Computer Security and Privacy (COM-301)

Introduction to applied cryptography

Some slides/ideas adapted from: Carmela Troncoso, George Danezis, Yoshi Kohno

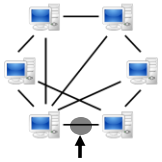
Plan of the Day

- Module 1 – Introduction
- Module 2 – Adversaries in Cryptography
- Module 3 – Perfect Secrecy and One-Time-Pad
- Module 4 – From OTP to Stream Ciphers
- Module 5 – Public Key Cryptography

Module 1: Introduction

4

Why cryptography matters?



Data in transit

Data at rest

- **Data in Transit:** Securing communication (e.g., HTTPS, VPNs).
- **Data at Rest:** Securing stored information (e.g., encrypted backups).

What can we do with cryptography

Cryptography is also the building block for:

- **Integrity:** Ensuring data hasn't been tampered with (Hashing/Signatures).
- **Authentication:** Verifying identity (Protocols).
- But also **Anonymity**, etc.

In this first lecture we will focus on the basics of using cryptography for confidentiality.

The Core Problem For Today

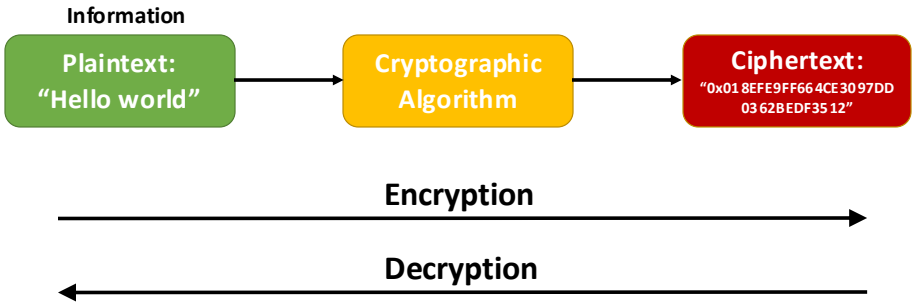
Secure Communication over an Insecure Channel

We need a way for Alice to send a message to Bob, so that Eve (the adversary) cannot read it.

Note: Confidentiality for data at rest can be seen as a particular case, where Alice = Bob.

The origins of cryptography: the quest for confidentiality

Confidentiality: information cannot be accessed by unauthorized parties



Encryption cannot be reversed without a KEY

Encryption algorithm: cryptographic instructions that convert a plaintext message into ciphertext

Decryption algorithm: cryptographic instructions that convert a ciphertext message into plaintext

Both encryption and decryption algorithm require a **key**

Cryptography as Functions

Encryption/Decryption

- **Plaintext (M):** Original message.
- **Key (K):** Secret value.
- **Ciphertext (C):** Scrambled message.

$$C = E_K(M) \quad \text{and} \quad M = D_K(C)$$

Cryptographic algorithms for confidentiality

1. **Get a key shared between Alice and Bob.**

2. Encrypt message **m -> Enc(k,m)**

3. Send encrypted message **Enc(k,m)**

4. Decrypt message **Dec(k,m) -> m**

10

The Core Requirement - Invertibility

The decryption function D_K *must* be the **invert** of E_K:

$$\forall \mathrm{K}, M, D_K\big(E_K(M)\big) = M$$

If not, Bob cannot recover the original message M.

SECURITY REQUIREMENT: To provide security, these functions must also be **hard to invert** *without* knowing the key K.

Hard to Invert: The Key Space

- **Concrete Meaning:** To find the plaintext (M), in an ideal cryptosystem, the only viable attack for the adversary (Eve) must be to **try every single possible secret key** K (a **brute-force attack**).

- **Goal:** A secure scheme forces the attacker into this situation.

Example 1: Caesar's Cipher

- **Encryption:** Shift each letter by a fixed number (K).
- **Decryption:** Shift each letter by a fixed number ($-K$).
- **Key Space:** Only 25 possible keys.

Plaintext: `ATTACK AT DAWN` $\xrightarrow{K=3}$ Ciphertext: `DWWDFN DW GDZQ`

Encrypt Decrypt

Security Insight: $\log_2(25) \approx \mathbf{4.6}$ bits of security. This is **far too small** for real-world use.

While 25 different keys is very small, and would never be used as it would be trivial to break it by a brute force attack, this scheme is pedagogically interesting to introduce a first attack.

Cryptanalysis: The Flaw

- Eve knows that Alice wrote a long-enough message for Bob in English.
- She knows that English most frequent letter is ‘e’.
- She can then simply identify the most frequent letter in the ciphertext: it likely corresponds to the letter ‘e’.
- She can then directly recover the key (a number between 0 and 25). This is a **frequency analysis attack**.

She can look at a long-enough ciphertext sent to Bob, and identify the most common letter in the ciphertext. That allows her to identify which likely one key out of the 25 keys was used by Alice.

More than 25 Keys: The Substitution Cipher

- Alternative encryption/decryption strategy: **Permuting the Alphabet**
- **Operation:** Each letter is mapped to a unique, different letter, defined by a **permutation** of the 26 letters.

Key: ABCDEFGHIJKLMNOPQRSTUVWXYZ
HOWBUGIACRYEVZXPJQMSNTFDKL

Encrypt/Decrypt: substitute by opposite letter

hello world ⟶ aueex fxqeb

- **Key Space:** 26! (26 factorial) possible keys.

Maybe the problem was 25 different keyswe can go for bigger number of keys!

From 25 Keys to ≈4×10^26 Keys

- $26!$ **(Number of Keys):** ≈ 403 septillion keys (4.03×10^{26}).
- **Bit Security:** $\log_2(26!) \approx \mathbf{88.4}$ bits of security?
- **Insight:** This key space might look like it is big enough, but this cipher is **still easily broken** by the same frequency analysis.

Cryptanalysis: The Flaw, Again

- The same **frequency analysis attack** is effective against the Substitution Cipher.
- **Vulnerability:** The cipher preserves the statistical properties of the plaintext language (e.g., single-letter frequencies).
- If Eve (the attacker) has a long enough ciphertext C, statistical analysis will break the cipher, rendering the full $26!$ key space **irrelevant**.

- **Goal of the adversary**: Notice that the goal of the adversary might not be necessarily to reccover the full key, depending on the scenario, it might only attempt to recover part of the key (the mapping for a subset of letters).
- Sometimes the goal of the attacker is even smaller: Eve might want only to know whether Alice sent the message "Yes" or "No" to Bob.

Concretely, imagine Alice encrypted the entire Harry Potter book series (only a few MB of data) using a key K, to send to Bob. Doing statistical analysis over this ciphertext, Eve can find the most common letter, and do a good guess for each letter: what letter is it sent to (starting with looking at the most common letter). Maybe, this won't work at 100%, but Eve can enumerate substitution that are the most likely, and using an english dictionary, Eve can rule out the keys that produce plaintext that are not valid english. In the end Eve will not have to test all the keys (far from it), and within a few tries she will find the key K that was used to encrypt the message.

Bits of Security Versus Key Space Size

- **Goal:** An N-bit key should offer security as close to N bits as possible (i.e., require 2^N attempts).
- **The Break:** If a 1024-bit key is broken with only 2^{40} attempts (40 bits of security), the algorithm is considered broken.
- **Caesar Lesson:** The 25-key space was cleverly reduced to **1 possibility**, similarly the ≈ 88 bits of the substitution cipher was reduced to just a few tries.

These were example of Bad cryptography. Before we take a look at much better cryptography encryption and decryption functions, let's discuss different adversarial models, as we should always do when studying computer security.

Module 2: Adversaries in Cryptography

19

Eve's Power: Passive Eavesdropping

In Caesar's (and the substitution) Cipher, Eve's capabilities were very limited:

- She observed a long-enough ciphertext C that corresponds to the encryption of a piece of English.
- (And some compute power to test a few keys).

This is not the only possible adversary model.

In the first lecture of the class we put emphasis on the importance to define the threat model, i.e., what the adversary can do. In this module we elaborate on the different typical threat model, why they are justified, and intuitively why they are different, through our little substitution example.

Other Security Models: Types of Adversaries

The security of a scheme is always measured relative to the power of the **adversary (Eve)**.

- **Passive Eavesdropper:** Eve can only *read* the ciphertext (the model we had until now).
- **More Active Attacker:** Eve can somehow influence the system by for example convincing Alice to send to Bob very specific messages, observing the precise time it takes Bob to run its algorithm, etc… anything that can go beyond just observing communication.

So far, the attacker was passive, but we have seen that we should be thinking about active adversaries as well: what Eve could do if we were to be given the opportunities to convince Alice to send Bob specific messages, get access to see Bob received messages for a little bit (looking at Bob's screen for 2 minutes?), or other situations.

Active Model 1: Known Plaintext Attack (KPA)

- **Scenario:** Eve is given access to multiple pairs (M, C) corresponding to messages and their corresponding ciphertext, all encrypted with the secret key K.
- **Goal:** From these pairs, she is trying to guess the key K to decrypt other messages.

Why is KPA a Realistic Model?

- **Standard Headers:** Many encrypted messages contain predictable headers (e.g., “From:”, timestamps).
- **Malicious Guessing:** If Eve guesses a message (or a part of a message) and observes its encryption, she has an M/C pair.

Why is KPA is a realistic model that we should protect against? It is very common that the encryption algoritms are used on very structured data, for example encrypting an entire structure packet that contains an explicit destination, or a timestamp.

KPA and the Substitution Cipher

- **Vulnerability:** Under a KPA, the substitution cipher becomes **even more broken**.
- **Why?** If Eve knows that $M =$ 'THE QUICK BROWN FOX', she immediately recovers the mappings for multiple letters, drastically reducing the effective key space, without even needing to do a statistical analysis study!

If later Alice uses the key K, a significant part of the key has already been leaked through the known plaintext/ciphertext key, using that and an English dictionary Eve can likely very easily recover with very few tries of keys.

Active Model 2: Chosen Plaintext Attack (CPA)

- **Scenario:** Eve can somehow convince Alice (or the system) to encrypt arbitrary messages (chosen by Eve) with the secret key.
- **More Abstractly:** For any chosen plaintext m, Eve can get access to $E_K(m)$.
- We often say that Eve is assumed to have access to an **Encryption Oracle**.

In the previous scenario, Eve did not get to choose which message, ciphertext pair she was allowed to observe. In this model, Eve actually get to choose messages m, and get the corresponding pair (m,enc(k,m)), so CPA is a strictly stronger model than KPA.

Why is CPA a realistic model?

We can imagine two simple scenarios.

- **Alternative Scenario:** Think of a secure messaging service (like Signal or WhatsApp) that uses the sender's key to encrypt the message. If Eve gets limited access to Alice' s phone for a few minutes, she can maybe send **any** piece of data she chooses tot Bob, to generate an (M, C) pair.
- **Encryption API:** Eve might interact with an encrypted API endpoint where she provides data and receives the encrypted output.

CPA and the Substitution Cipher

- **Complete Break:** The substitution cipher is **COMPLETELY broken** by a CPA.
- **Attack:** Eve chooses a plaintext that reveals the entire key: M = 'abcdefghijklmnopqrstuvwxyz'.
- **Result:** The resulting ciphertext C is the entire key (the full permutation mapping), in just *one* query!

Practical Threat Model: Side-Channel Attacks

- **Scenario:** Eve is allowed to not only access an ideal oracle, but also measure physical characteristics of the actual implementation while it is encrypting/decrypting.
- **Timing Attack:** Eve measures **how long** it takes for a given message to get encrypted or a ciphertext to be decrypted.
- **Power Analysis:** Eve observes the **energy consumed** by the device (e.g., a credit card) doing the crypto.

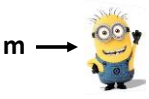
Why are Side-Channel a Realistic Model?

- **Malicious Ownership:** You often own a device (like a phone, smart card, or embedded chip) that holds keys belonging to another principal (e.g., Google, your bank) that are meant to be unreadable to you.
- **Concrete Example (DRM):** The decryption key for Digital Rights Management (DRM) content (like a movie or song) is embedded in your media player or console. By measuring power or timing signals, an attacker could potentially extract that secret key.

These are just a few adversary models, and we might describe new adversary models as needed in different exercises, always following the general principle of the class, that security should always be studied under a strategic adversary that is given limited well-defined resources.

Module 3: Perfect Secrecy and One-Time-Pad

Obtaining perfect secrecy: One Time Pad (OTP)



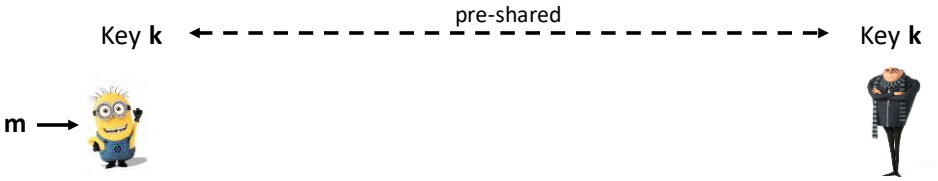
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: $a \oplus b \oplus b = a$)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



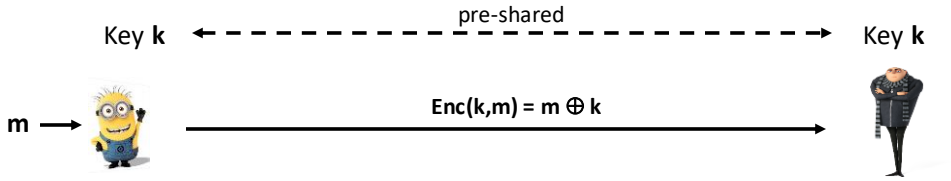
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: $a \oplus b \oplus b = a$)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



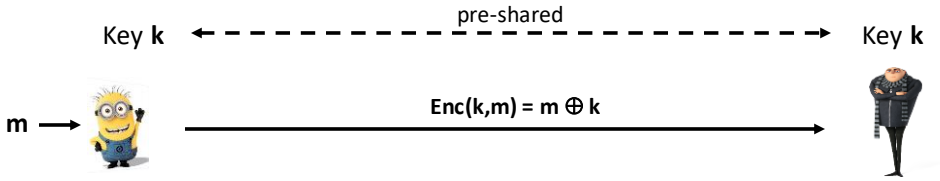
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: a⊕b⊕b=a)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



Message	YEAH (ASCII Hex: 59454148)
Binary	01011001010001010100000101001000
OTP-Key ⊕	01110101000111010100101001001010
Encryption	00101100010110000000010110000010

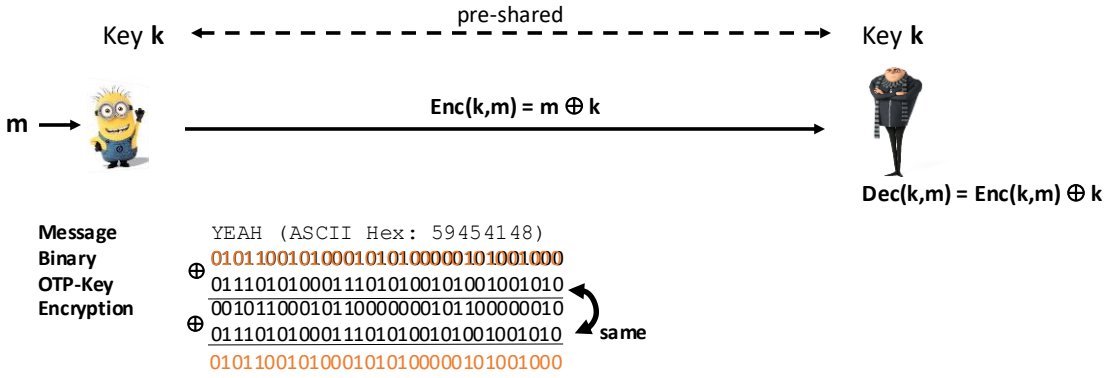
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: a⊕b⊕b=a)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



Message	YEAH (ASCII Hex: 59454148)
Binary	01011001010001010100001010010000
⊕ **OTP-Key**	01110101000111010100101001001010
Encryption	00101100010110000000101100000010
⊕	01110101000111010100101001001010 same
	01011001010001010100000101001000

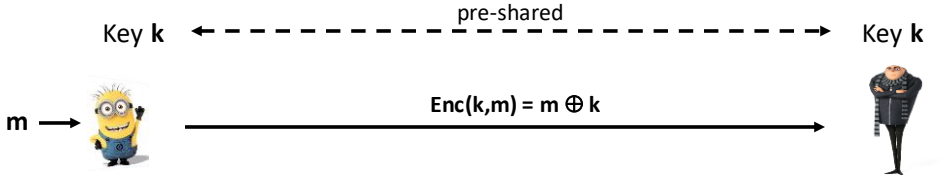
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

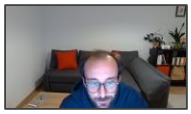
To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: a⊕b⊕b=a)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



Message	YEAH (ASCII Hex: 59454148)		NOPE (ASCII Hex: 4e4f5045)
Binary	01011001010001010100000101001000		01001110010011110101000001000101
OTP-Key ⊕	01110101000111010100101001001010	same ⟷ ⊕	01110101000111010100101001001010
Encryption	00101100010110000000101100000010		00111011010100100001101000001111



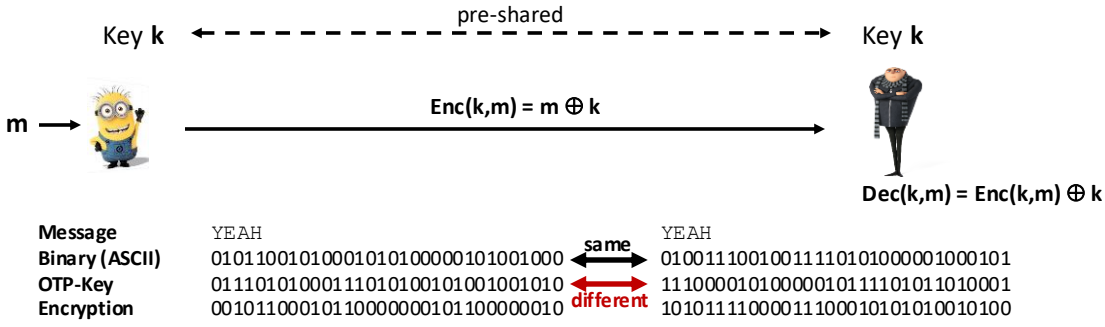
How to remove frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key. To decrypt one XORs the encrypted message with the key (Recall: a⊕b⊕b=a)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits as long as the message



One Time Pads are call like this because they should **not** be reused. If you reuse them, even though you cannot recover the full message, you can recover information about plaintexts. This information can be used to inform frequency analysis and help recover the messages themselves.

In particular, for ASCII characters one can use that spaces start by 00 and letters by 01 to detect whether a one of the messages has a space:

2 letters = 01 ⊕ 01 = 00
2 spaces = 00 ⊕ 00 = 00
Letter + space = 01 ⊕ 00 = 01

Proof of Security (Back to AICC1)

Why is the OTP *Perfectly* Secure?

- For any ciphertext C, **every possible plaintext M is equally likely**.
- The key K is uniformly random, so C gives the adversary **zero new information** about M.

Formally (Perfect Secrecy): $\forall m\ c\ , \mathrm{P}(M = \mathrm{m} | E_K(\mathrm{m}) = \mathrm{c}) = \mathrm{P}(M = \mathrm{m})$

$$\mathrm{P}(M = \mathrm{m} | E_K(\mathrm{m}) = \mathrm{c}) = \frac{\mathrm{P}(M=\mathrm{m} \cap E_K(m)=c))}{\mathrm{P}(E_K(m)=c))} = \frac{\mathrm{P}(M=\mathrm{m} \cap (m \text{ xor } \mathrm{K})=c))}{\mathrm{P}(E_K(m)=c))}$$

$$= \frac{\mathrm{P}(M = \mathrm{m} \cap \mathrm{K} = \mathrm{m} \text{ xor } \mathrm{c})}{\mathrm{P}(\mathrm{K} = \mathrm{m} \text{ xor } \mathrm{c})}$$

$$= \frac{\mathrm{P}(M = \mathrm{m}) * \mathrm{P}(\mathrm{K} = \mathrm{m} \text{ xor } \mathrm{c})}{\mathrm{P}(\mathrm{K} = \mathrm{m} \text{ xor } \mathrm{c})}$$
$$= P_{\mathrm{K}}(M = \mathrm{m})$$

We have many plaintexts m and each one has a certain probability P(M = m) of being used by Alice. For example, the message “Yes” may be more probable than the message “Autobus”. We assume a uniform distribution of keys

The Key Problem of OTP

The Practical Limitation

OTP requires keys that are **Truly Random, Message-Length, and Used Only Once.**

The Showstopper:

Securely pre-distributing message-length keys for real-world applications is **impractical**.

One might think – if it is hard to exchange keys, can I just reuse keys?

Cryptanalysis Question: Key Reuse

Suppose Alice reuses the key K to encrypt **two single-word english messages of length 5**, M_1 and M_2, producing C_1 and C_2.

$$C_1 = M_1 \oplus K \quad \text{and} \quad C_2 = M_2 \oplus K$$

Your task: Show that Eve can "likely" recover the key and then the two messages M_1 and M_2.

Eve can look up an English dictionary of all the words of length 5.
Then can produce

OTP: Integrity is NOT Included

The Integrity Flaw

- OTP guarantees **Confidentiality** (nobody can read it).
- It **DOES NOT** guarantee **Integrity** (nobody can tamper with it).

Attack Example (Eve flips the first bit of M):

1. Eve flips the first bit of C to get C'.
2. Bob decrypts C': $M' = C' \oplus K = (M \oplus K \oplus \Delta) \oplus K = M \oplus \Delta$.

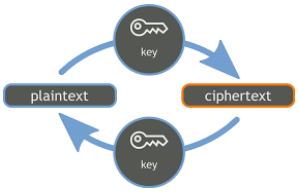
Result: Eve successfully changed the message (M' is M with one bit flipped) **without knowing K.**

Module 4: From OTP to Stream Ciphers

Two Main Types of Symmetric Schemes

Encryption of plaintext and decryption of ciphertext are done using

THE SAME KEY



- **Block Ciphers:** Operate on fixed-size blocks (e.g., 128 bits).
  - *Example:* **AES (Advanced Encryption Standard)**.
- **Stream Ciphers:** Operate one bit/byte at a time, like a pseudo-OTP.
  - *Example:* **ChaCha20**.

We will dive deeper into **Stream Ciphers**.

What is a symmetric cryptographic key?

Fixed-size input to symmetric cryptographic primitives.

The size of the key influences the level of security provided

Key properties

Known to both parties

Partners must agree on the key **before** starting using the primitive

It is reused

The key is pre-shared once* and then reused

* keys do have a “duration”

It must be secret

Revealing the key eliminates any protection provided by the primitive

Stream Ciphers: The Pseudo-OTP

A stream cipher attempt to emulate the OTP's security while solving the key-length problem.

1. **Short Key (K):** Alice and Bob share a small, manageable key.
2. **Key Stream Generator:** Uses K and an **Initialization Vector (IV)** to produce an arbitrary long, *pseudo-random* bit stream (S).
3. **Encryption:** $C = M \oplus S$.

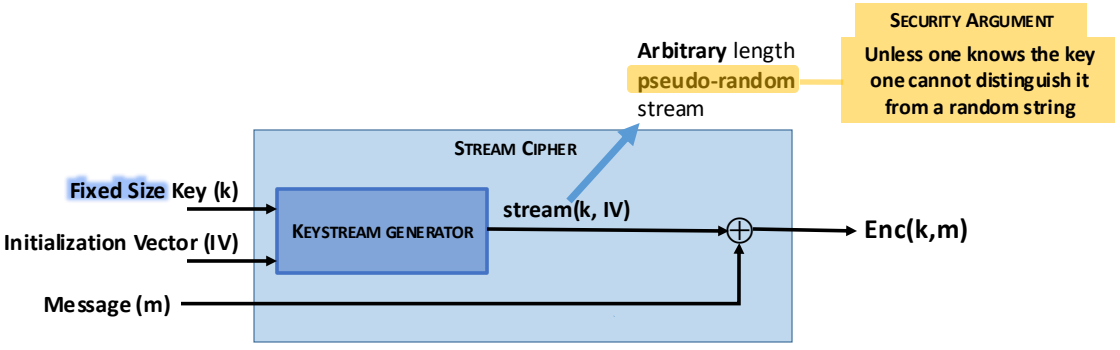
Question: Why does the Key Stream Generator takes in also an IV, and not just the key?

The KSG needs an IV to make sure that the encryption of two different messages will generate two different randomness, without that, one could simply XOR two ciphertexts together to recover the prefix of S, and decode future messages.

KSG: Finite Secret, Infinite "Randomness"?

- We are generating **potentially infinitely long randomness** (the stream S) from a **finite secret** (K).
- **Security:** Depends entirely on S being computationally indistinguishable from a truly random OTP key.

Stream Ciphers



A stream cipher receives two inputs:

- A small (much smaller than the message!) key that must be kept secret
- An initialization vector that does not need to be secret (see next slide)

The stream cipher outputs a stream of bits that is pseudorandom (i.e., looks random for an adversary that does not have the key)

To **encrypt** the message, similar to the one time pad, we XOR the message bits with the output of the stream cipher.

To **decrypt**, the receiver repeats the operation: feeds the stream cipher with the shared key and the IV (notice it is sent with the encrypted message) to create the same stream. Then XORs this stream with the encrypted message to obtain the plaintext.

Strengths and Weaknesses

Speed	Algorithms are linear in time and constant in space.
Low Error Propagation	Errors in one bit do not affect subsequent symbols.
Low Diffusion	A change in one plaintext bit only affects one ciphertext bit.
Susceptibility to Modification	Low diffusion makes it easier to tamper with the message (like the OTP).

Cryptanalysis: The Problem with Periodicity

- Key Stream Generators which have finite state are eventually **periodic**.
- If the message is longer than the period, the stream *repeats*.
- **The Attack:** If the period is identified, and too small, the key stream is known for the entire message.

Interlude On Building KSG

You are not expected to fully understand the following next 3 slides, they are here to illustrate some design patterns for KSG, and challenges involved in building secure KSG, and hint at the math needed to study these objects.

50

The **Linear Feedback Shift Register** Building Block for KSG

- **Linear Recurrence Relation**, The next output bit is a linear function (XOR) of previous bits, apply a recurrence relation to extend the initial random sequence of bits, into an arbitrary length sequence of (maybe?) randomly looking bits:
- For example, start with an initial state of 4 bits, a_0,…,a_3, and
$$a_n = a_{n-3} \oplus a_{n-4}$$
- **Output Sequence (starting from 1, 0, 0, 0):** 1,0,0,0,1,0,0,1,1,…
- **Advantage:** Uses only **shift registers** (for state storage) and **XOR gates** (for feedback), so it is extremely fast and efficient to implement in hardware.

A 15mn video on LFSR: https://www.youtube.com/watch?v=Ks1pw1X22y4

Randomness of LFSR

- **Maximal Period:** If the characteristic polynomial (AICC1) of the recurrence relation of the LFSR is primitive (MATH-215, Algebra 3), the maximum possible number of states before repeating: For an L-bit register, the maximum period is $2^L - 1$ states.

- **Distribution Property:** As a consequence, the sequence generated exhibit good distribution properties. Specifically, every possible non-zero state appears exactly once in the cycle, so over one full cycle of values, the output sequence are highly balanced. For example, the sequence will contain exactly 2^{L-1} ones and $2^{L-1} - 1$ zeros.

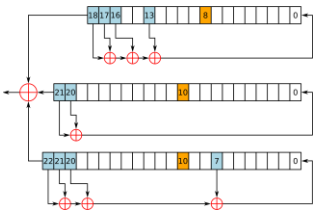
Using Math that you have seen in AICC1, AICC2, we could show some interesting properties of LFSRs.

The first 7mn video on LFSR: https://www.youtube.com/watch?v=Ks1pw1X22y4

LFSR - Math Danger

- Despite passing some statistical tests, the underlying operation of an LFSR is purely **linear**.
- **Consequence:** Techniques like the Berlekamp-Massey algorithm (beyond the scope of the class) can recover the entire LFSR state and the key stream by observing only a short segment of the output, breaking the cipher.
- **LFSRs should not be used directly as a Key Stream Generators**, though it has been traditionally used as a building block of Key Stream Generators.

The A5/1 Cipher

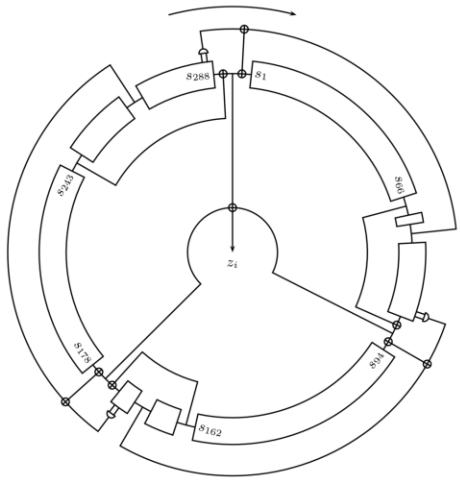


- **A5/1:** Used historically to secure GSM mobile phone communications.
- It was built by **combining multiple LFSRs** in a non-linear way to try and overcome the linearity flaw while still being cheap and having the nice statistical properties.
- **The Break:** Despite the increased complexity, mathematical weaknesses were found, allowing the cipher to be broken much faster than brute force.

A5/1 was used for GSM communication, a simple LFSR based stream cipher. It is practically broken.

Non-broken Stream Ciphers

- There are alternative, modern stream ciphers for which there are no known practical cryptanalysis.
- *Examples:* **Trivium** (based on NLFSRs, variation of LFSRs) and **Salsa20** (using different techniques).



There are alternative, modern Stream Cipher, Trivium and Salsa20 for example, for which there are no known practical cryptanalysis.
The image represent the Trivium, it is based on 3 Shift Registers, which are glued together with XOR, and AND gates, in a complex way.

Bit Security and Consequences

- **Cost of Breaking:** Exploiting a mathematical property or bug can break the scheme much faster than $2^{|K|}$ attempts, **reducing the *bit security***.
- **The Difficulty:** Designing mathematically secure and subtly resistant primitives is incredibly hard.
- **Conclusion:** Never design your own stream cipher, (block cipher, or hash function).
- **Best Practice:** Use **well-vetted, peer-reviewed, standard algorithms and their implementation** (like AES, ChaCha20, SHA-3).

Module 5: Public Key Cryptography

Recap: The Key Distribution Problem

Symmetric ciphers are fast, but require a **pre-shared secret key**.

Question: How can Alice and Bob agree on a secret key if all their communication is observed by Eve?

The Magic of Asymmetric Crypto

- **The Paradox:** Before the 1970s, it was considered impossible to communicate securely without a pre-shared secret.
- **The Revelation:** Public-Key Cryptography allows two parties who have never met to establish a secure channel.

Diffie-Hellman Setup

Alice, Bob (and Eve) know large public parameters: a prime number p and a generator g.

Alice's Actions

1. Chooses **private** secret a.
2. Computes **Public Value** $A = g^a \pmod{p}$.
3. Sends A to Bob.

Bob's Actions

1. Chooses **private** secret b.
2. Computes **Public Value** $B = g^b \pmod{p}$.
3. Sends B to Alice.

Note: Eve sees A, B, g, and p, but cannot find a or b due to the Discrete Log Problem.

While it has a low complexity to compute g to the power b modulo p (with fast modular exponentiation algorithm, for example), there are no known fast (polynomial) algorithm to recover a from A and p (or b from B and p).

The Shared Secret

Both Alice and Bob can compute a **shared secret** K:

$$K = B^a \pmod p = \left(g^b\right)^a \pmod p$$
$$= g^{\mathrm{ba}} \pmod p$$
$$= g^{ab} \pmod p$$
$$\left(g^b\right)^a \pmod p = A^b \pmod p$$

Where:

$K = B^a \pmod p$ is computable by Alice (she has the secret a)

$K = A^b \pmod p$ is computable by Bob (he has the secret b)

But Eve cannot compute K, because she has neither a nor b.

G^(b*a) is known by Alice, has it is public, and she knows a, so she can compute it, but it is equal to B^a

Trapdoor Functions

- This example of DH is one of the many examples of how to do public key cryptography to establish a shared secret (a shared key) to then to symmetric cryptography.
- They boil down to mathematical functions that are **easy to compute** in one direction.
- But **hard (high complexity) to do in the other direction**
- **Underlying Difficulty:** For DH, it is the **Discrete Logarithm Problem** .

Beyond DH

There are other ways to set a shared secret (or even to directly communicate), all involve interesting mathematics.

- **RSA:** Based on the Factoring problem.

- **Elliptic Curve Cryptography (ECC):** Can do Discrete Log on ECC, instead of on public-key crypto.

- Post-Quantum Cryptography: New primitives (like **Lattice Crypto**) to resist quantum computers.

Man-In-The-Middle (MITM) (1/2)

Vanilla DH is Vulnerable to Active Attack

- DH only guarantees key *agreement*, not *authenticity* (verifying *who* you are agreeing with).
- **Eve Intercepts:** Eve intercepts A from Alice and B from Bob.

MITM (2/2)

Eve Impersonates

1. Eve sends E_B to Alice. Alice computes K_{AE}.
2. Eve sends E_A to Bob. Bob computes K_{BE}.

- Eve shares a key K_{AE} with Alice and a separate key K_{BE} with Bob.
- She can read, modify, and relay all communication.

The Need for Trust (Authentication)

- DH solves **Key Distribution**, but creates the **Identity Verification Problem**.
- **Solution:** DH must be paired with a mechanism (like **Digital Signatures**) to verify identity. (We will discuss that next week)
- **Setup:** Parties still need to trust a third party (a Certificate Authority) for public key ownership.