$\mathsf{Dec}(k, \mathsf{Enc}(k, m)) \equiv k^{p-1} \cdot k \cdot m \equiv k^p \cdot m \equiv (g^j)^p \cdot m \equiv (g^p)^j \cdot m \equiv m \pmod q.$$

How do we know this is computationally secure? We don't. If one assumes modular exponentiation is a one-way function then a computationally bounded eavesdropper, Eve, would have only negligible probability of recovering Alice's or Bob's secret key, a or b. It is difficult to picture an algorithm for computing $g^{ab} \pmod q$ given the values of g^a and $g^b \pmod q$, without knowing one of $\{a, b\}$. However, that doesn't mean we know how to prove the security of the Diffie-Hellman key exchange protocol from the assumption that the discrete logarithm is computationally hard. Instead, the assumption that underlies the security of Diffie-Hellman key exchange is called the *decisional Diffie-Hellman (DDH) assumption*. It concerns the problem of distinguishing ordered triples of the form $(g^a, g^b, g^{ab}) \pmod q$ from ordered triples of the form $(g^a, g^b, g^c) \pmod q$, when q is a large prime and the order of g modulo q is another large prime p. If a, b, c are sampled uniformly at random from $[p]$, the DDH assumption is that for any efficiently computable test τ, the probabilities $\Pr[\tau(g^a, g^b, g^{ab}) = 1]$ and $\Pr[\tau(g^a, g^b, g^c) = 1]$ differ by less than $\varepsilon(n)$, where $n = \log_2(p)$ and $\varepsilon(n)$ tends to zero faster than any inverse-polynomial function of n. In other words, the DDH assumption is basically just a way of rephrasing the assertion that Diffie-Hellman key exchange is computationally secure.