Cryptography

The field of *cryptography* traces its roots back to classical times. The core premise is simple: How can we protect communications that may be intercepted by an adversary? Can we ensure that even if somebody is able to see our message, they cannot to understand it? In the modern era, this is achieved through mechanisms which take advantage of advanced mathematical theory, including number theory, elliptic curves, and lattices. However, earlier approaches didn't have the benefits of such principles, and relied instead on techniques such as scrambling (e.g., encoding "Hello World" as "lelHo orWld") or substitutions. The most famous example of the latter is the *Caesar cipher*, named after Julius Caesar who is reported to have incorporated it in his private correspondence. The Caesar cipher is a shift cipher, where every letter is replaced by the letter that occurs three positions later in the alphabet (with letters near the end wrapping around to the beginning).

Plaintext	A	B	C	D	...	W	X	Y	Z
Ciphertext	D	E	F	G		Z	A	B	C

(1)

Following the mechanism described by the above table, the message "Hello World" would be encoded "Khoor Zruog." More generally, *substitution ciphers* replace each occurrence of a particular character with a new character, with no two distinct characters mapping to the same replacement. While substitution ciphers have some advantages over scrambling (they are invertible so messages are unambiguous, and this inversion can be done quickly), they can be broken by modern computers in a matter of seconds.

In the context of computational complexity, we see that the problem of decoding an encrypted message, under the assumption that a substitution cipher has been used, is in NP.

Theorem 1. *The following problem is in* NP*:*

$$\mathsf{SUB} = \{w \mid w \text{ is the result of applying a substitution cipher to an English sentence}\}$$

Proof. A valid certificate would be the function that describes the substitution. This has constant length (it depends only on the alphabet, not the input w), which a verifier could confirm by inverting the substitution and applying this transformation to w, then checking that the result is a valid sentence. This last step is left a bit ambiguous, but could involve something like checking that each word appears in a dictionary and parsing the sentence structure as an element of a fixed grammar. The runtime of such an algorithm is $O(|w|)$. □

A similar proof shows that this is also the case for block substitution ciphers, which can substitute multiple characters at a time. It turns out that the problem of decoding a message which is known to be encrypted using a block substitution cipher is NP-complete.

There is a more general parallel between the treatment of problems in computational complexity and cryptography. Of particular interest to cryptographers are functions which are difficult to solve in general, but can be solved easily when you are provided a small amount of additional information. Sound familiar? Consider the following specialization of the above statement:

> ... **decision problems** which are difficult to solve in general, but can be solved **in polynomial time** when you are provided a **polynomial** amount of additional information.

We recover exactly the definition of NP! This helps explain some of the significance of complexity-theoretic results in cryptography: if $\mathsf{P} \neq \mathsf{NP}$, then there exist "efficient secure cryptographic protocols"—that is, problems which are easy to solve with additional information (so they are in NP) but not on their own (so they are not in P).

Another relationship involves a probabilistic complexity classes we have encountered, BPP. If $\mathsf{P} = \mathsf{BPP}$, then, loosely speaking, we can translate any efficient probabilistic algorithm into an efficient deterministic algorithm. In practice, this is done by using a tool known as a pseudorandom number generator (PRNG), which takes a small number of random bits and extends them to generate a distribution which looks "close to random" on a much larger number of bits. Then rather than sampling over a large distribution, we can brute force over all possible inputs to the PRNG and sample over the results, to obtain an approximation to the actual result.

Many cryptographic results are based on the assumption that certain PRNGs work. Indeed, there are a wide variety of PRNGs that, under the appropriate assumptions, suffice for different problems. We construct different PRNGs, because there may be different features of the resulting "approximately uniform" random distribution we wish to emphasize, depending upon the problem. One way to show that $\mathsf{P} = \mathsf{BPP}$ would be to create a single general-purpose PRNG that works for every problem (it turns out that these are actually equivalent). On the other hand, a proof that $\mathsf{P} \neq \mathsf{BPP}$ would show that there is some problem for which no PRNG is good enough; that is, there is no efficient way to simulate the necessary randomness. This would in turn collapse the assumptions that the aforementioned results rely upon.

Much of the work in modern cryptography does require more advanced mathematics than we have covered in the course. We focus our attention to a few specific examples that work well with the machinery we have already established.

Zero-Knowledge Proofs

Suppose you are a corporate spy and claim to have knowledge of some secret recipe (say, for a particular brand of soda, or a spice mix for fried chicken). To reap the benefits of this knowledge, you offer to sell the recipe to a top competitor, for a hefty price of course. Naturally, your buyer is cautious. Before any money changes hands, they want a guarantee that you're the *real deal*, and that you actually have the information that you claim to.

One approach that would lend credence to your argument would be to produce a sample of the product which you've synthesized using your knowledge of the recipe. "Well wait a second," interjects your potential buyer, "How do I know you didn't just buy one of the genuine articles and remove the label? Then your product certainly would taste like the real thing—because it is! But that doesn't mean you know the recipe." You have to admit, it's a compelling argument. How could you provide better evidence? "How about I watch you cook a batch?" offers the buyer.

At first, this seems like a reasonable request—it would certainly convince the buyer that you possess the knowledge you claim. But there's a problem here: if they watch you, then they can learn the recipe for free! So you clearly can't do this, as you'd be revealing too much information. You could try to minimize the amount of information that you leak, perhaps by using unmarked containers. But this would still tell an observer *some* information, such as the number of different ingredients used. Is there a way to convince the buyer that you actually do know the recipe without leaking *any* information at all?

Surprisingly, for some problems, the answer is "yes!" Such a method is called a *zero-knowledge*

proof, as you prove that the claim holds, without providing any additional knowledge.

Recall from the textbook the following definition of the complexity class IP (*Interactive Proof*):

Definition 1. *A language A is in* IP *if there is a polynomial-time computable function V such that for some function P and for every function $\widetilde{P}$, the following conditions hold for all $w \in \Sigma^*$:*

1. *$w \in A$ implies that* $\Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$.
2. *$w \notin A$ implies that* $\Pr[V \leftrightarrow \widetilde{P} \text{ accepts } w] \leq \frac{1}{3}$.

Recall that V is allowed to be probabilistic; in this definition, this property is captured by allowing it to take in another input string r that tells it which decisions to make at every coin-flip step. Also remember that, although this definition is given in a "one-off" format, the intent is to run multiple independent iterations to improve the error bounds using the amplification lemma. A slight modification of this definition additionally captures the zero-knowledge property:

Definition 2. *A language A is in* ZK *(*zero-knowledge*) if there is a polynomial-time computable function V such that for some function P and for every function $\widetilde{P}$, the following conditions hold for all $w \in \Sigma^*$:*

- ***Completeness:*** *$w \in A$ implies that* $\Pr[V \leftrightarrow P \text{ accepts } w] = 1$. *In other words, a verifer can always be convinced of the veracity of a true statement by an honest prover.*
- ***Soundness:*** *$w \notin A$ implies that* $\Pr[V \leftrightarrow \widetilde{P} \text{ accepts } w] \leq 1 - \frac{1}{|w|^c}$ *for a constant c. This means that there is only a "low" probability that the verifier can be tricked into believing a false statement is true.*
- ***Zero-Knowledge:*** *No verifier learns any additional information from the interaction, except that the statement is true (and any logical consequences of this fact).*

The first two conditions yield a modified version of correctness, in an analogous way to how BPP is modified to obtain RP. The zero-knowledge condition can be a little tricky to formalize properly. One way to do this is to say that any verifier has a corresponding *simulator* (another polynomial-time PTM) that, using only the information available to the verifier, could produce an identical transcript of the interaction between the prover and the verifier on any w. Intuitively, this simulator has "no more knowledge than the verifier except for w's membership in A," and can re-create the same end result of the interaction on its own. Also note that, although the error probability may be very close to 1 (and even increasingly close to 1 as $|w| \to \infty$), we can use the amplification lemma to improve the error rate to any constant ε using only polynomially many iterations.

Theorem 2. *3-*COLOR *is in* ZK.

Proof. Consider the following zero-knowledge protocol. Given a 3-colorable graph G, the prover P first assigns it a valid 3-coloring (say, with the colors cyan, magenta, and yellow). P then randomly picks a function that replaces each of these colors with one of red, blue, and green such that each original color becomes a distinct new color. P relabels the vertices with these updated colors to get a new valid 3-coloring and outputs an encrypted version of the new coloring. This is done using a *commitment scheme*, i.e., a coded message that incorporates the details in a way that is difficult to decode in general, but can be selectively decrypted in independent segments using additional information. In this case, you can think of the prover writing every vertex and its color

on opposite sides of individual slips of paper, and putting them all on a table with the vertex-side up. The verifier V can then randomly pick an edge in G and ask the prover to reveal the colors of its endpoints, which it does by flipping over the pieces of paper for these two vertices. Then V accepts if and only if these colors are different.

If G is truly 3-colorable, then there is a prover that can come up with a viable 3-coloring, and any edge that the verifier asks to check will have different-colored endpoints. Thus, V accepts with probability 1. On the other hand, if G is not 3-colorable, any potential prover that tries to 3-color G must have at least one edge whose endpoints are the same color. The verifier has at least a $\frac{1}{|E|}$ chance of choosing this edge, so soundness holds with an error rate of at most $1 - \frac{1}{|E|}$. Finally, there is no leakage of information—the only thing that the verifier learns is that a particular edge has two endpoints of different colors, which must be true if G was 3-colorable! Another way to think about this is that a simulator could fabricate an answer to V's query by just naming 2 different colors as the colors of the endpoints without any further knowledge of the actual solution.

The runtime of V is the time it takes to pick a random edge (in $O(|E|)$ time) and check that two colors are different, which is $O(1)$ time. The overall runtime is $O(|E|)$. □

The role of the commitment scheme is to ensure that the prover can't cheat by modifying their answer in response to the verifier's query. Without it, a dishonest prover could always say "the endpoints are red and blue," even if the graph is not actually 3-colorable, and the verifier would be tricked into believing them. With the commitment scheme, the colors must be assigned *before* the verifier picks an edge (though they are revealed until later, or not at all).

Note that the choice of information that we allow the verifier to ask about is quite important! For example, we could have instead allowed the verifier to ask "show me all of the red vertices." In this case, the verifier is still likely to reject non 3-colorable instances, since with probability at least $\frac{1}{3}$ there will be an edge with two red endpoints. However, this violates zero-knowledge by leaking additional information: in the case that the prover did commit to a valid 3-coloring, the verifier now knows that there is a 3-coloring with this particular set of red vertices, which they could use to construct the rest of the 3-coloring in polynomial time (2-COLOR is in P). Similarly, we can't allow them to ask about an arbitrary pair of vertices, as this may reveal the fact that "u and v have the same color in a valid 3-coloring," even though this piece of information doesn't seem that useful—for zero-knowledge to hold, there cannot be *any* leakage.

The prover randomly permuting the colors is just as important. If not, the verifier could use information from multiple rounds of questioning to learn about the solution. For example, queries that reveal that u is red and v is green, and also that u is red and w is green would then imply that v and w are both green! The re-randomization in each round ensures that each run of the protocol doesn't leak information that the verifier could abuse if remembered in future rounds.

Going back to the recipe scenario, what would an actual zero-knowledge protocol look like? One possibility is as follows: you pre-mix a variety of ingredients and bring them in n unmarked containers. If you like, you can even bring mixtures that are not used in the recipe at all. The potential buyer is allowed to randomly pick any container and test it to see if it contains their competitor's product[1]. Then, you complete the recipe unobserved, in a room which has previously been screened to guarantee that there's no hidden ingredients stashed anywhere, after which you present your sample. Under reasonable assumptions about your buyer's palate, this constitutes a zero-knowledge protocol.

[1]Let's say that this is done by a simple taste test, and not something like mass spectrometry, which could reveal a lot more information.

Differential Privacy

A similar problem aims to answer "How can we obtain aggregate statistics concerning a large data set without exposing the data of a particular member?" For example, a hospital may wish to accrue data to construct a budget for future expenditures, or inform future research investment opportunities. But the hospital is bound by HIPAA, which prevents it from revealing data about specific patients[2].

At a minimum, data should be stored in a way that anonymizes its source. But it turns out that this is insufficient to protect user information. In 2006, Netflix (at the time, still a DVD mailing service) held a contest for the best algorithm to predict user ratings. The scope of the data was quite restricted—the training data was released as a list of tuples of the form `<userID, movie, date of grade, grade>`. The algorithms would then have to propose estimates of grades on the test set, given only the values of the first three fields of the tuple. Originally, the contest was intended to find a methodology that improved upon Netflix's own Cinematch, which had stalled by that point. Within a week, submissions had already improved upon the baseline. In 2009, the initial goal of improving the algorithm by 10% was achieved, and the contest was closed.

Despite the relatively pared-down nature of the information distributed in the dataset, two researchers were able to effectively de-anonymize the user data. That is, given the training data and a small amount of outside information about a particular individual, they could determine the user ID of the individual in the anonymized dataset, despite the fact that this dataset was constructed in such a way as to prevent revealing user identities! In fact, their method works with high probability even if (1) the available information is approximate, e.g., the date is a two-week window, (2) only a small subset of the dataset is released, and (3) some of the data is incorrect. In this case, the additional information came largely through publicly available IMDb user data, which provides a correlated set of ratings submitted around similar times. In some sense, this hinged on the assumption that if you felt strongly enough about a movie to give it a rating on IMDb, you probably did so on Netflix as well.

Differential privacy more precisely refers to the notion that the inclusion or exclusion of a single person's data should not greatly change the distribution of any accessible statistics. This also works in the reverse direction, in that available statistics should not be fine-grained enough to identify a specific person. For example, if a hospital stores health records and makes the counts of various blood types easily accessible, then one could potentially identify a specific person's blood type by querying these counts on a dataset of all the patients, and one excluding this specific patient, *even if* there is no direct identifying information publicly available.

Most differential privacy mechanisms work by injecting a small amount of random noise into the data, so that any queries are randomly distributed around the true value. Intuitively, this noise should be constructed so that the relevant statistics will be indistinguishable in distribution from the same dataset with a single person added or removed. This does mean that proving pertinent results generally requires more statistics than we have covered in CSC-341 (or its prerequisites).

We can consider a relatively simple problem, which proceeds as follows: A class of n students wishes to determine the average grade on an exam. However, nobody wants to reveal their grade to any other student! One solution would be to appeal to a trusted third-party, e.g., a mutual friend who is not in the class. However, such a situation can be difficult to implement (maybe not all of the students share common friends) and problematic in practice (what happens if the friend goes rogue

[2]Even if this were not the case, releasing patient data would certainly be ill-advised!

and starts publishing people's grades?). Luckily, it turns out that the students can collaboratively construct a differentially private mechanism using only messages between the participants.

Example 1. *There is a differentially private mechanism such that (1) every student learns the true average, and (2) no student can learn any other student's score.*

Proof. The students number themselves 1 to n; let Student i's score be denoted s_i. They proceeds as follows: First, Student 1 selects a random number r, using a method *that they keep secret*. The selection mechanism is not important, so long as nobody else knows it: the strategies "Always pick 7" and "Sample a Gaussian" both work equally well. They then tell Student 2 the value of $r + s_1$. Student 2 adds their score and tells Student 3 the value of $r + s_1 + s_2$. This proceeds until all n students have gone (Student n tells their sum to Student 1). Student 1 then subtracts r from this total and announces the remaining value to the group. This value is exactly $s_1 + s_2 + \cdots + s_n$, the sum of scores, from which the average can be obtained by dividing by n.

Note that no single student can ever learn enough information to extract the specific value of another student's score. They will always receive a noisy signal (due to Student 1's choice of r) or the aggregate value of all of the scores. □

This approach does have a few vulnerabilities involving collusion and dishonest participants. You will explore these in the lab exercise. We can improve this protocol somewhat, through a slightly more complicated procedure.

Theorem 3. *There is a differentially private mechanism such that (1) every student learns the true average, and (2) no student's grade can be determined by* any *subset of $n - 2$ other students, even if they pool information.*

Proof. Note that $n - 2$ is truly the best bound possible: if $n - 1$ students all pool information and the true class average is known, then there is no way to stop them from figuring out the last student's score.

First, each Student i comes up with a secret list of n numbers $x_{i,1}, x_{i,2}, \ldots, x_{i,n}$ such that $\sum_{j=1}^{n} x_{i,j} = s_i$, through a mechanism which they also keep secret. Once again, the specific mechanism used does not matter as long as nobody else knows it. A straightforward way to ensure the sum is s_i is to pick $n - 1$ numbers using any method, and then setting $x_{i,n} = s_i - \sum_{j<n} x_{i,j}$. Just as above, "always pick 7" and "repeatedly sample from a Gaussian distribution and subtract the previous number squared" both work equally well for selecting the first $n - 1$ numbers, *so long as* nobody else knows how the numbers are chosen. The protocol then proceeds in three stages.

In the first stage, there are n rounds. In the jth such round, for all $1 \leq i \leq n$, Student i tells the number $x_{i,j}$ to Student j. In other words, every student distributes their list almost entirely across the other students, but only tells each student one number.[3]

In the second stage, each Student j first sums all of the numbers they have been told and adds their own $x_{j,j}$ to obtain an "obfuscated score." The value of Student j's obfuscated score is $\sum_{i=1}^{n} x_{i,j}$ (note the index of summation).

In the third stage, everybody sends their obfuscated scores to Student 1, who sums them. This is the total sum of scores, which can be divided by n to obtain the average score.

[3] In practice, this step is more efficiently implemented by distributing communication more evenly among pairs at each timestep, but we still end up with a poly-time algorithm.

The shared information in an execution for $n = 4$ is displayed in a table to help illustrate how the mechanism works (Figure 1). The lists will be 1-indexed to make the student numbers line up with the round numbers more nicely.

	$\mathbf{j = 1}$	$\mathbf{j = 2}$	$\mathbf{j = 3}$	$\mathbf{j = 4}$
$\mathbf{i = 1}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$\mathbf{i = 2}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$\mathbf{i = 3}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$\mathbf{i = 4}$	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$

Figure 1: All of the information that changes hands in Stage 1. The (i, j)-th entry corresponds to the value which Student i tells Student j.

Note that every student's true score can be obtained by summing across their row. Additionally, every student's column contains all the numbers which they have been told, the sum of which is exactly the value they compute in Stage 2. Adding all of the column sums, as is done in Stage 3, will yield the same value as adding all of the row sums, which is exactly the desired sum of scores.

In this scenario, collusion by any set of students excluding Student i and another Student j, even if they pool all of their information, is insufficent to determine s_i. Observe that there are four values which are known to only these i and j: $\{x_{i,i}, x_{j,j}, x_{i,j}, x_{j,i}\}$. The only information observable by other students that includes these values are the respective column sums. However, there are two degrees of freedom in each sum, so it is impossible to determine the specific values from just these two sums alone. □

This approach is better in protecting against collusion if everyone behaves truthfully, however it shares some of the same vulnerabilities to dishonest agents who may deviate from the prescribed behavior.

Acknowledgments

This reading was written by Dr. Shawn Ong.