Computer Security
Theory

Christian Rossi

Academic Year 2023-2024

Abstract

The course topics are:

- Introduction to information security.
- A short introduction to cryptography.
- Authentication.
- Authorization and access control.
- Software vulnerabilities.
- Secure networking architectures.
- Malicious software.

Contents

CHAPTER 1

Introduction

1.1 Basic security requirements

The fundamental security principles, known as the CIA paradigm for information security, outline three key requirements:

- *Confidentiality*: only authorized entities can access information.
- *Integrity*: information can only be modified by authorized entities in authorized ways.
- *Availability*: information must be accessible to all authorized parties within specified time limits.

It's worth noting that the availability requirement can sometimes conflict with the other two, as higher availability exposes the system for longer durations.

1.2 Definitions

Definition (*Vulnerability*)**.** A vulnerability is a flaw that can be exploited to violate one of the constraints of the CIA paradigm.

Definition (*Exploit*)**.** An exploit is a specific method of leveraging one or more vulnerabilities to achieve a particular objective that breaches the constraints.

Definition (*Asset*)**.** An asset is anything of value to an organization.

Definition (*Threat*)**.** A threat is a potential event that could lead to a violation of the CIA paradigm.

Definition (*Attack*)**.** An attack is a deliberate use of one or more exploits with the aim of compromising a system's CIA.

Definition (*Threat agent*)**.** A threat agent is any entity or factor capable of causing an attack.

Definition (*Hacker*)**.** A hacker is an individual with advanced knowledge of computers and networks, driven by a strong curiosity and desire to learn.

Definition (*Black hats*)**.** Malicious hackers are commonly referred to as black hats.

1.3 Ethical hacking

White hats, also known as security professionals or ethical hackers, are tasked with:

- *Identifying vulnerabilities.*
- *Developing exploits.*
- *Creating attack-detection methods.*
- *Designing countermeasures against attacks.*
- *Engineering security solutions.*

Since no system is invulnerable, it's crucial to assess its risk level. This involves evaluating the potential damage due to vulnerabilities and threats through the concept of risk:

Definition (*Risk*)**.** Risk is a statistical and economic evaluation of potential damage resulting from the presence of vulnerabilities and threats:

$$\text{Risk} = \text{Asset} \times \text{Vulnerabilities} \times \text{Threats}$$

Assets and vulnerabilities can be managed, but threats are independent variables.

To ensure system security, a balance must be struck between cost and reducing vulnerabilities and containing damage. The costs of securing a system can be categorized as direct and indirect. Direct costs include management, operational, and equipment expenses, while indirect costs, which often form the larger portion, stem from:

- *Reduced usability.*
- *Slower performance.*
- *Decreased privacy* (due to security controls).
- *Lower productivity* (as users may be slower).

It's important to note that simply spending more money on security may not always resolve the issue.

In real-world systems, setting boundaries is essential, meaning that a portion of the system must be assumed as secure. These secure parts consist of trusted elements determined by the system developer or maintainer. For example, the level of trust in a particular system can be determined at the software, compiler, or hardware level.

CHAPTER 2

Cryptography

2.1 Introduction

Definition (*Cryptography*)**.** Cryptography refers to the field of study concerned with developing techniques that enable secure communication and data storage in the presence of potential adversaries.

Cryptography offers several essential features, including:

- *Confidentiality*: ensures that data can only be accessed by authorized entities.
- *Integrity/freshness*: detects or prevents tampering or unauthorized replays of data.
- *Authenticity*: certifies the origin of data and verifies its authenticity.
- *Non-repudiation*: ensures that the creator of data cannot deny their responsibility for creating it.
- *Advanced features*: includes capabilities such as proofs of knowledge or computation.

2.1.1 History

Cryptography has a history as ancient as written communication itself, originating primarily for commercial and military purposes. Initially, cryptographic algorithms were devised and executed manually, using pen and paper.

The early approach to cryptography involved a contest of intellect between cryptographers, who devised methods to obscure messages, and cryptanalysts, who sought to break these ciphers.

A significant development occurred in 1553 when Bellaso pioneered the idea of separating the encryption method from the key.

In 1883, Kerchoff formulated six principles for designing robust ciphers:

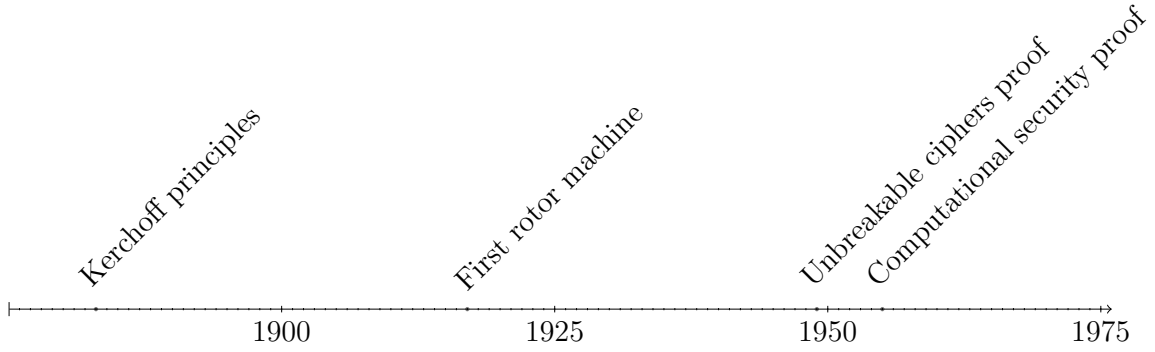
1. The cipher should be practically, if not mathematically, unbreakable.
2. It should be possible to disclose the cipher to the public, including enemies.

3. The key must be communicable without written notes and changeable at the discretion of correspondents.
4. It should be suitable for telegraphic communication.
5. The cipher should be portable and operable by a single person.
6. Considering the operational context, it should be user-friendly, imposing minimal mental burden and requiring a limited set of rules.

The landscape of cryptography underwent a significant transformation in 1917 with the introduction of mechanical computation, exemplified by Hebern's rotor machine, which became commercially available in the 1920s. This technology evolved into the German Enigma machine during World War II, whose encryption methods were eventually deciphered by cryptanalysts at Bletchley Park, contributing significantly to the Allied victory.

After World War II, in 1949 Shannon proved that a mathematically secure ciphers exists.

Following World War II, in 1949, Shannon demonstrated the existence of mathematically secure ciphers. Subsequently, in 1955, Nash proposed the concept of computationally secure ciphers, suggesting that if the interaction of key components in a cipher's determination of ciphertext is sufficiently complex, the effort required for an attacker to break the cipher would grow exponentially with the length of the key ($\mathcal{O}(2^\lambda)$), surpassing the computational capabilities of the key owner ($\mathcal{O}(\lambda^2)$) for sufficiently large key lengths (λ).



2.1.2 Definitions

Definition (*Plaintext space*)**.** A plaintext space P is the set of possible messages $ptx \in P$.

Definition (*Ciphertext space*)**.** A ciphertext space C is the set of possible ciphertext $ctx \in P$.

It's worth noting that the ciphertext space C may have a larger cardinality than the plaintext space P.

Definition (*Key space*)**.** A key space K is the set of possible keys.

The length of the key often correlates with the desired level of security.

Definition (*Encryption function*)**.** An encryption function $\mathbb{E}$ is a mapping that takes an element from the plaintext space P and a key from the key space K, and produces an element from the ciphertext space C:

$$\mathbb{E} : P \times K \to C$$

Definition (*Decryption function*)**.** A decryption function $\mathbb{D}$ is a mapping that takes an element from the ciphertext space C and a key from the key space K, and yields an element from the plaintext space P:

$$\mathbb{D} : C \times K \to P$$

2.2 Computational security

The objective of ensuring confidentiality is to prevent unauthorized individuals from comprehending the data. Various methods can compromise confidentiality:

- Passive interception by an attacker.
- Knowledge of a set of potential plaintexts by the attacker.
- Data manipulation by the attacker to observe the reactions of an entity capable of decryption.

Definition (*Perfect cipher*)**.** In a perfect cipher, for any plaintext ptx in the plaintext space P and any corresponding ciphertext ctx in the ciphertext space C, the probability of the plaintext being sent is equal to the conditional probability of that plaintext given the observed ciphertext:

$$\mathrm{P}(ptx\ sent = ptx) = \mathrm{P}(ptx\ sent = ptx | ctx\ sent = ctx)$$

In other words, observing a ciphertext $c \in C$ provides no information about the corresponding plaintext it represents.

Theorem 2.2.1 (Shannon 1949)**.** *Any symmetric cipher* $\langle P, K, C, \mathbb{E}, \mathbb{D} \rangle$ *with* $|P| = |K| = |C|$*, achieves perfect security if and only if every key is utilized with equal probability* $\frac{1}{|K|}$*, and each plaintext is uniquely mapped to a ciphertext by a unique key:*

$$\forall\, (ptx, ctx) \in P \times C, \exists! k \in K \text{ such that } \mathbb{E}\,(ptx, k) = ctx$$

Example:
Let's consider P, K, and C as sets of binary strings. The encryption function selects a uniformly random, fresh key k from K each time it's invoked and computes the ciphertext as $ctx = ptx \oplus k$.

Gilbert Vernam patented a telegraphic machine in 1919 that implemented $ctx = ptx \oplus k$ using the Baudot code. Joseph Mauborgne proposed utilizing a random tape containing the key k.

Combining Vernam's encryption machine with Mauborgne's approach results in a perfect cipher implementation.

It's crucial to understand that while a cipher may achieve perfect security, this doesn't necessarily mean it's practical or user-friendly. Managing key material and regularly changing keys can be exceptionally challenging.

In practice, perfect ciphers often face vulnerabilities due to issues such as key theft or reuse. Additionally, the generation of truly random keys has historically been problematic, leading to potential vulnerabilities and breaches.

In practical terms, ensuring the security of a cipher involves ensuring that a successful attack would also require solving a computationally difficult problem efficiently. The most commonly utilized computationally hard problems for ciphers include:

- Solving a generic nonlinear Boolean simultaneous equation set.
- Factoring large integers or finding discrete logarithms.

- Decoding a random code or finding the shortest lattice vector.

These problems cannot be solved faster than exponential time. However, with some hints, they can become easier to solve within polynomial time.

At this juncture, proving computational security involves the following steps:

1. Define the ideal attacker's behavior.
2. Assume a specific computational problem is difficult.
3. Prove that any non-ideal attacker would need to solve the difficult problem.

The attacker is typically represented as a program capable of accessing given libraries that implement the cipher in question. The security property is defined as the ability to respond to a specific query. The attacker succeeds if it breaches the security property more frequently than would be possible through random guessing.

2.3 Pseudorandom number generators

To expand the key for use in a Vernam cipher with a finite-length key, we require a pseudorandom number generator (PRNG). We assume that the attacker's computational capability is limited to poly(λ) computations.

Definition (*Cryptographically safe pseudorandom number generators*)**.** A cryptographically secure pseudorandom number generator is a deterministic function:

$$\mathrm{PRNG} : \{0,1\}^{\lambda} \to \{0,1\}^{\lambda+I}$$

where I is an expansion factor, such that the output of the PRNG cannot be distinguished from a uniformly random sample $\{0,1\}^{\lambda+I}$ with computational complexity $\mathcal{O}(\mathrm{poly}(\lambda))$.

In practice, cryptographic pseudo-random number generators (CSPRNGs) are considered as candidates because there is no conclusive evidence supporting the existence of a definitive pseudo-random number generator (PRNG) function. Demonstrating the existence of a CSPRNG would imply $\mathcal{P} \neq \mathcal{NP}$.

Developing a CSPRNG from scratch is feasible but not the usual approach due to inefficiency. Typically, they are constructed using another fundamental element called Pseudorandom Permutations (PRPs), which are derived from PseudoRandom Functions (PRFs).

To randomly select a function, we start by considering the set:

$$F = \left\{ f : \{0,1\}^{in} \to \{0,1\}^{out}, in, out \in \mathbb{N} \right\}$$

A uniformly randomly sampled $f \xleftarrow{\$} F$ can be represented by a table with 2^{in} entries, each entry being *out* bits wide:

$$|F| = \left(2^{out}\right)^{2^{in}}$$

Example:
For instance, if $in = 2$ and $out = 2$, the function set $F = \{f : \{0,1\}^2 \to \{0,1\}^1\}$ consists of the 16 Boolean functions with two inputs. Each function is represented by a 4-entry truth table. The total number of functions is 16, corresponding to the $2^4 = 16 = (2^1)^{2^2}$ tables.

2.3.1 Pseudorandom function

Definition (*Pseudorandom function*)**.** A pseudorandom function (PRF) is denoted as:

$$prf_{seed} : \{0,1\}^{in} \rightarrow \{0,1\}^{out}$$

Where it takes an input and a λ-bit seed.

Consequently, prf_{seed} is entirely determined by the seed value. It cannot be distinguished from a random function:

$$f \in \left\{ f : \{0,1\}^{in} \rightarrow \{0,1\}^{out} \right\}$$

within polynomial time in λ. In other words, given $a \in \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}\}$, it is computationally infeasible to determine which of the following is true:

- $a = prf_{seed}(\cdot)$ with seed $\xleftarrow{\$} \{0,1\}^{\lambda}$.
- $b \xleftarrow{\$} F$, where $F = \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}\}$.

2.3.2 Pseudorandom permutation

Definition (*Pseudorandom permutation*)**.** A pseudorandom permutation is a bijective pseudorandom function defined as:

$$prf_{seed} : \{0,1\}^{len} \rightarrow \{0,1\}^{len}$$

It is characterized solely by its seed value and cannot be distinguished from a random function within poly(λ). This permutation represents a rearrangement of all possible strings of length len. In practical terms:

- It operates on a block of bits and yields another block of equal size.
- The output appears unrelated to the input.
- Its behavior is entirely determined by the seed, akin to a key in conventional cryptography.

However, there is no formally proven pseudorandom permutation because its existence would imply $\mathcal{P} \neq \mathcal{NP}$. Construction of such a pseudorandom permutation typically involves three steps:

1. Compute a small bijective Boolean function f with input and key.
2. Compute f again between the previous output and the key.
3. Repeat the second step until satisfaction.

PRP selection Modern Pseudorandom Permutations (PRPs) often emerge from public competitions, where cryptanalytic techniques help identify and eliminate biases in their outputs, ensuring robust designs.

These PRPs are commonly known as block ciphers. A block cipher is considered broken if it can be distinguished from a PRP with less than 2^{λ} operations, achieved by:

- Deriving the input corresponding to an output without knowledge of the key.
- Determining the key identifying the PRP or narrowing down plausible options.

- Detecting non-uniformities in their outputs.

The key length λ is chosen to be sufficiently large to render computing 2^λ guesses impractical. For different security levels:

- Legacy-level security typically employs λ around 80.
- For a security duration of five to ten years, λ is set to 128.
- Long-term security requires λ of 256.

2.3.3 Standard block ciphers

The Advanced Encryption Standard (AES) operates on a 128-bit block size and offers three key lengths: 128, 192, and 256 bits. Chosen as a result of a three-year public competition by NIST on February 2, 2000, AES emerged as the preferred standard out of 15 candidates and has since been standardized by ISO. Modern processor architectures such as ARMv8 and AMD64 include dedicated instructions to accelerate the computation of AES.

The predecessor to AES, known as the Data Encryption Standard (DES), was established by NIST in 1977. DES operated with a relatively short 56-bit key length, leading to security concerns. It was bolstered through triple encryption, effectively achieving an equivalent security level of $\lambda = 112$. Although still present in some legacy systems, DES has been officially deprecated.

2.4 Plaintext encryption

Encrypting plaintexts with a length less than or equal to the block size using a block cipher is effective. This method can be expanded by employing multiple blocks with a split-and-encrypt approach, also known as Electronic CodeBook (ECB) mode.

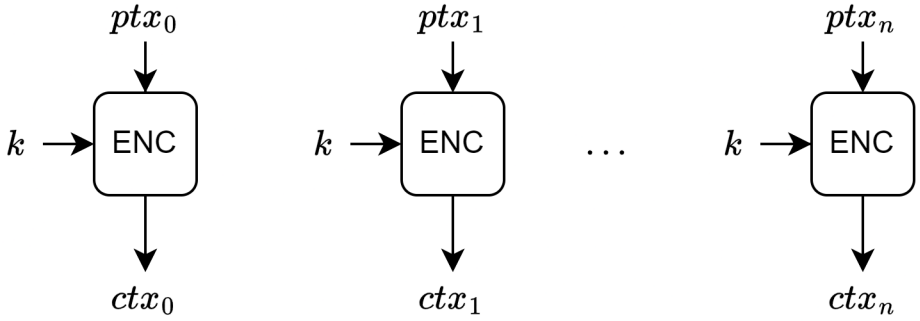


Figure 2.1: ECB encryption mode

However, this technique becomes problematic when there is redundancy within plaintext segments, as the resulting ciphertext may still reveal patterns. This vulnerability arises from the deterministic nature of ECB encryption, where identical plaintext blocks produce identical ciphertext blocks, making it susceptible to certain cryptographic attacks.

To address the issue of pattern visibility in ciphertexts caused by redundancy in plaintext segments, we can employ a counter to differentiate the strings submitted to each block during encryption. This counter, unique for each block, helps mitigate the predictability inherent in traditional encryption modes.

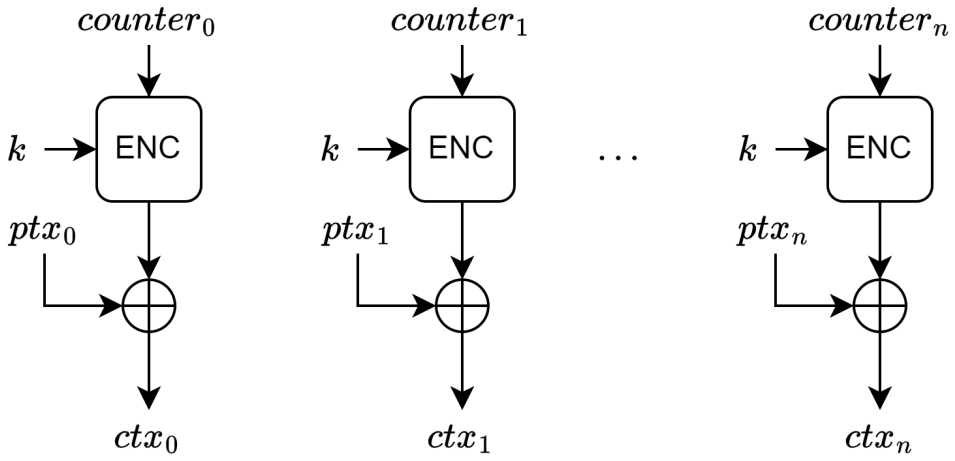


Figure 2.2: CTR encryption mode

This method ensures that even if plaintext blocks are repeated, the resulting ciphertext blocks are different due to the unique counter values assigned to each block.

2.4.1 Chosen plaintext attacks

Now, let's consider a scenario where the attacker has access to both the ciphertext and a portion of the plaintexts. In this type of attack, the attacker is familiar with a series of plaintexts that undergo encryption, and their objective is to determine the specific plaintext being encrypted.

In an ideal situation, the attacker should not be able to distinguish between two plaintexts of equal length when provided with their encrypted versions. Such scenarios frequently occur in contexts like managing data packets within network protocols and discerning between encrypted commands sent to a remote host.

The Counter (CTR) mode of operation is vulnerable to Chosen-Plaintext Attacks (CPA) due to its deterministic encryption process. To enhance security and achieve decryptable non-deterministic encryption, we can implement the following steps:

1. *Rekeying*: change the encryption key for each block using a mechanism like a ratchet, ensuring that each block's encryption is independent and unpredictable.

2. *Randomize the encryption*: introduce (removable) randomness into the encryption process by altering the mode of employing PseudoRandom Permutations (PRPs). This randomization enhances the unpredictability of the ciphertext, making it more resistant to cryptanalysis.

3. *Nonce usage*: utilize numbers used once (NONCEs) to introduce additional variability into the encryption process. In the case of CTR mode, a NONCE is chosen as the starting point for the counter. This NONCE can be public, adding an extra layer of unpredictability to the encryption.

By implementing these measures, we can significantly enhance the security of the encryption process and mitigate vulnerabilities associated with deterministic encryption modes like CTR.

Symmetric ratcheting The term ratcheting is derived from the mechanical device called a ratchet, which allows movement in one direction while preventing backward movement. Similarly, in symmetric ratcheting, the encryption keys are ratcheted forward in a manner that prevents an attacker from decrypting past messages even if they compromise the current key.

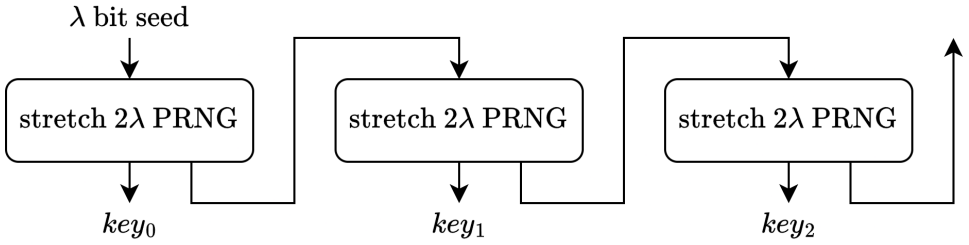


Figure 2.3: Ratcheting

Symmetric ratcheting ensures that even if an attacker manages to compromise the current encryption key, they cannot decrypt past messages or predict future messages due to the frequent key updates. This technique effectively limits the impact of key compromise and strengthens the security of encrypted communication over time.

Chosen plaintext attacks secure encryption Secure encryption schemes are designed to withstand CPA by ensuring that an attacker cannot gain any useful information about the encryption key or plaintexts, even if they have access to ciphertexts for chosen plaintexts. This is accomplished by utilizing the NONCE in conjunction with the Counter (CTR) mode of operation.

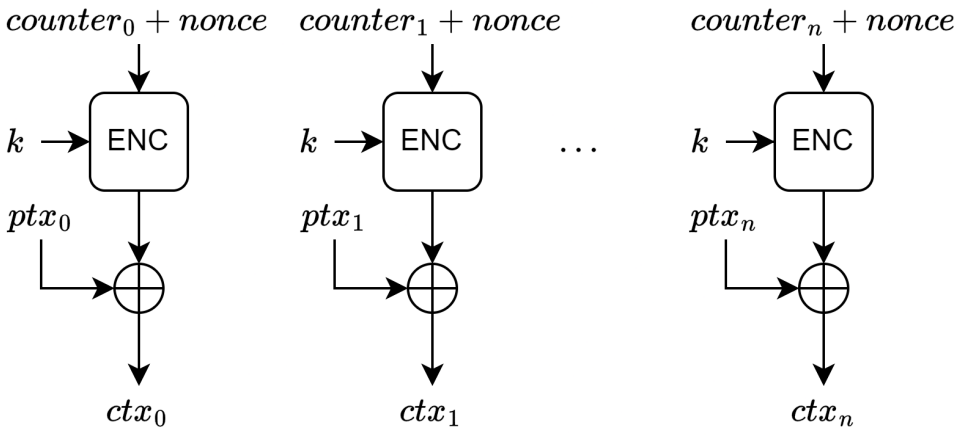


Figure 2.4: Secure ctr

2.5 Data integrity

Malleability refers to the ability to make alterations to the ciphertext, without knowledge of the encryption key, resulting in predictable modifications to the plaintext. This characteristic can be exploited in various ways to launch decryption attacks and manipulate encrypted data.

However, malleability can also be leveraged as a desirable feature, as seen in homomorphic encryption schemes.

To mitigate malleability, it is crucial to design encryption schemes that are inherently non-malleable and incorporate mechanisms to ensure data integrity against attackers. While current encryption schemes primarily provide confidentiality, they do not detect changes in the ciphertext effectively.

To address this limitation, a small piece of information known as a tag can be added to the encrypted message, allowing for integrity testing of the encrypted data itself. Simply adding the tag to the plaintext before encryption is not sufficient, as Message Authentication Codes (MACs) are required for proper data authentication.

2.5.1 Message authentication codes

A message authentication code consists of a pair of functions:

- `compute_tag(string, key)`: generates the tag for the input string.
- `verify_tag(string, tag, key)`: verifies the authenticity of the tag for the input string.

In an ideal attacker model, the attacker may possess knowledge of numerous message-tag pairs but should be unable to forge a valid tag for a message they do not already know. Additionally, tag splicing from valid messages should also be prevented.

CBC-MAC Cipher block chaining message authentication code (CBC-MAC) is a method for generating a fixed-size authentication tag from variable-length messages using a block cipher in CBC mode. Here's how CBC-MAC works:

1. *Initialization*: CBC-MAC operates on fixed-size blocks of data, so if the message is not a multiple of the block size, padding is applied to make it fit. The MAC is initialized with a zero or an initial value.
2. *Block Encryption*: the message is divided into blocks of equal size. Each block is encrypted using the block cipher in CBC mode. The ciphertext of each block is then XORed with the next plaintext block before encrypting the next block.
3. *Finalization*: once all blocks are encrypted, the last ciphertext block becomes the MAC.

CBC-MAC possesses several noteworthy characteristics. It is computationally efficient, requiring only a single pass through the message. The MAC generates a fixed-length authentication tag determined by the block size of the underlying block cipher. Additionally, CBC-MAC offers collision resistance, making it extremely difficult to find two different messages that produce the same MAC.

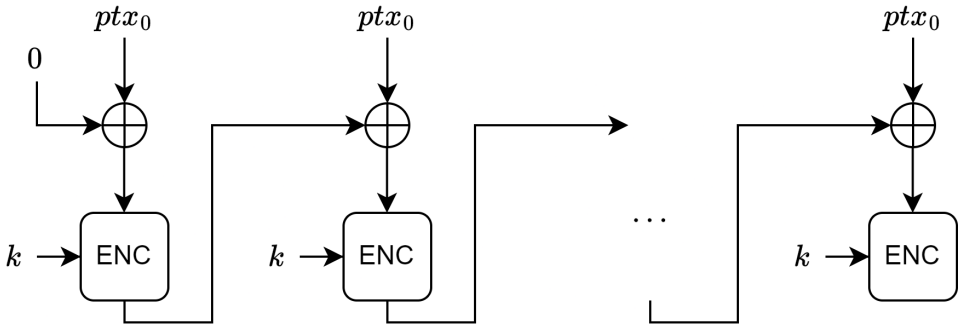


Figure 2.5: CTR encryption mode

CBC-MAC is widely used in practice for message authentication in various cryptographic protocols and applications, including network protocols, file authentication, and secure messaging systems. However, it is important to use CBC-MAC correctly and securely to avoid potential vulnerabilities.

MAC usages HTTP cookies serve as a form of "note to self" for HTTP servers, providing a means to store information locally within a user's browser. However, it is crucial that this information remains unaltered between server reads. To address this concern, the server employs a process where it computes a tag for the cookie using a cryptographic function, denoted as `compute_tag(cookie, k)`. This tag is then stored alongside the corresponding cookie as a pair (cookie, tag), ensuring the integrity and authenticity of the cookie's contents.

2.5.2 Cryptographic hashes

Ensuring the integrity of a file typically involves either comparing it bit by bit with an intact copy or reading the entire file to compute a message authentication code. However, it would be highly advantageous to verify the integrity of a file using only short, fixed-length strings, regardless of the file's size, thereby simplifying the process and reducing computational overhead. Unfortunately, a significant obstacle arises due to the inherent lower bound on the number of bits required to accurately encode a given content without any loss of information. This limitation presents a challenge when attempting to devise a method for efficiently testing the integrity of files.