CMSC 28400 Introduction to Cryptography **Autumn 2021**

Notes #5: Stream Ciphers and Computational Security

Instructor: David Cash

5.1 Limitations of the One-Time Pad

In the last notes we analyzed the One-Time Pad (OTP) cipher, which had a lot going for it: An implementation should be very fast, since it's just applying exclusive-OR on two strings, and intuitively-strong security was proved. However, while the OTP has been used in the past[1], the OTP itself is not frequently used today, at least not in the form we saw. This is due to a few reasons:

LONG KEYS. To encrypt a message m, the OTP requires a key of the same length. If m is a single credit-card number this may be okay, but to encrypt the large amount of traffic on the Internet, this is problematic. The key needs to be transmitted out-of-band, and doing so is typically expensive. Classically, this meant sending someone with a briefcase of books of keys. In the modern setting, we do this via something called *key exchange*, which we will cover in the second half of the course. Key exchange runs quickly enough to transmit short keys on any modern laptop, but it's not fast enough to run for very large keys.

ONE-TIME SECURITY. As we saw before, the OTP loses security if a key is used twice. One could simply avoid doing so, but then the key-length problem bites again. We'd like to be able to encrypt multiple times with the same key if possible, and get some meaningful security.

MALLEABILITY. This issue is more subtle, but is very relevant in practice. Suppose the OTP is being used to encrypt a web session, and an attacker completely controls a router in the middle, with the ability to change the bits of the traffic however it likes. A ciphertext $c = k \oplus m$ is sent. Then the attacker is able to modify c so that *the receiver will recover a message other than* m, and moreover the attacker can control what is received in a predictable way. For example, suppose the attacker flips the first bit of c; call the result c'. If the receiver decrypts c', they'll recover m, except with the first bit flipped.

This type of attack shows that the OTP is *malleable*, in the language of modern cryptography. It may look a little strange, but malleability attacks lead to practical vulnerabilities all the time!

5.1.1 Our Plan

The above issues (long keys, one-time security, malleability) will guide our development of practical encryption methods for the next few weeks. These notes will begin addressing the first issue. In the next section we'll prove that *any* cipher unavoidably has long keys if we require it to be perfectly secret. In the following sections we'll look at something called a *stream cipher*, which is used to "stretch" a short key, and discuss their security. In the next set of notes we'll get back to applying

[1] You may enjoy reading about the Venona Project, where American agencies were able to decrypt Soviet OTP-encrypted messages from the 40's to the 80's.

stream ciphers to encryption. The latter two issues require more work and will be addressed in the next two weeks.

5.2 Perfectly Secret Ciphers Must Have $|\mathcal{K}| \geq |\mathcal{M}|$

The OTP has key as long as the messages. At first glance, one may hope to develop a different perfectly-secret cipher that has a shorter key. The following theorem shows this is in some sense impossible (and is precisely impossible if one insists that $\mathcal{K}$ is a set of fixed-length bitstrings).

Theorem 1. *Suppose a cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is perfectly secret. Then $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof. We prove the contrapositive, meaning we show that if $|\mathcal{K}| < |\mathcal{M}|$, then E is *not* perfectly secret. In order to show E is not perfectly secret, we find some $m_0, m_1 \in \mathcal{M}$ and $c \in \mathcal{C}$ such that

$$\Pr[E(\mathbf{K}, m_0) = c] \neq \Pr[E(\mathbf{K}, m_1) = c], \tag{5.1}$$

where $\mathbf{K}$ is uniform on $\mathcal{K}$.

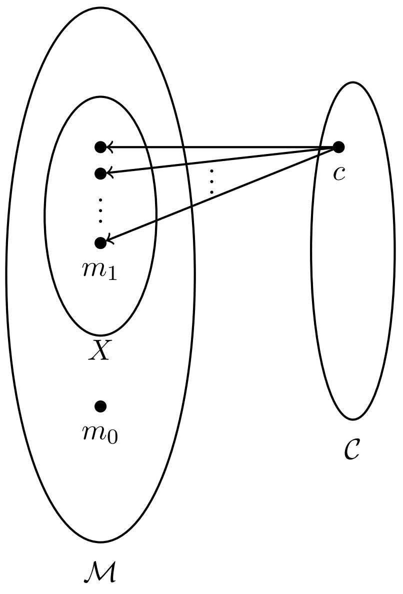
We start by taking $c \in \mathcal{C}$ to be an arbitrary element of the image of E (so, we know that $E(k, m) = c$ for some $k \in \mathcal{K}, m \in \mathcal{M}$, but we don't care how its picked beyond that).

Define the set X of "all possible decryptions of c" as

$$X = \{E^{-1}(k, c) \mid k \in \mathcal{K}\}.$$

Notice that $|X| \leq |\mathcal{K}|$, because at most one element is added to X per element of $\mathcal{K}$ (it might be less because we might add the same element twice). By our choice of c we also have that X is non-empty; pick some arbitrary $m_1 \in X$.

Since $|X| \leq |\mathcal{K}|$, and we assumed $|\mathcal{K}| < |\mathcal{M}|$, we have $|X| \leq |\mathcal{M}|$. Therefore X does not contain all of $\mathcal{M}$, and can choose some $m_0 \in \mathcal{M}$ that is not in X. The following diagram shows the relationship between c, m_1 and m_0.



We have chosen our m_0, m_1, and c. It remains to verify that (5.1) holds. This follows by two observations:

- $\Pr[E(\mathbf{K}, m_0) = c] = 0$, because there is no key k such that $E(k, m_0) = c$.
- $\Pr[E(\mathbf{K}, m_1) = c] \geq 1/|\mathcal{K}| > 0$, because there is at least one key k such that $E(k, m_1) = c$.

This shows the probabilities are not equal and completes the proof. $\square$

5.3 Plan: "Psuedo"-One-Time Pad Encryption

In the last section we showed that *any* perfectly secret cipher $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ must satisfy $|\mathcal{K}| \geq |\mathcal{M}|$. Thus for each n, OTP_n is "optimal" in terms of key-length for perfectly-secret ciphers.

In practice the one-time pad is rarely used because exchanging the key is cumbersome. In practice one resolves this barrier by giving up on perfect secrecy, and using a weaker-but-good-enough notion of security. In order to work with a shorter key, we'll look for a function that can "securely stretch" a short key into a long one. More formally we'll want some

$$G : \{0,1\}^n \to \{0,1\}^\ell$$

where $\ell \ll n$; For example, $n = 256$ and ℓ practically-infinite (like $\ell = 2^{64}$) is typical. (Of course we need that we evaluate G efficiently so we can actually use it; Typically we want to be able to compute the bits of G from the first to however many we need, rather than needing to compute them all at the same time, up front.)

Functions G for stretching keys will be called *pseudorandom generators* or *stream ciphers* (the two names mean the same thing for us; The applied community often uses "stream cipher" while the theoretical community typically uses "pseudorandom generators"). Note that stream ciphers are not really *ciphers* as defined earlier, since we they only take one input, and we are not requiring them to satisfy any one-to-one condition.

We will use G to build a cipher as follows:

Definition 5.1. *Let n, ℓ be positive integers, and let $G : \{0,1\}^n \to \{0,1\}^\ell$ be a function. Define the* G-pseudo-one-time-pad cipher

$$\mathrm{OTP}_G : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$$

defined by

$$\mathrm{OTP}_G(k, m) = G(k) \oplus m.$$

First, the bad news.

Theorem 2. *Let n, ℓ be positive integers, and let $G : \{0,1\}^n \to \{0,1\}^\ell$ be a function. If $\ell > n$ then OTP_G is not perfectly secret.*

This is true because there are fewer keys (2^n) than messages (2^ℓ) when $\ell > n$.

The good news is that, practically speaking, we can still do very well; We just need to be careful about how we design G, and then also careful about what level of security we are actually achieving. Thus we shift our focus from OTP_G to G itself for now. In future lectures we'll give a definition to analyze OTP_G.

5.4 Defining Security of Pseudorandom Generators

We want a pseudorandom generator that will work well in stretching a key to use in place of a one-time pad. A natural requirement is that the output of G, when run on a uniformly random input, should pass an array of statistical tests. The numbers of zeros and ones should be approximately balanced, like a random string, and there should be typical runs of consecutive zeros and ones, and so on.

Typically scientific applications of pseudorandom generators get by with this sort of thinking. Since they employ the random bits in simulating physical phenomena, they only need to worry about a specific model behaving normally on the output bits. As we will see, our situation is however much more demanding.

Defining the quality of pseudorandom bits is tricky. Even if G always outputs something very non-random (like the all-zeros string), we can't for sure say that it's *not* random; After all, for a uniform sample, all outcomes are equally likely, including the all-zeros string. Instead of looking at individual strings, we need to look at the *distribution* output by G.

Let us first give a definition to measure how effective a statistical test (denoted $\mathcal{D}$ below, for "distinguisher") is against a pseudorandom generator.

Definition 5.2. *Let $G : \{0,1\}^n \to \{0,1\}^\ell$ and $\mathcal{D} : \{0,1\}^\ell \to \{0,1\}$ be functions and let $\mathbf{K}$ be uniform on $\{0,1\}^n$ and $\mathbf{U}$ be uniform on $\{0,1\}^\ell$. Then we define the* (pseudorandom generator) distinguishing advantage of $\mathcal{D}$ against G *to be*

$$\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) = |\Pr[\mathcal{D}(G(\mathbf{K})) = 1] - \Pr[\mathcal{D}(\mathbf{U}) = 1]| .$$

The definition is useful because, for a specific G and any test $\mathcal{D}$, it assigns a number between 0 and 1 (the higher the better $\mathcal{D}$ is). Thus we can compare distinguishers, and speak of how well G passes statistical tests.

Let's do a warm-up example with the definition, and then return to interpreting the definition later.

Example 5.1. *Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be defined by $G(k) = k\|0$, where we use the symbol $\|$ to denote string concatenation (in words, G simply puts a zero on the end of k). For any reasonable definition of pseudorandom generation, G should fail – It's just adding a zero bit!*

To formalize the failure of G, take

$$\mathcal{D}(w) = \begin{cases} 1 & \text{if } w \text{ ends in } 0 \\ 0 & \text{otherwise} \end{cases} .$$

Then we claim

$$\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) = \frac{1}{2}.$$

We can compute this by looking at the two probabilities in the definition above:

$$\Pr[\mathcal{D}(G(\mathbf{K})) = 1] = 1$$

because $G(\mathbf{K})$ will end in a 0 with probability 1. Next,

$$\Pr[\mathcal{D}(\mathbf{U}) = 1] = \frac{1}{2}.$$

This latter probability is $1/2$ because a random string ends in 0 with probability $1/2$.

We note from this example that G is a terrible pseudorandom generator, and this distinguisher only achieves advantage 1/2; Thus we should consider such an advantage to "very large".

Exercise 5.1. *Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be defined by $G(k) = k\|k$ (i.e it copies k twice). Find a good distinguisher $\mathcal{D}$ for G and calculate $\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D})$.*

Armed with this definition we can ask that $\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D})$ be "low" for *any* distinguisher $\mathcal{D}$. If we're feeling ambitious, we could even ask that it's always zero. Unfortunately this is impossible, as we now prove.

Theorem 3. *Let n, ℓ be positive integers with $\ell > n$, and let $G : \{0,1\}^n \to \{0,1\}^\ell$ be a function. Then there exists $\mathcal{D} : \{0,1\}^\ell \to \{0,1\}$ such that*

$$\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) \geq 1 - \frac{2^n}{2^\ell} \geq \frac{1}{2}.$$

Proof. Let Im G be the image of G, i.e. $\mathrm{Im}\ G = \{G(k) \ : \ k \in \{0,1\}^n\}$. The function $\mathcal{D} : \{0,1\}^\ell \to \{0,1\}$ is defined by

$$\mathcal{D}(w) = \begin{cases} 1 & \text{if } w \in \mathrm{Im}\ G \\ 0 & \text{otherwise} \end{cases}.$$

To analyze $\mathcal{D}$ we need two observations about Im G. First, the random variable $G(\mathbf{K})$ is always in Im G. Second, $|\mathrm{Im}\ G| \leq 2^n$, because each $k \in \{0,1\}^n$ adds at most one element to the image, and there are 2^n such k (it may be less when G is not one-to-one).

Then letting $\mathbf{K}, \mathbf{U}$ as in the theorem,

$$\Pr[\mathcal{D}(G(\mathbf{K})) = 1] = 1$$

because $G(\mathbf{K})$ is always in the image of G. On the other hand,

$$\Pr[\mathcal{D}(\mathbf{U}) = 1] \leq \frac{2^n}{2^\ell},$$

because $\mathcal{D}(\mathbf{U}) = 1$ is equivalent to $\mathbf{U} \in \mathrm{Im}\ G$, which happens with probability $|\mathrm{Im}\ G|/2^\ell$. By our observation on the size Im G, this probability is at most $\frac{2^n}{2^\ell}$. Finally, plug these in to get

$$\begin{aligned} \mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) &= |\Pr[\mathcal{D}(G(\mathbf{K})) = 1] - \Pr[\mathcal{D}(\mathbf{U}) = 1]| \\ &\geq \Pr[\mathcal{D}(G(\mathbf{K})) = 1] - \Pr[\mathcal{D}(\mathbf{U}) = 1] \\ &\geq 1 - \frac{2^n}{2^\ell} \geq 1 - \frac{1}{2} = \frac{1}{2}. \end{aligned}$$

□

Exercise 5.2. *Let $G : \{0,1\}^n \to \{0,1\}^\ell$. Show that for* any *$\mathcal{D} : \{0,1\}^\ell \to \{0,1\}$,*

$$0 < \mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) < 1.$$

That is, no distinguisher can be "perfectly correct" or "perfectly incorrect".

5.4.1 The Computational View of Security

On the one hand, this theorem is bad for us, because it says that there will always be a distinguisher $\mathcal{D}$ that can detect a pattern in G. However, in practical terms this distinguisher is not so concerning; Unless we have some insight into the workings of G, to implement the $\mathcal{D}$ from the theorem on a computer, we need to test if an input string w is in the image of G. To do this, we would need to loop over *all* $k \in \{0,1\}^n$ and test if $G(k)$ is equal to w.

This resulting program would run *exponential time* 2^n. For small n like $n = 30$ or 40 this would run relatively quickly. But for $n = 60$ or so this would easily overwhelm a personal computer. And for $n = 128$, it is commonly thought that such a computation would be beyond even the strongest agencies on earth. For $n = 256$, such a computation seems *beyond the physical limits of the universe*, as some estimates put the number of atoms in the universe at around 2^{256}.

Exercise 5.3. *To get an idea for the scaling involved, try estimating the cost to implement a* 2^{128}*-level computation in one year. Find a recent graphics card, and estimate the number of operations per year that it can compute (taking it simply the number of cores times the clock speed). Assuming generously that one key can be tried per cycle per core, calculate the number of graphics cards (and hence the cost) required to try all* 128*-bit keys.*

In practice one typically designs a G and tries to find efficient distinguishing attacks. For an attack $\mathcal{D}$ under consideration, we care about both its runtime T and its advantage $\mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D})$. An example security target for G might be:

$$\text{For all } \mathcal{D} \text{ with running time } 2^{128} \text{ or less, } \mathbf{Adv}_G^{\mathrm{prg}}(\mathcal{D}) \leq 1/2^{128}. \tag{5.2}$$

Note that we've allowed a huge running time and still want an incredibly small advantage. The small advantage is to ensure that even after using G an astronomical number of times, any detectable bias should still be very, very small.

Note that we have been informal with our notion of "running time." We will leave this informal for the purposes of this course, but if we really wanted to formalize it we could turn to models of computation like Turing Machines and circuit families; This course is designed to work without knowledge of those concepts. There are some interesting details to sort out there, but we'll get by just fine without commenting on that again. One thing we *can't* do, however, is use big-Oh analysis: We don't care about limits going to infinity. We will care about how hard it is to break our constructions.

Finally, we remark on the possibility of finding a good stream cipher G and then *proving* a statement like that of Equation 5.2. This would require somehow analyzing the structure of any possible efficient attack, and arguing that none could possibly work. Unfortunately, such an argument is totally beyond the state-of-the-art of theoretical computer science. In fact, making progress on such questions is one of the central goals of *computational complexity theory.* You have probably heard of the "P vs. NP" problem, and the same issue is at work here: We have almost no tools for proving that efficient computation is incapable of doing anything in particular. This is not great for practice (when do we know if something is secure?), but it adds quite a bit of mystery to cryptography, since we never *really* know if there is a more clever attack or not. In any case, it does not mean that the type of statement above is without value, since it gives us a standard by which to declare G "broken," should someone present an attack.