# Notes for Lecture 1

Mon, 1/12/2026

## Review: The calculus of congruences

**Example 1.** Today is Monday. What day of the week will it be a year (365 days) from now?

**Solution.** Since $365 \equiv 1 \pmod 7$, it will be Tuesday on 1/12/2027.

$$a \equiv b \pmod n \qquad \text{means} \qquad a = b + mn \quad (\text{for some } m \in \mathbb{Z})$$

In that case, we say that "$a$ is congruent to $b$ modulo $n$".

In other words: $a \equiv b \pmod n$ if and only if $a - b$ is divisible by $n$.

**Example 2.** $17 \equiv 5 \pmod{12}$ as well as $17 \equiv 29 \equiv -7 \pmod{12}$

We say that $5, 17, 29, -7$ all represent the same **residue** modulo $12$.

There are exactly $12$ different residues modulo $12$.

**Example 3.** Every integer $x$ is congruent to one of $0, 1, 2, 3, 4, \ldots, 11$ modulo $12$.

We therefore say that $0, 1, 2, 3, 4, \ldots, 11$ form a **complete set of residues** modulo $12$.

Another natural complete set of residues modulo $12$ is: $0, \pm 1, \pm 2, \ldots, \pm 5, 6$

[$-6$ is not included because $-6 \equiv 6$ modulo $12$.]

**Online homework.** When entering solutions modulo $n$ for online homework, your answer needs to be from one of the two natural sets of residues above.

**Example 4.** Modulo $7$, we have the complete sets of residues $0, 1, 2, 3, 4, 5, 6$ and $0, \pm 1, \pm 2, \pm 3$.
A less obvious set is $0, 3, 3^2, 3^3, 3^4, 3^5, 3^6$.

**Review.** Note that $3^6 \equiv 1 \pmod 7$ by **Fermat's little theorem**. Because $6$ is the smallest positive exponent such that $3^k \equiv 1 \pmod 7$, we say that the **multiplicative order** of $3 \pmod 7$ is $6$. This makes $3 \pmod 7$ a **primitive root**.

On the other hand, the **multiplicative order** of $2 \pmod 7$ is $3$. (Why?!)

**Example 5.** $67 \cdot 24 \equiv 4 \cdot 3 \equiv 5 \pmod 7$

The point being that we can (and should!) reduce the factors individually first (to avoid the large number we would get when actually computing $67 \cdot 24$ first). This idea is crucial in the computations we (better, our computers) will later do for cryptography.

**Example 6. (but careful!)** If $a \equiv b \pmod n$, then $ac \equiv bc \pmod n$ for all integers $c$.
However, the converse is not true! We can have $ac \equiv bc \pmod n$ without $a \equiv b \pmod n$ (even assuming that $c \not\equiv 0$).

**For instance.** $2 \cdot 4 \equiv 2 \cdot 1 \pmod 6$ but $4 \not\equiv 1 \pmod 6$

**However.** $2 \cdot 4 \equiv 2 \cdot 1 \pmod 6$ means $2 \cdot 4 = 2 \cdot 1 + 6m$. Hence, $4 = 1 + 3m$, or, $4 \equiv 1 \pmod 3$.

The issue is that $2$ is not invertible modulo $6$.

$a$ is invertible modulo $n \iff \gcd(a, n) = 1$

Similarly, $ab \equiv 0 \pmod n$ does not always imply that $a \equiv 0 \pmod n$ or $b \equiv 0 \pmod n$.

**For instance.** $4 \cdot 15 \equiv 0 \pmod 6$ but $4 \not\equiv 0 \pmod 6$ and $15 \not\equiv 0 \pmod 6$

Armin Straub
straub@southalabama.edu

**Good news.** These issues do not occur when $n$ is a **prime** $p$.

- If $ab \equiv 0 \pmod{p}$, then $a \equiv 0 \pmod{p}$ or $b \equiv 0 \pmod{p}$.
- Suppose $c \not\equiv 0 \pmod{p}$. If $ac \equiv bc \pmod{p}$, then $a \equiv b \pmod{p}$.

**Example 7.** Determine $4^{-1} \pmod{13}$.

**Recall.** This is asking for the **modular inverse** of $4$ modulo $13$. That is, a residue $x$ such that $4x \equiv 1 \pmod{13}$.

**Brute force solution.** We can try the values $0, 1, 2, 3, \ldots, 12$ and find that $x = 10$ is the only solution modulo $13$ (because $4 \cdot 10 \equiv 1 \pmod{13}$).

This approach may be fine for small examples when working by hand, but is not practical for serious congruences. On the other hand, the Euclidean algorithm, reviewed below, can compute modular inverses extremely efficiently.

**Glancing.** In this special case, we can actually see the solution if we notice that $4 \cdot 3 = 12$, so that $4 \cdot 3 \equiv -1 \pmod{13}$ and therefore $4^{-1} \equiv -3 \pmod{13}$.

**Example 8.** Solve $4x \equiv 5 \pmod{13}$.

**Solution.** From the previous problem, we know that $4^{-1} \equiv -3 \pmod{13}$.
Hence, $x \equiv 4^{-1} \cdot 5 \equiv -3 \cdot 5 = -2 \pmod{13}$.

**(Bézout's identity)** Let $a, b \in \mathbb{Z}$ (not both zero). There exist $x, y \in \mathbb{Z}$ such that

$$\gcd(a, b) = ax + by.$$

The integers $x, y$ can be found using the **extended Euclidean algorithm**.
In particular, if $\gcd(a, b) = 1$, then $a^{-1} \equiv x \pmod{b}$ (as well as $b^{-1} \equiv y \pmod{a}$).

Here, $\mathbb{Z}$ denotes the set of all integers $0, \pm 1, \pm 2, \ldots$

**Example 9.** Find $d = \gcd(17, 23)$ as well as integers $r, s$ such that $d = 17r + 23s$.

**Solution.** We apply the extended Euclidean algorithm:

$$\begin{aligned} &\gcd(17, 23) && \boxed{23} = 1 \cdot \boxed{17} + 6 && \text{or:} && \boxed{A} \quad 6 = 1 \cdot \boxed{23} - 1 \cdot \boxed{17} \\ &= \gcd(6, 17) && \boxed{17} = 3 \cdot \boxed{6} - 1 && && \boxed{B} \quad 1 = -1 \cdot \boxed{17} + 3 \cdot \boxed{6} \\ &= 1 \end{aligned}$$

Backtracking through this, we find that:

$$1 \underset{\boxed{B}}{=} -1 \cdot \boxed{17} + 3 \cdot \boxed{6} \underset{\boxed{A}}{=} -1 \cdot \boxed{17} + 3 \cdot \left(1 \cdot \boxed{23} - 1 \cdot \boxed{17}\right) = -4 \cdot \boxed{17} + 3 \cdot \boxed{23}$$

That is, **Bézout's identity** takes the form $1 = -4 \cdot 17 + 3 \cdot 23$.

**Comment.** Note how our second step was $\boxed{17} = 3 \cdot \boxed{6} - 1$ rather than $\boxed{17} = 2 \cdot \boxed{6} + 5$. The latter works as well but requires a third step (do it!). In general, we save time by allowing negative remainders if they are smaller in absolute value.

**Example 10.** Determine $17^{-1} \pmod{23}$.

**Solution.** By the previous example, $1 = -4 \cdot 17 + 3 \cdot 23$. Reducing modulo $23$, we get $-4 \cdot 17 \equiv 1 \pmod{23}$. Hence, $17^{-1} \equiv -4 \pmod{23}$. [Or, if preferred, $17^{-1} \equiv 19 \pmod{23}$.]

Armin Straub
straub@southalabama.edu

**Example 11.** Determine $16^{-1} \pmod{25}$.

**Solution.** We apply the extended Euclidean algorithm:

$$\begin{aligned} &\gcd(16,25) && \boxed{25} = 2\cdot\boxed{16} - 7 && \text{or:} && \boxed{A}\ 7 = -1\cdot\boxed{25} + 2\cdot\boxed{16} \\ &= \gcd(7,16) && \boxed{16} = 2\cdot\boxed{7} + 2 && && \boxed{B}\ 2 = 1\cdot\boxed{16} - 2\cdot\boxed{7} \\ &= \gcd(2,7) && \boxed{7} = 3\cdot\boxed{2} + 1 && && \boxed{C}\ 1 = \boxed{7} - 3\cdot\boxed{2} \\ &= 1 \end{aligned}$$

Backtracking through this, we find that:

$$1 \underset{\boxed{C}}{=} \boxed{7} - 3\cdot\boxed{2} \underset{\boxed{B}}{=} 7\cdot\boxed{7} - 3\cdot\boxed{16} \underset{\boxed{A}}{=} -7\cdot\boxed{25} + 11\cdot\boxed{16}$$

That is, **Bézout's identity** takes the form $-7\cdot 25 + 11\cdot 16 = 1$.

Reducing modulo $25$, we get $11\cdot 16 \equiv 1 \pmod{25}$. Hence, $16^{-1} \equiv 11 \pmod{25}$.

## Application: credit card numbers

Have you ever thought about the numbers on your credit card? Usually, these are 16 digits. For instance, 4266 8342 8412 9270.

No worries (or false hopes...). While close, this is not exactly my credit card number.

- The first digit(s) of a credit card identify the issuer of the card. For instance, a leading $4$ is typically Visa, $51$ to $55$ indicate Mastercard, and $34$, $37$ indicate American Express. The above credit card is indeed a Visa card.

  More information at: https://en.wikipedia.org/wiki/Payment_card_number

- The last digit is a **check digit**, and a valid credit card number must pass the **Luhn check** (patented by IBM scientist Hans Peter Luhn in 1954/60; now in public domain).

  This works as follows: every second digit, starting with the first, is doubled. If that results in a two-digit number, we take the sum of those two digits.

$$\begin{bmatrix} & 4 & 2 & 6 & 6 & 8 & 3 & 4 & 2 & 8 & 4 & 1 & 2 & 9 & 2 & 7 & 0 \\ \times 2 & 8 & & 12 & & 16 & & 8 & & 16 & & 2 & & 18 & & 14 & \\ & 8 & 2 & 3 & 6 & 7 & 3 & 8 & 2 & 7 & 4 & 2 & 2 & 9 & 2 & 5 & 0 \end{bmatrix}$$

  The other half of the digits is left unchanged. We then add all these digits and reduce modulo $10$:

$$8+2+3+6+7+3+8+2+7+4+2+2+9+2+5+0 \equiv 0 \pmod{10}$$

  The result of that computation must be $0$. Otherwise, the credit card number fails the Luhn check and is invalid.

**Example 12. (extra exercise)**

(a) Check that the number 4266 8342 8412 9280 fails the Luhn check.

(b) How do we have to change the last digit to turn this into a valid credit card number?

Armin Straub
straub@southalabama.edu

The purpose of the Luhn check is to detect accidental errors.

[A random credit card number has a 90% chance of failing the Luhn check. Why?!]

On the other hand, as the previous example shows, it provides basically no protection against malicious attacks (except against amateur criminals not aware of the Luhn check).

The Luhn check was designed before online banking (patent filed in 1954). So a special focus is on detecting accidental errors that occur frequently when writing down (things like) credit card numbers by hand.

- For instance, it is common that a single digit gets messed up. Every such error is detected by the Luhn check. (Why?!)
- Another common error is to transpose two digits. Every such error (with the exception of 09 versus 90) is detected.

  **For instance.** A 82 at the beginning contributes $7+2=9$ to the check sum, while a 28 contributes $4+8\equiv 2$ to the sum. Hence, replacing one with the other will result in the Luhn check failing.

  **Advanced comment.** An alternative checksum formula that can detect all single digit changes as well as all transpositions is the Verhoeff algorithm (1969). It is, however, much more complicated and cannot be readily performed by hand.

**Example 13.** The doubling and sum-of-digits procedure permutes the digits as follows:

| original digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| adjusted digit | 0 | 2 | 4 | 6 | 8 | 1 | 3 | 5 | 7 | 9 |
| difference $(\mathrm{mod}\,10)$ | 0 | 1 | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 0 |

**Note.** Looking at the differences modulo $10$, we can see why the Luhn check is able to detect all transposition errors (except 09 versus 90).

**Example 14.** The Luhn check has the somewhat complicated feature that every second digit has to be doubled. Why do we not just add all the original digits and reduce the sum modulo $10$?

**Solution.** One reason is that this simplified check does not catch the transposition of two digits. Why?!

[On the other hand, that simplified check does also detect if just a single digit is incorrect.]

**Example 15. (extra)** The International Standard Book Number ISBN-10 consists of nine digits $a_1a_2\ldots a_9$ followed by a tenth check digit $a_{10}$ (the symbol $X$ is used if the digit equals $10$), which satisfies

$$a_{10}\equiv\sum_{k=1}^{9} ka_k \pmod{11}.$$

The ISBN 0-13-186239-? is missing the check digit (printed as "?"). Compute it!

**Solution.** $1\cdot 0+2\cdot 1+3\cdot 3+4\cdot 1+5\cdot 8+6\cdot 6+7\cdot 2+8\cdot 3+9\cdot 9=210\equiv 1 \pmod{11}$

Hence, the full ISBN is 0-13-186239-1.

**Comment.** The check digit is designed so that it is always possible to detect when a single digit is messed up. It is also always possible to detect when two digits are transposed.

This is another example of **error checking**, which is standard practice for all sorts of identification numbers (such as bank account numbers, VIN, ...). With a little more effort **error correction** is also possible.

## Euler's phi function

**Definition 16. Euler's phi function** $\phi(n)$ gives the number of integers in $\{1, 2, ..., n\}$ that are relatively prime to $n$.

In other words, $\phi(n)$ counts how many residues are invertible modulo $n$.

**Example 17.** Compute $\phi(n)$ for $n = 1, 2, ..., 8$.

**Solution.** $\phi(1) = 1$, $\phi(2) = 1$, $\phi(3) = 2$, $\phi(4) = 2$, $\phi(5) = 4$, $\phi(6) = 2$, $\phi(7) = 6$, $\phi(8) = 4$.

**Observation.** $\phi(n) = n - 1$ if and only if $n$ is a prime.
This is true because $\phi(n) = n - 1$ if and only if $n$ is coprime to all of $\{1, 2, ..., n-1\}$.

**Observation.** If $p$ is a prime, then $\phi(p^k) = p^k - p^{k-1} = p^k\left(1 - \frac{1}{p}\right)$.

This is true because, if $p$ is a prime, then $n = p^k$ is coprime to all $\{1, 2, ..., p^k\}$ except $p, 2p, 3p, ..., p^k$ (the multiples of $p$, of which there are $p^k / p = p^{k-1}$ many).

If the prime factorization of $n$ is $n = p_1^{k_1} \cdots p_r^{k_r}$, then $\phi(n) = n\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_r}\right)$.

**Why is this true?**

- We observed above that the formula is true if $n = p^k$ is a prime power.
- On the other hand, for composite $n$, say $n = ab$, we have: $\boxed{\phi(ab) = \phi(a)\phi(b) \text{ if } \gcd(a, b) = 1}$
  This is a consequence of the Chinese remainder theorem. (Review if necessary! We'll use it later but will only review it briefly then.)

The above formula follows from combining these two observations. Can you fill in the details?

**Example 18.** Compute $\phi(35)$.

**Solution.** $\phi(35) = \phi(5 \cdot 7) = \phi(5)\phi(7) = 4 \cdot 6 = 24$

**Example 19.** Compute $\phi(100)$.

**Solution.** $\phi(100) = \phi(2^2 \cdot 5^2) = \phi(2^2)\phi(5^2) = (2^2 - 2^1) \cdot (5^2 - 5^1) = 40$

[Alternatively: $\phi(100) = \phi(2^2 \cdot 5^2) = 100\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{5}\right) = 40$]

**Example 20.** Compute $\phi(1000)$.

**Solution.** $\phi(1000) = \phi(2^3) \cdot \phi(5^3) = (8 - 4)(125 - 25) = 400$

[Alternatively: $\phi(1000) = \phi(2^3 \cdot 5^3) = 1000\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{5}\right) = 400$.]

Armin Straub
straub@southalabama.edu

## Historical examples of symmetric encryption

Alice wants to send a secret message to Bob.

What Alice sends will be transmitted through an unsecure medium (like the internet), meaning that others can read it. However, it is important to Alice and Bob that no one else can understand it.

The original message is referred to as the **plaintext** $m$. What Alice actually sends is called the **ciphertext** $c$ (the encrypted message).

**Symmetric encryption** algorithms rely on a secret key $k$ (from some **key space**) shared by Alice and Bob (but unknown to anyone else).


Our ultimate goal will be to secure messaging against both:

- eavesdropping (goal: **confidentiality**)
- tampering (goal: **integrity** and, even stronger, **authenticity**)

  The symmetric encryption approach, by itself, cannot fully protect against tampering. For instance, an attacker can collect previously sent messages, resend them, or use them to replace new messages. (You could preface each message with something like a time stamp to address these issues. But that's getting ahead of ourselves; and there are better ways.)

### Shift cipher

The alphabet for our messages will be $A, B, ..., Z$, which we will identify with $0, 1, ..., 25$.

So, for instance, $C$ is identified with the number $2$.

**Example 21. (shift cipher)** A key is an integer $k \in \{0, 1, ..., 25\}$. Encryption works character by character using

$$E_k: \quad x \mapsto x + k \pmod{26}.$$

Obviously, the decryption $D_k$ works as $x \mapsto x - k \pmod{26}$.

The **key space** is $\{0, 1, ..., 25\}$. It has size $26$. [Well, $k = 0$ is a terrible key. Maybe we should exclude it.]

**For instance.** If $k = 1$, then the message $HELLO$ is encrypted as $IFMMP$.

If $k = 2$, then the message $HELLO$ is encrypted as $JGNNQ$.

**Historic comment.** Caesar encrypted some private messages with a shift cipher (typically using $k = 3$). The shift cipher is therefore also often called Caesar's cipher.

While completely insecure today, it was fairly secure at the time (with many of his enemies being illiterate).

**Modern comment.** Many message boards on the internet "encrypt" things like spoilers or solutions using a shift cipher with $k = 13$. This is called ROT13. What's special about the choice $k = 13$?

**Solution.** Since $-13 \equiv 13 \pmod{26}$, for ROT13, encryption and decryption are the same!

Armin Straub
straub@southalabama.edu

**Example 22. (affine cipher)** A slight upgrade to the shift cipher, we encrypt each character as

$$E_{(a,b)}: \quad x \mapsto ax + b \pmod{26}.$$

How does the decryption work? How large is the key space?

**Solution.** Each character $x$ is decrypted via $x \mapsto a^{-1}(x-b) \pmod{26}$.

The key is $k = (a,b)$. Since $a$ has to be invertible modulo $26$, there are $\phi(26) = \phi(2) \cdot \phi(13) = 12$ possibilities for $a$. There are $26$ possibilities for $b$. Hence, the key space has size $12 \cdot 26 = 312$.

## Fermat's little theorem

**Example 23. (warmup)** What a terrible blunder... Explain what is wrong!

$$\text{(incorrect!)} \quad 10^9 \equiv 3^2 = 9 \equiv 2 \pmod{7}$$

**Solution.** $10^9 = 10 \cdot 10 \cdot \ldots \cdot 10 \equiv 3 \cdot 3 \cdot \ldots \cdot 3 = 3^9$. Hence, $10^9 \equiv 3^9 \pmod{7}$.

However, there is no reason, why we should be allowed to reduce the exponent by $7$ (and it is incorrect).

**Corrected calculation.** $3^2 \equiv 2$, $3^4 \equiv 4$, $3^8 \equiv 16 \equiv 2$. Hence, $3^9 = 3^8 \cdot 3^1 \equiv 2 \cdot 3 \equiv -1 \pmod{7}$.

By the way, this approach of computing powers via exponents that are $2,\ 4,\ 8,\ 16,\ 32,\ \ldots$ is called **binary exponentiation**. It is crucial for efficiently computing large powers.

**Corrected calculation (using Fermat).** $3^6 \equiv 1$ just like $3^0 = 1$. Hence, we are allowed to reduce exponents modulo $6$. Hence, $3^9 \equiv 3^3 \equiv -1 \pmod{7}$.

**Theorem 24. (Fermat's little theorem)** Let $p$ be a prime, and suppose that $p \nmid a$. Then

$$a^{p-1} \equiv 1 \pmod{p}.$$

**Proof. (beautiful!)** Since $a$ is invertible modulo $p$, the first $p-1$ multiples of $a$,

$$a, 2a, 3a, \ldots, (p-1)a$$

are all different modulo $p$. Clearly, none of them is divisible by $p$.

Consequently, these values must be congruent (in some order) to the values $1, 2, \ldots, p-1$ modulo $p$. Thus,

$$a \cdot 2a \cdot 3a \cdot \ldots \cdot (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (p-1) \pmod{p}.$$

Cancelling the common factors (allowed because $p$ is prime!), we get $a^{p-1} \equiv 1 \pmod{p}$. □

**Remark.** The "little" in this theorem's name is to distinguish this result from Fermat's last theorem that $x^n + y^n = z^n$ has no integer solutions if $n > 2$ (only recently proved by Wiles).

## Vigenere cipher (vector shift cipher)

See Section 2.3 of our book for a full description of the Vigenere cipher.

This cipher was long believed by many (until early 20th) to be secure against ciphertext only attacks (more on the classification of attacks shortly).

**Example 25.** Let us encrypt $HOLIDAY$ using a Vigenere cipher with key $BAD$ (i.e. $1, 0, 3$).

| | $H$ | $O$ | $L$ | $I$ | $D$ | $A$ | $Y$ |
|---|---|---|---|---|---|---|---|
| $+$ | $B$ | $A$ | $D$ | $B$ | $A$ | $D$ | $B$ |
| $=$ | $I$ | $O$ | $O$ | $J$ | $D$ | $D$ | $Z$ |

Hence, the ciphertext is $IOOJDDZ$.

**Example 26. (bonus challenge!)** You find a post-it with the following message:

```
NIVU QV JR DTTS ULIFI FOI KIVVF
```

Can you make any sense of it? Word on the street is that Alice was using a Vigenere cipher with key of size $3$ with last letter `R`.

(To collect a bonus point, send me an email before next class with the plaintext and how you found it.)

If you can decipher the above message, you have successfully mounted a **ciphertext only attack**.

That is, just knowing the encrypted message, we were able to decrypt it (and discover the key that was used). This is the worst kind of vulnerability.

## Attacks

So far, we considered the weakest kind of attack only: namely, a **ciphertext only attack**. And, even then, the historical ciphers prove to be terribly insecure.

However, we need to also worry about attacks where our enemy has additional insight.

- In a **known plaintext attack**, the enemy somehow has knowledge of a plaintext-ciphertext pair $(m, c)$.
- In a **chosen plaintext attack**, the enemy can, herself, compute $c = E(m)$ for a chosen plaintext $m$ ("gained some sort of access to our encryption device").
- In a **chosen ciphertext attack**, the enemy can, herself, compute $m = D(c)$ for a chosen ciphertext $c$ ("gained some sort of access to our decryption device").

There exist many variations of these. Sometimes, the attacker can make several choices (maybe even adaptively), sometimes she only has partial information.

**Example 27.** Alice sends the ciphertext $BKNDKGBQ$ to Bob. Somehow, Eve has learned that Alice is using the Vigenere cipher and that the plaintext is $ALLCLEAR$. Next day, Alice sends the message $DNFFQGE$. Crack it and figure out the key that Alice used! (What kind of attack is this?)

Armin Straub
straub@southalabama.edu

**Solution.** This is a known plaintext attack.

Since $m+k=c$ (to be interpreted characterwise, modulo $26$, and with $k$ repeated as necessary), we can find $k$ simply as $k=c-m$.

For instance, since $A$ (value $0$!) got encrypted to $B$, the first letter of the key is $B$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $c$ | | | $B$ | $K$ | $N$ | $D$ | $K$ | $G$ | $B$ | $Q$ |
| $m$ | | $-$ | $A$ | $L$ | $L$ | $C$ | $L$ | $E$ | $A$ | $R$ |
| $k$ | | $=$ | $B$ | $Z$ | $C$ | $B$ | $Z$ | $C$ | $B$ | $Z$ |

We conclude that the key is $k=BZC$.

**Note.** Now, we can decrypt any future message that Alice sends using this key. For instance, we easily decrypt $DNFFQGE$ to $CODEREDD$ (using $m=c-k$).

All of the historical ciphers we have seen, including the substitution cipher that we will discuss shortly, fall apart completely under a known plaintext attack.

## Euler's theorem

**Example 28.** Compute $3^{1003} \pmod{101}$.

**Solution.** Since $101$ is a prime, $3^{100} \equiv 1 \pmod{101}$ by Fermat's little theorem.

Because $3^{100} \equiv 3^0 \pmod{101}$, this enables us to reduce exponents modulo $100$.

In particular, since $1003 \equiv 3 \pmod{100}$, we have $3^{1003} \equiv 3^3 = 27 \pmod{101}$.

Fermat's little theorem is a special case of Euler's theorem :

**Theorem 29. (Euler's theorem)** If $n \geqslant 1$ and $\gcd(a,n)=1$, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

**Proof.** Euler's theorem can be proved along the lines of our earlier proof of Fermat's little theorem. The only adjustment is to only start with multiples $ka$ where $k$ is invertible modulo $n$. There are $\phi(n)$ such residues $k$, and so that's where Euler's phi function comes in. Can you complete the proof? □

**Example 30.** What are the last two (decimal) digits of $3^{7082}$?

**Solution.** We need to determine $3^{7082} \pmod{100}$. $\phi(100) = \phi(2^2 5^2) = \phi(2^2)\phi(5^2) = (2^2-2^1)(5^2-5^1) = 40$.

Since $\gcd(3,100)=1$ and $7082 \equiv 2 \pmod{40}$, Euler's theorem shows that $3^{7082} \equiv 3^2 = 9 \pmod{100}$.

## Binary exponentiation

**Example 31.** Compute $3^{25} \pmod{101}$.

**Solution.** Fermat's little theorem is not helpful here.

Instead, we do **binary exponentiation**:

$3^2 = 9$, $3^4 = 81 \equiv -20$, $3^8 \equiv (-20)^2 = 400 \equiv -4$, $3^{16} \equiv (-4)^2 \equiv 16$, all modulo $101$

$25 = 16+8+1$ [Every integer $n \geqslant 0$ can be written as a sum of distinct powers of $2$ (in a unique way).]

Hence, $3^{25} = 3^{16} \cdot 3^8 \cdot 3^1 \equiv 16 \cdot (-4) \cdot 3 = -192 \equiv 10 \pmod{101}$.

**Example 32. (extra practice)** Compute $2^{20} \pmod{41}$.

**Solution.** $2^2 = 4$, $2^4 = 16$, $2^8 = 256 \equiv 10$, $2^{16} \equiv 100 \equiv 18$. Hence, $2^{20} = 2^{16} \cdot 2^4 \equiv 18 \cdot 16 = 288 \equiv 1 \pmod{41}$.

Or: $2^5 = 32 \equiv -9 \pmod{41}$. Hence, $2^{20} = (2^5)^4 \equiv (-9)^4 = 81^2 \equiv (-1)^2 = 1 \pmod{41}$.

**Comment.** Write $a = 2^{20} \pmod{41}$. It follows from Fermat's little theorem that $a^2 = 2^{40} \equiv 1 \pmod{41}$. The argument below shows that $a \equiv \pm 1 \pmod{41}$ [but we don't know which until we do the calculation].

The equation $x^2 \equiv 1 \pmod{p}$ is equivalent to $(x-1)(x+1) \equiv 0 \pmod{p}$ [b/c $(x-1)(x+1) = x^2-1$]. Since $p$ is a prime and $p \mid (x-1)(x+1)$, we must have $p \mid (x-1)$ or $p \mid (x+1)$. In other words, $x \equiv \pm 1 \pmod{p}$.

## Representations of integers in different bases

We are commonly using the **decimal system** of writing numbers. For instance:

$$1234 = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0.$$

$10$ is called the base, and $1, 2, 3, 4$ are the digits in base $10$. To emphasize that we are using base $10$, we will write $1234 = (1234)_{10}$. Likewise, we write

$$(1234)_b = 1 \cdot b^3 + 2 \cdot b^2 + 3 \cdot b^1 + 4 \cdot b^0.$$

In this example, $b > 4$, because, if $b$ is the base, then the digits have to be in $\{0, 1, ..., b-1\}$.

**Comment.** In the above examples, it is somewhat ambiguous to say whether $1$ or $4$ is the first or last digit. To avoid confusion, one refers to $4$ as the **least significant digit** and $1$ as the **most significant digit**.

**Example 33.** $25 = 16 + 8 + 1 = \boxed{1} \cdot 2^4 + \boxed{1} \cdot 2^3 + \boxed{0} \cdot 2^2 + \boxed{0} \cdot 2^1 + \boxed{1} \cdot 2^0$.
Accordingly, $25 = (11001)_2$.

While the approach of the previous example works well for small examples when working by hand (if we are comfortable with powers of $2$), the next example illustrates a more algorithmic approach.

**Example 34.** Express $49$ in base $2$.

**Solution.**

- $49 = 24 \cdot 2 + \boxed{1}$. Hence, $49 = (...1)_2$ where ... are the digits for $24$.
- $24 = 12 \cdot 2 + \boxed{0}$. Hence, $49 = (...01)_2$ where ... are the digits for $12$.
- $12 = 6 \cdot 2 + \boxed{0}$. Hence, $49 = (...001)_2$ where ... are the digits for $6$.
- $6 = 3 \cdot 2 + \boxed{0}$. Hence, $49 = (...0001)_2$ where ... are the digits for $3$.
- $3 = 1 \cdot 2 + \boxed{1}$. Hence, $49 = (...10001)_2$ where ... are the digits for $1$.
- $1 = 0 \cdot 2 + \boxed{1}$. Hence, $49 = (110001)_2$.

**Other bases.**
What is $49$ in base $3$? $49 = 16 \cdot 3 + \boxed{1}$, $16 = 5 \cdot 3 + \boxed{1}$, $5 = 1 \cdot 3 + \boxed{2}$, $1 = 0 \cdot 3 + \boxed{1}$. Hence, $49 = (1211)_3$.
What is $49$ in base $5$? $49 = (144)_5$.
What is $49$ in base $7$? $49 = (100)_7$.

**Example 35.** Bases $2$, $8$ and $16$ (binary, octal and hexadecimal) are commonly used in computer applications.

For instance, in JavaScript or Python, $0$b... means $(...)_2$, $0$o... means $(...)_8$, and $0$x... means $(...)_{16}$.
The digits $0, 1, ..., 15$ in hexadecimal are typically written as $0, 1, ..., 9, A, B, C, D, E, F$.
**Example.** FACE value in decimal? $(FACE)_{16} = 15 \cdot 16^3 + 10 \cdot 16^2 + 12 \cdot 16 + 14 = 64206$
**Practical example.** `chmod 664 file.tex` (change file permission)

664 are octal digits, consisting of three bits: $1 = (001)_2$ execute (x), $2 = (010)_2$ write (w), $4 = (100)_2$ read (r)
Hence, 664 means rw,rw,r. What is rwx,rx,-? 750
By the way, a fourth (leading) digit can be specified (setting the flags: setuid, setgid, and sticky).

Armin Straub
straub@southalabama.edu

## Modern ciphers

**Example 36.** For modern ciphers, we will change the alphabet from $A, B, ..., Z$ to $0, 1$. One of the most common ways of encoding text is **ASCII**.

In ASCII (American Standard Code for Information Interchange), each letter is represented using 8 bits (1 byte). Among the $2^8 = 256$ many characters are the usual letters, as well as common symbols.

For instance: space $= (20)_{16}$, "0"$=(30)_{16}$, $A = (41)_{16} = (0100, 0001)_2 = 65$, $a = (61)_{16} = (0110, 0001)_2 = 97$

See, for instance, http://www.asciitable.com for the full table.

**Example 37.** The new (8/2018) insignia of **FinCEN** features binary digits. What do they mean?

01000110 01101001 01101110 01000011 01000101 01001110 https://www.fincen.gov

**By the way.** If you ever have more than $\$10,000$ in foreign accounts, you must file a report to FinCEN.

## One-time pad

**Definition 38.** The "exclusive or" (XOR), often written $\oplus$, is defined bitwise:

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | 1 | 1 |
| $\oplus$ | 0 | 1 | 0 | 1 |
| $=$ | 0 | 1 | 1 | 0 |

**Note.** On the level of individual bits, this is just addition modulo 2.

**By the way.** Best thing about a boolean: even if you are wrong, you are only off by a bit.

**Example 39.** $1011 \oplus 1111 = 0100$

**Example 40.** Observe that $a \oplus b \oplus b = a$.

One way to see that is to think bitwise in terms of addition modulo 2. Then, $a + b + b = a + 2b \equiv a \pmod 2$.

A **one-time pad** works as follows. We use a key $k$ of the same length as the message $m$. Then the ciphertext is

$$c = E_k(m) = m \oplus k.$$

To decipher, we use $m = D_k(c) = c \oplus k$.

As the name indicates, we must never use this key again!

**Note.** Observe that encryption and decryption are the same routine.

**Comment.** If that is helpful, a one-time pad is doing exactly the same as the Vigenere cipher if we use a key of the same length as the message (also, we use $0, 1$ as our letters instead of the classical $A, B, ..., Z$).

**Example 41.** Using a one-time pad with key $k = 1100, 0011$, what is the message $m = 1010, 1010$ encrypted to?

**Solution.** $c = m \oplus k = 0110, 1001$

If a one-time pad (with perfectly random key) is used exactly once to encrypt a message, then **perfect confidentiality** is achieved (eavesdropping is hopeless).

Meaning that Eve intercepting the ciphertext can draw absolutely no conclusions about the plaintext (because, without information on the key, every text of the right length is actually possible and equally likely), see next example.

## Historical example: substitution cipher

**Example 42. (substitution cipher)** In a substitution cipher, the key $k$ is some permutation of the letters $A, B, \ldots, Z$. For instance, $k = FRA\ldots$. Then we encrypt $A \to F$, $B \to R$, $C \to A$ and so on. How large is the key space?

**Solution.** Key space has size $26! \approx 10^{26.6} \approx 2^{88.4}$, so a key can be stored using $89$ bits. That's actually a fairly large key space (for instance, DES has a key size of $56$ bits only). Too large to go through by brute force.

**However, still easy to break.** Since each letter is always replaced with the same letter, this cipher is susceptible to a **frequency attack**, exploiting that certain letters (and, more generally, letter combinations!) occur much more frequently in, say, English text than others. For instance, Lewand's book on Cryptology lists the following frequencies:

E: $12.7\%$, T: $9.1\%$, A: $8.2\%$, O: $7.5\%$, I: $7\%$, N: $6.7\%$, S: $6.3\%$, H: $6.1\%$, R: $6\%$, D: $4.3\%$, L: $4\%$, C: $2.8\%$, ...

The rarest letters are Q and Z with a frequency of about $0.1\%$ only. (The exact frequencies and precise ordering various between different sources and the body of text that the frequencies were obtained from.)

The most common letter pairs (digrams) are TH HE AN RE ER IN ON AT ND ST ES EN OF TE ED OR TI HI AS TO.

More information at: https://en.wikipedia.org/wiki/Letter_frequency

**Comment.** Note that the frequencies and even the ranking depend considerably on the source of text. For instance, using government telegrams, a military resource lists EN followed by RE, ER as the most frequent digrams. That same manual suggests SENORITA as a mnemonic to remember the most frequent letters.

http://www.umich.edu/~umich/fm-34-40-2/ (Field Manual 34-40-2, Department of the Army, 1990)

**Example 43.** It seems convenient to add the space as a 27th letter in the historic encryption schemes. Can you think of a reason against doing that?

**Solution.** In most texts, the space occurs more frequently and more regularly than any other letter. Adding it to the encryption schemes would make them even more susceptible to attacks.

**Example 44. (bonus challenge!)** You intercept the following message from Alice:

WHCUHFWXOWHUQXOMOMQVSQWAMWHCUHFXOLNWXQMQVSQWAWMQLN

Your experience tells you that Alice is using a substitution cipher. You also know that this message contains the word "secret". Can you crack it?

**Note.** In modern practice, it is not uncommon to know (or suspect) what a certain part of the message should be. For instance, PDF files start with "%PDF" (0x25504446).

See https://en.wikipedia.org/wiki/Magic_number_(programming) for more such instances.

(To collect a bonus point, send me an email within the next week with the plaintext and how you found it.)

## One-time pad (continued)

**Example 45.** A ciphertext only attack on the one-time pad is entirely hopeless. Explain why!

**Solution.** The attacker only knows $c = m \oplus k$. The attacker is unable to get any information on $m$, because every other message $m'$ (of the right length) could have resulted in the same ciphertext $c$.

Indeed, the key $k' = m' \oplus c$ encrypts $m'$ to $c$ as well (because $m' \oplus k' = m' \oplus (m' \oplus c) = c$). Moreover, every plaintext $m'$ is equally likely because it corresponds to a unique key.

The next example highlights the importance of only using the key once.

**Example 46. (attack on the two-time pad)** Alice made a mistake and encrypted the two plaintexts $m_1$, $m_2$ using the same key $k$. How can Eve exploit that?

**Solution.** Eve knows the two ciphertexts $c_1 = m_1 \oplus k$ and $c_2 = m_2 \oplus k$.

Hence, she can compute $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$.

This means that Eve knows $m_1 \oplus m_2$, which is information about the original plaintexts (no key involved!). That's a cryptographic disaster: Eve should never be able to learn *anything* about the plaintexts.

**In fact.** If the plaintexts are, say, English text encoded using ASCII then Eve very possibly can (almost) reconstruct both $m_1$ and $m_2$ from $m_1 \oplus m_2$. The reason for that is that the messages are expressed in ASCII, which means $8$ bits per character of text. However, the **entropy** (a measure for the amount of information in a message) of (longer) typical English text is frequently below $2$ bits per character.

Some details and beautiful graphical illustrations are given at:

http://crypto.stackexchange.com/questions/59/taking-advantage-of-one-time-pad-key-reuse

We saw in Example 45 that ciphertext only attacks on the one-time pad are entirely hopeless. What about other attacks?

Attacks like known plaintext or chosen plaintext don't apply if the key is only to be used once.

Yet, the one-time pad by itself provides **little protection of integrity**. The next example shows how tampering is possible without knowledge about the key.

**Example 47.** Alice sends an email to Bob using a one-time pad. Eve knows that and concludes that, per email standard, the plaintext must begin with `To: Bob`. Eve wants to tamper with the message and change it to `To: Boo`, for a light scare.

- Eve wants to change the 7th letter of the plain text $m$ from $b$ to $o$.
- Since $b$ is $0x62$ and $o$ is $0x6F$, we have $b \oplus o = 0x0D$. Hence, $b \oplus 0x0D = o$.
- Therefore, if $e = 0x\underbrace{000000000000}_{\text{6 characters}}0D00\ldots$, then $\underbrace{\text{“TO: Bob...”}}_{m} \oplus e = \underbrace{\text{“TO: Boo...”}}_{m'}$.
- Alice sends $c = m \oplus k$. If Eve changes the ciphertext $c$ to $c' = c \oplus e$, then Bob receives $c'$ and decrypts it to $c' \oplus k = \overbrace{\underbrace{m \oplus k}_{=c} \oplus e}^{c'} \oplus k = m \oplus e = m'$, which is what Eve intended.

Using the one-time pad presents several challenges, including:

- keys must not be reused (see Example 46)
- while perfectly protecting against eavesdropping (if done correctly), the one-time pad is not secure against tampering (see Example 47)
- key distribution and management

  Alice and Bob have to somehow exchange huge amounts of keys, so that, at a later time, they are able to communicate securely.
- for perfect confidentiality, the key must be perfectly random

  But how can we produce huge amounts of random bits?

  Especially, how to teach a deterministic machine like a computer to do that? Think about it! This is much more challenging that it may seem at first...

These issues make one-time pads difficult to use in practice.

**Historic comment.** During the Cold War, the "hot line" between Washington and Moscow apparently used one-time pads for secure communication.

**Example 48.** One thing that makes the one-time pad difficult to use is that the key needs to be the same length as the plaintext. What if we have a shorter key and just repeat it until it has the length we need?

That's essentially the Vigenere cipher (in a different alphabet).

**Solution.** Assuming the attacker knows the length of our key (if she doesn't she can just try all possibilities), this is equivalent to using the one-time pad several times with the same key. That should never be done! Even using a key twice means that we become susceptible to a ciphertext only attack (see Example 46).

So, repeating the key is a terrible idea. However, the idea to create a longer (random) key out of a shorter (random) key is good (we will discuss pseudorandom generators next).

Let us emphasize that, in order to be perfectly confidential, the key for a one-time pad must be chosen completely at random (otherwise, an attacker can make assumptions on the used keys).

Indeed, the need to generate random numbers shows in every modern cipher.

## Stream ciphers

Once we have a way to generate **pseudorandom numbers**, we can use the idea of the one-time pad to create a **stream cipher**.

Start with key of moderate size (say, 128 bits).

Use the key $k$ and a PRG (**pseudorandom generator**) to generate a much longer **pseudorandom keystream** $\mathrm{PRG}(k)$. Then encrypt $E_k(m) = m \oplus \mathrm{PRG}(k)$.

We lost perfect confidentiality. Security relies on choice of PRG (must be unpredictable).

As with the one-time pad, we must never reuse the same keystream! That does not mean that we cannot reuse the key: we can do that using a **nonce**: $E_k(m) = m \oplus \mathrm{PRG}((\mathrm{nonce}, k))$, where the seed is produced by combining the $\mathrm{nonce}$ and $k$ (for instance, just concatenating them).

The nonce is then passed (unencrypted) along with the message.

To make sure that we never reuse the same keystream, we must never use the same nonce with the same key.

**Remark.** A nonce can only be used once, as is in its name. Apparently, according to Urban Dictionary, it is also common as a British insult, roughly equivalent to wanker.

## How to generate random numbers?

Natural randomness is surprisingly difficult to harness.

You can for instance play around with a Geiger counter but our department is short on these and getting lots of random numbers is again challenging.

### Linear congruential generators

**(linear congruential generator)** Let $a, b, m$ be chosen parameters.
From the seed $x_0$, we produce the sequence $x_{n+1} \equiv a x_n + b \pmod{m}$.

The choice of $a, b, m$ is crucial for this to generate acceptable pseudorandom numbers.

For instance, glibc uses $a = 1103515245$, $b = 12345$, $m = 2^{31}$. (This is one of two implementations.) In that case, each $x_i$ is represented by precisely 31 bits. [Note that the choice of $m$ makes this very fast.]

https://en.wikipedia.org/wiki/Linear_congruential_generator

Linear congruential generators (LCG) are easy to predict and must not be used for cryptographic purposes. More generally, all polynomial generators are cryptographically insecure. They are still used in practice, because they are fast and easy to implement and have decent statistical properties. (For instance, our online homework is generated using random numbers, and there is no need for crypto-level security there.)

**Statistical trouble.** Can you see why the sequences produced by the glibc LCG alternate between even and odd numbers? (Similarly, other low bits are much less "random" than the higher bits.) Because of this defect, some programs (and other implementations of `rand()` based on LCGs) throw away the low bits entirely.

**Comment.** The particular choices of $a$ and $b$ in glibc are somewhat mysterious. See, for instance:

https://stackoverflow.com/questions/8569113/why-1103515245-is-used-in-rand

**Example 49.** Generate values using the linear congruential generator $x_{n+1} \equiv 5x_n + 3 \pmod{8}$, starting with the seed $x_0 = 6$.

**Solution.** $x_1 \equiv 1$, $x_2 \equiv 0$, $x_3 \equiv 3$, $x_4 \equiv 2$, $x_5 \equiv 5$, $x_6 \equiv 4$, $x_7 \equiv 7$, $x_8 \equiv 6$. This is the value $x_0$ again, so the sequence will now repeat. Note that we went through all 8 residues before repeating. Period $8$.

**Note.** Because $8 = 2^3$ we can represent each $x_i$ using exactly 3 bits. Then $x_1, x_2, x_3, \ldots = 1, 0, 3, \ldots$ corresponds to the bit stream $(001\ 000\ 011\ \ldots)_2$.

**Example 50. (extra)** Observe that the sequence produced by the linear congruential generator $x_{n+1} \equiv a x_n + b \pmod{m}$ must repeat, at the latest, after $m$ terms. (Why?!)

One can give precise conditions on $a, b, m$ to achieve a full period $m$. Namely, this happens if and only if $\gcd(b, m) = 1$ and $a - 1$ is divisible by all primes (as well as $4$) dividing $m$.

(a) Generate values using a linear congruential generator $x_{n+1} \equiv 2x_n + 1 \pmod{10}$, starting with the seed $x_0 = 5$. When do they repeat? Is that consistent with the mentioned condition?

(b) What are possible values for $a$ so that the LCG $x_{n+1} \equiv a x_n + 11 \pmod{100}$ has period $100$?

(c) glibc uses $a = 1103515245$, $b = 12345$, $m = 2^{31}$. After how many terms will the sequence repeat?

**Solution.**

(a) $x_1 \equiv 1$, $x_2 \equiv 3$, $x_3 \equiv 7$, $x_4 \equiv 5$. This is the value $x_0$ again, so the sequence will repeat. Period $4$.
[The period is less than $10$. This is as predicted by the mentioned condition, because $a - 1$ is not divisible by $2$ and $5$.]

(b) We need that $a - 1$ is divisible by $4$ and $5$. Equivalently, $a \equiv 1 \pmod{20}$. Hence, possible values are $a = 1, 21, 41, 61, 81$.

(c) Clearly, $\gcd(b, m) = 1$. Also, $a - 1$ is divisible by $4$ (and no primes other than $2$ divide $m$). Hence, for every seed, values repeat only after going through all $2^{31}$ residues.

**Example 51.** Let's use the PRG $x_{n+1} \equiv 5x_n + 3 \pmod 8$ as a stream cipher with the key $k = 4 = (100)_2$. The key is used as the seed $x_0$ and the keystream is $\mathrm{PRG}(k) = x_1 x_2 \ldots$ (where each $x_i$ is 3 bits). Encrypt the message $m = (101\ 111\ 001)_2$.

**Solution.** We first use the PRG with seed $x_0 = k$ to produce the keystream $\mathrm{PRG}(k) = 7,\ 6,\ 1,\ \ldots = (111\ 110\ 001\ \ldots)_2$.

We then encrypt and get $c = E_k(m) = m \oplus \mathrm{PRG}(k) = (101\ 111\ 001)_2 \oplus (111\ 110\ 001)_2 = (010\ 001\ 000)_2$.

**Decryption.** Observe that decryption works in the exact same way:

$D_k(c) = c \oplus \mathrm{PRG}(k) = (010\ 001\ 000)_2 \oplus (111\ 110\ 001)_2 = (101\ 111\ 001)_2$.

**Note.** The keystream continues as $\mathrm{PRG}(k) = 7, 6, 1, 0, 3, 2, 5, 4, \ldots$ At this point it repeats itself because we obtained the value $4$, which was our seed. Since the state of this PRG only depends on the value of $x_n$, and there are $8$ possible values for $x_n$, the period $8$ is the longest possible. The previous (extra) example gave conditions on the PRG that guarantee that the period is as long as possible.

**Example 52.** Can you think of a way in which the numbers produced by a linear congruential generator differ from truly random ones?

**Solution.** An easy observation for our small examples is the following: by construction, $x_{n+1} \equiv a x_n + b \pmod m$, individual values don't repeat unless a full period is reached and everything repeats. Truly random numbers do repeat every now and then (however, if $m$ is large, then this observation is not exactly practical).

Of course, knowing the parameters $a,\ b,\ m$, the numbers generated by the PRG are terribly **predictable**. Knowing just one number, we can produce all the next ones (as well as the ones before). A PRG that is safe for cryptographic purposes should not be predictable like that! (See next example.)

The next example illustrates the vulnerability of stream ciphers, based on predictable PRGs.

Recall that it is common to know or guess pieces of plaintexts; for instance, every PDF begins with $\%\mathrm{PDF}$.

**Example 53.** Eve intercepts the ciphertext $c = (111\ 111\ 111)_2$. It is known that a stream cipher with PRG $x_{n+1} \equiv 5x_n + 3 \pmod 8$ was used for encryption. Eve also knows that the plaintext begins with $m = (110\ 1\ldots)_2$. Help her crack the ciphertext!

**Solution.** Since $c = m \oplus \mathrm{PRG}$, we learn that the initial piece of the keystream is $\mathrm{PRG} = m \oplus c = (110\ 1\ldots)_2 \oplus (111\ 1\ldots)_2 = (001\ 0\ldots)_2$. Since each $x_n$ is 3 bits, we conclude that $x_1 = (001)_2 = 1$.

Because the PRG is predictable, we can now recreate the entire keystream! Using $x_{n+1} \equiv 5x_n + 3 \pmod 8$, we find $x_2 \equiv 0$, $x_3 \equiv 3$, ... In other words, $\mathrm{PRG} = 1, 0, 3, \ldots = (001\ 000\ 011\ \ldots)_2$.

Hence, Eve can decrypt the ciphertext and obtain $m = c \oplus \mathrm{PRG} = (111\ 111\ 111)_2 \oplus (001\ 000\ 011)_2 = (110\ 111\ 100)_2$.

**Review.**

- A **pseudorandom generator** (PRG) takes a seed $x_0$ and produces a stream $\mathrm{PRG}(x_0) = x_1 x_2 x_3 \ldots$ of numbers, which should "look like" random numbers.

  For cryptographic purposes, these numbers should be indistinguishable from random numbers. Even for somebody who knows everything about the PRG except the seed. (See Example 57.)

- Once we have a PRG, we can use it as a **stream cipher**: Using the key $k$, we encrypt $E_k(m) = m \oplus \mathrm{PRG}(k)$. [Here, the key stream $\mathrm{PRG}(k)$ is assumed to be in bits.]

  As with the one-time pad, we must never reuse the same keystream!

- To reuse the key, we can use a **nonce**: $E_k(m) = m \oplus \mathrm{PRG}((\mathrm{nonce}, k))$, where the seed is produced by combining the nonce and $k$ (for instance, just concatenating them).

  The nonce is then passed (unencrypted) along with the message.

  To never reuse the same keystream, we must never use the same nonce with the same key.

## Linear feedback shift registers

Here is another basic idea to generate pseudorandom numbers:

**(linear feedback shift register (LFSR)** Let $\ell$ and $c_1, c_2, \ldots, c_\ell$ be chosen parameters. From the seed $(x_1, x_2, \ldots, x_\ell)$, where each $x_i$ is one bit, we produce the sequence

$$x_{n+\ell} \equiv c_1 x_{n+\ell-1} + c_2 x_{n+\ell-2} + \ldots + c_\ell x_n \pmod 2.$$

This method is particularly easy to implement in hardware (see Example 55), and hence suited for applications that value speed over security (think, for instance, encrypted television).

**Example 54.** Which sequence is generated by the LFSR $x_{n+2} \equiv x_{n+1} + x_n \pmod 2$, starting with the seed $(x_1, x_2) = (0, 1)$?

**Solution.** $(x_1, x_2, x_3, \ldots) = (0, 1, 1, 0, 1, 1, \ldots)$ has period $3$.

**Note.** Observe that the two previous values determine the state, so there are $2^2 = 4$ states of the LFSR. The state $(0, 0)$ is special (it generates the zero sequence $(0, 0, 0, 0, \ldots)$), so there are $3$ other states. Hence, it is clear that the generated sequence has to repeat after at most $3$ terms.

**Comment.** Of course, if we don't reduce modulo $2$, then the sequence $x_{n+2} = x_{n+1} + x_n$ generates the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

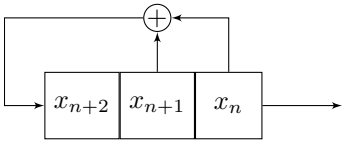
**Example 55.** Which sequence is generated by the LFSR $x_{n+3} \equiv x_{n+1} + x_n \pmod 2$, starting with the seed $(x_1, x_2, x_3) = (0, 0, 1)$? What is the period?

[Let us first note that the LFSR has $2^3 = 8$ states. Since the state $(0, 0, 0)$ remains zero forever, $7$ states remain. This means that the generated sequence must be periodic, with period at most $7$.]

**Solution.** $(x_1, x_2, x_3, \ldots) = (0, 0, 1, 0, 1, 1, 1, 0, 0, 1, \ldots)$ has period $7$.

Again, this is not surprising: $3$ previous values determine the state, so there are $2^3 = 8$ states. The state $(0, 0, 0)$ is special, so there are $7$ other states.

Note that this LFSR can be implemented in hardware using three registers (labeled $x_n, x_{n+1}, x_{n+2}$ in the sketch to the right). During each cycle, the value of $x_n$ is read off as the next value produced by the LFSR.




**Note.** In the part $0, 0, 1, 0, 1, 1, 1$ that repeats, the bit $1$ occurs more frequently than $0$.

The reason for that is that the special state $(0, 0, 0)$ cannot appear.

For the same reason, the bit $1$ will always occur slightly more frequently than $0$ in LFSRs. However, this becomes negligible if the period is huge, like $2^{31} - 1$ in Example 56.

Armin Straub
straub@southalabama.edu

**Example 56.** The recurrence $x_{n+31} \equiv x_{n+28} + x_n \pmod 2$, with a nonzero seed, generates a sequence that has period $2^{31} - 1$.

Note that this is the maximal possible period: this LFSR has $2^{31}$ states. Again, the state $(0, 0, ..., 0)$ is special (the entire sequence will be zero), so that there are $2^{31} - 1$ other states. This means that the terms must be periodic with period at most $2^{31} - 1$.

**Comment.** glibc (the second implementation) essentially uses this LFSR.

**Advanced comment.** One can show that, if the characteristic polynomial $f(T) = x^\ell + c_1 x^{\ell-1} + c_2 x^{\ell-2} + ... + c_\ell$ is irreducible modulo $2$, then the period divides $2^\ell - 1$. Here, $f(T) = T^{31} + T^{28} + 1$ is irreducible modulo $2$, so that the period divides $2^{31} - 1$. However, $2^{31} - 1$ is a prime, so that the period must be exactly $2^{31} - 1$.

**Example 57.** Eve intercepts the ciphertext $c = (1111\ 1011\ 0000)_2$ from Alice to Bob. She knows that the plaintext begins with $m = (1100\ 0...)_2$. Eve thinks a stream cipher using a LFSR with $x_{n+3} \equiv x_{n+2} + x_n \pmod 2$ was used. If that's the case, what is the plaintext?

**Solution.** The initial piece of the keystream is $\mathrm{PRG} = m \oplus c = (1100\ 0...)_2 \oplus (1111\ 1...)_2 = (0011\ 1...)_2$.

Each $x_n$ is a single bit, and we have $x_1 \equiv 0$, $x_2 \equiv 0$, $x_3 \equiv 1$. The given LFSR produces $x_4 \equiv x_3 + x_1 \equiv 1$, $x_5 \equiv x_4 + x_2 \equiv 1$, $x_6 \equiv 0$, $x_7 \equiv 1$, and so on. Continuing, we obtain $\mathrm{PRG} = x_1 x_2 ... = (0011\ 1010\ 0111)_2$.

Hence, the plaintext would be $m = c \oplus \mathrm{PRG} = (1111\ 1011\ 0000)_2 \oplus (0011\ 1010\ 0111)_2 = (1100\ 0001\ 0111)_2$.

A PRG is **predictable** if, given the stream it outputs (but not the seed), one can with some precision predict what the next bit will be (i.e. do better than just guessing the next bit).

In other words: the bits generated by the PRG must be indistinguishable from truly random bits, even in the eyes of someone who knows everything about the PRG except the seed.

The PRGs we discussed so far are entirely predictable because the state of the PRGs is part of the random stream they output.

For instance, for a given LFSR, it is enough to know any $\ell$ consecutive outputs $x_n, x_{n+1}, ..., x_{n+\ell-1}$ in order to predict all future output.

We have seen two simple examples of PRGs so far:

- linear congruential generators $x_{n+1} \equiv a x_n + b \pmod{m}$
- LFSRs $x_{n+\ell} \equiv c_1 x_{n+\ell-1} + c_2 x_{n+\ell-2} + \ldots + c_\ell x_n \pmod{2}$

Of course, we could also combine LFSRs and linear congruential generators (i.e. look at recurrences like for LFSRs but modulo any parameter $m$).

However, much of the appeal of an LFSR comes from its extremely simple hardware realization, as the sketch in Example 55 indicates.

**Example 58. (extra)** One can also consider nonlinear recurrences (it mitigates some issues). Our book mentions $x_{n+3} \equiv x_{n+2} x_n + x_{n+1} \pmod{2}$. Generate some numbers.

**Solution.** For instance, using the seed $0, 0, 1$, we generate $\overbrace{0, 0, 1}^{\text{seed}}, 0, 1, 1, 1, 0, 1, \ldots$ which now repeats (with period $4$) because the state $1, 0, 1$ appeared before. Observe that the generated sequences is only what is called eventually periodic (it is not strictly periodic because $0, 0, 1$ never shows up again).

**Example 59.** Suppose we have two PRGs that output bits. The first repeats after $14$ bits, the second after $18$ bits. After how many bits do they repeat simultaneously?

What if the two PRGs repeat after $13$ and $17$ bits instead?

**Solution.** Note that the first PRG again repeats after $28$ bits, after $42$ bits and, in general after $14m$ bits where $m$ is any positive integer. Likewise, the second PRG repeats after $18m$ bits where $m$ is any positive integer.

Therefore, both PRGs repeat simultaneously after $\operatorname{lcm}(14, 18) = \frac{14 \cdot 18}{2} = 126$ bits.

**Review.** Here, $\operatorname{lcm}$ is the **least common multiple**. We can always compute the $\operatorname{lcm}$ through the Euclidean algorithm by using $\operatorname{lcm}(a, b) = \frac{a \cdot b}{\gcd(a, b)}$.

If the two PRGs repeat after $13$ and $17$ bits instead, then they repeat simultaneously after $\operatorname{lcm}(13, 17) = 13 \cdot 17 = 221$ bits.

**Comment.** Certain cicadas spend more than 99% of their life underground as nymphs and only emerge as adults for 4-6 weeks. Interestingly, this life cycle is highly synchronized: cicadas of one species in a region appear all at once. In 2024, "Brood XIII" and "Brood XIX" co-emerged. These emerge every 17 and 13 years, respectively. Therefore this co-emergence is a rare event that only happens every 221 years (though, the same thing happened 2015 with two different broods).

https://en.wikipedia.org/wiki/Periodical_cicadas

**Example 60. (bonus!)** Eventually the output of the baby CSS in Example 61 has to repeat (though it need not be perfectly periodic; see Example 58). Once it repeats, what is the period?

**Note.** The state of the system is determined by $3 + 4 + 1 = 8$ bits (3 bits for LFSR-1, 4 bits for LFSR-2, and 1 bit for the carry). Hence, there are $2^8 = 256$ many states. Since the state with everything $0$ is again special, that means that after at most $255$ steps our PRG will reach a state it has been in before. At that point, everything will repeat.

(To collect a bonus point, send me an email within the next week with the period and how you found it.)

Armin Straub
straub@southalabama.edu

## Combining two LFSRs to get the CSS (content scramble system)

A popular way to reduce predictability is to combine several LFSRs (in a nonlinear fashion):

**Example 61.** The CSS (content scramble system) is based on 2 LFSRs and used for the encryption of DVDs. Before discussing the actual CSS let us consider a baby version of CSS. Our PRG uses the LFSR $x_{n+3} \equiv x_{n+1} + x_n \pmod 2$ as well as the LFSR $x_{n+4} \equiv x_{n+2} + x_n \pmod 2$. The output of the PRG is the output of these two LFSRs added with carry.

Adding with carry just means that we are adding bits modulo $2$ but add an extra $1$ to the next bits if the sum exceeded $1$. This is the same as interpreting the output of each LFSR as the binary representation of a (huge) number, then adding these two numbers, and outputting the binary representation of the sum.

If we use $(0,0,1)$ as the seed for LFSR-1, and $(0,1,0,1)$ for LFSR-2, what are the first 10 bits output by our PRG?

**Solution.** With seed $0,0,1$ LSFR-1 produces $0,1,1,1,0,0,1,0,1,1,...$

With seed $0,1,0,1$ LSFR-2 produces $0,0,0,1,0,1,0,0,0,1,...$

We now add these two:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | ⋯ |
| + | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | ⋯ |
| carry | | | | | 1 | | | | | 1 | |
| | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | ⋯ |

Hence, the output of our PRG is $0,1,1,0,1,1,1,0,1,0,....$

**Important comment.** Make sure you realize in which way this CSS PRG is much less predictable than a single LFSR! A single LFSR with $\ell$ registers is completely predictable since knowing $\ell$ bits of output (determines the state of the LFSR and) allows us to predict all future output. On the other hand, it is not so simple to deduce the state of the CSS PRG from the output. For instance, the initial $(0,1,...)$ output could have been generated as $(0,0,...)+(0,1,...)$ or $(0,1,...)+(0,0,...)$ or $(1,0,...)+(1,0,...)$ or $(1,1,...)+(1,1,...)$.

[In this case, we actually don't learn anything about the registers of each individual LFSR. However, we do learn how their values have to match up. That's the correlation that is exploited in **correlation attacks**, like the one described next class for the actual CSS scheme.]

**Advanced comment.** Is the carry important? Yes! Let $a_1, a_2, ...$ and $b_1, b_2, ...$ be the outputs of LFSR-1 and LFSR-2. Suppose we sum without carry. Then the output is $a_1+b_1, a_2+b_2, ...$ (with addition mod $2$). If Eve assigns variables $k_1$, $k_2$, ..., $k_7$ to the $3+4$ seed bits (the key in the stream cipher), then the output of the combined LFSR will be linear in these seven variables (because the $a_i$ and $b_i$ are linear combinations of the $k_i$). Given just a few more than $7$ output bits, a little bit of linear algebra (mod $2$) is therefore enough to solve for $k_1, k_2, ..., k_7$.

On the other hand, suppose we include the carry. Then the output is $a_1+b_1, a_2+b_2+a_1b_1, ...$ (note how $a_1b_1$ is $1 \pmod 2$ precisely if both $a_1$ and $b_1$ are $1 \pmod 2$, which is when we have a carry). This is not linear in the $a_i$ and $b_i$ (and, hence, not linear in the $k_i$), and we cannot use linear algebra to solve for $k_1, k_2, ..., k_7$ as before.

**Example 62.** In each case, determine if the stream could have been produced by the LFSR $x_{n+5} \equiv x_{n+2} + x_n \pmod 2$. If yes, predict the next three terms.

(STREAM-1) $...,1,0,0,1,1,1,1,0,1,...$ (STREAM-2) $...,1,1,0,0,0,1,1,0,1,...$

**Solution.** Using the LFSR, the values $1,0,0,1,1$ are followed by $1,1,1,0,...$ Hence, STREAM-1 was not produced by this LFSR.

On the other hand, using the LFSR, the values $1,1,0,0,0$ are followed by $1,1,0,1,1,1,0,...$ Hence, it is possible that STREAM-2 was produced by the LFSR (for a random stream, the chance is only $1/2^4 = 6.25\%$ that 4 bits matched up). We predict that the next values are $1,1,0,...$

**Comment.** This observation is crucial for the attack on CSS described in Example 63.