**Summary Book : Serious Cryptography**

A Practical Introduction to Modern Encryption

by Jean-Philippe Aumasson

**Chapiter 1 : Encryption**

Encryption ensures data confidentiality using ciphers and keys. Symmetric encryption employs the same key for both encryption and decryption, contrasting with asymmetric encryption. Encryption progresses from weaker classical ciphers to stronger, enduring forms.

1. **Classical Ciphers :**

* **The Caesar Cipher :**

**Key = 3-letter shift**

|  |  |
| --- | --- |
| **Encrypt Message “Derradji”** | **Decrypt Message “Derradji”** |
| **D + 3 = G**  **E + 3 = H**  **R + 3 = U**  **R + 3 = U**  **A + 3 = D**  **D + 3 = G**  **J + 3 = M**  **I + 3 = L** | **G - 3 = D**  **H - 3 = E**  **U - 3 = R**  **U - 3 = R**  **D - 3 = A**  **D - 3 = A**  **M - 3 = J**  **L - 3 = I** |

* **The Vigenère Cipher :**

**Key = “BATNA”**

|  |  |
| --- | --- |
| **Encrypt Message “Derradji”** | **Decrypt Message “Derradji”** |
| **D + B = F**  **E + A = E**  **R + T = X**  **R + N = Z**  **A + A = B**  **D + B = F**  **J + A = K**  **I + T = Q** | **F - B = D**  **E - A = E**  **X - T = R**  **Z - N = R**  **B - A = A**  **F - B = D**  **K - A = J**  **Q - T = I** |

1. **How Ciphers Work**

* **Permutation**: A function that transforms an item (like a letter or bits) with a unique inverse (Caesar cipher’s 3-letter shift).
* **Mode of Operation**: An algorithm using permutations to process messages of any size.
* **Cipher Security**: Classical ciphers are insecure due to predictable patterns, unlike modern, high-speed computer-based ciphers.

1. **One-Time-Pad**

**Key = “CUB”**

|  |  |
| --- | --- |
| **Encrypt Message “Derradji”** | **Decrypt Message “Derradji”** |
| **‘d’ XOR ‘C’ = ‘S’**  **‘e’ XOR ‘U’ = ‘O’**  **‘r’ XOR ‘B’ = ‘T’**  **‘r’ XOR ‘C’ = ‘X’**  **‘a’ XOR ‘U’ = ‘S’**  **‘d’ XOR ‘B’ = ‘V’**  **‘j’ XOR ‘C’ = ‘X’**  **‘i’ XOR ‘U’ = ‘T’** | **‘S’ XOR ‘C’ = ‘d’**  **‘O’ XOR ‘U’ = ‘e’**  **‘T’ XOR ‘B’ = ‘r’**  **‘X’ XOR ‘C’ = ‘r’**  **‘S’ XOR ‘U’ = ‘a’**  **‘V’ XOR ‘B’ = ‘d’**  **‘T’ XOR ‘C’ = ‘j’**  **‘U’ XOR ‘T’ = ‘i’** |

1. **Encryption Security**

**Attacks Models**

* **Black-Box Model**: Attacker lacks knowledge of the system’s internals; treats it as a “black box.”
* **Gray-Box Model**: Attacker has partial knowledge (high-level details) of the system.
* **Kerckhoffs’ Principle**: Security relies on the secrecy of the key, not the algorithm.

**Security Goals**

* **Indistinguishability (IND)**: It ensures that an attacker cannot distinguish between encrypted messages based on their ciphertexts.
* **Non-malleability (NM)**: It prevents an adversary from altering ciphertexts to produce valid but unintended plaintexts.

**Security Notions**

* **Randomized Encryption (IND-CPA)**:
  + Randomization during encryption prevents vulnerabilities.
  + IND-CPA ensures that ciphertexts reveal no information about the underlying plaintext.
* **Achieving Semantically Secure Encryption**:
  + By using randomized encryption schemes like Fernet, we achieve semantic security.
  + The ciphertext should be indistinguishable from random noise, even when the adversary interacts with the encryption oracle.
* **Comparing Security Notions**:
  + Different security notions address various attack scenarios.
  + Understanding their trade-offs helps in selecting appropriate cryptographic primitives.

1. **Asymmetric**

* **I**n asymmetric encryption, two keys are used: a public key for encryption and a private key for decryption. The public key is openly shared, while the private key remains confidential. Public-key cryptography allows easy computation in one direction but not in the other, ensuring security. Asymmetric encryption faces similar security concerns as symmetric encryption but operates under a chosen-plaintext attacker model. Symmetric and asymmetric encryption are often combined to create secure communication systems and serve as the foundation for more advanced cryptographic schemes.

1. **When Ciphers Do More Than Encryption**

* **Authenticated Encryption (AE):** Provides both ciphertext and authentication tag, ensuring message integrity and source authenticity.
* **Authenticated Encryption with Associated Data (AEAD):** Incorporates unencrypted data to generate authentication tags, useful for preserving data integrity in network protocols.
* **Format-Preserving Encryption (FPE):** Encrypts data while preserving its original format, enabling compatibility with systems that require specific data structures.
* **Fully Homomorphic Encryption (FHE):** Allows computation on encrypted data without decryption, promising enhanced privacy but hindered by computational inefficiency.
* **Searchable Encryption:** Enables search over encrypted databases without revealing search terms, potentially bolstering privacy in cloud-based services.
* **Tweakable Encryption (TE):** Incorporates a tweak parameter to customize encryption, particularly useful for disk encryption to ensure data security and prevent cloning.

1. **How Things Can Go Wrong**

* **Weak Cipher Example:** A5/1 cipher in 2G mobile communication was weaker than expected, enabling call interception, highlighting the necessity for robust encryption standards.
* **Wrong Model Example:** Padding oracle attacks exploit decryption errors to decrypt secure ciphertexts, emphasizing the importance of considering all potential attack vectors.
* **Anticipating Attacks:** Engineers must anticipate diverse attack methods and model them accurately to prevent security breaches, underscoring the need for comprehensive cryptographic security measures.
* **Cryptographer Oversight:** Cryptographers may overlook certain attacks like padding oracle attacks, emphasizing the importance of thorough analysis and consideration during cryptographic system design and deployment.

**2.2 Chapiter 2 : Randomness**

Randomness is fundamental in cryptography, from secret key generation to encryption schemes and thwarting attacks. It ensures unpredictability, crucial for maintaining security, as predictable operations would render cryptography ineffective.

**2.1 Random or non-Random**

In cryptography, the concept of randomness is critical, yet often misunderstood. While random-looking sequences may appear secure, they can be misleading, masking potential vulnerabilities. Mistaking non-randomness for randomness and vice versa poses significant security risks in cryptographic systems. In crypto, non-randomness correlates with insecurity, emphasizing the importance of distinguishing between random-looking and truly random elements for robust cryptographic design and implementation.

**2.2 Randomness as a Probability Distribution**

randomized processes are defined by probability distributions, outlining the likelihood of various outcomes. Probability distributions assign probabilities to each potential outcome, ranging from 0 (impossible) to 1 (certain). A distribution encompasses all possible events, ensuring that the sum of probabilities equals 1. A uniform distribution implies equal likelihood among outcomes, while non-uniform distributions indicate varying probabilities, as seen in biased coin tosses. Understanding and analyzing probability distributions are fundamental to assessing randomness and ensuring cryptographic security.

**2.3 Entropy: A Measure of Uncertainty**

* **Definition**: Entropy quantifies the uncertainty or disorder in a system.
* **Analogy**: Think of entropy as the level of surprise in the outcome of a random process. Higher entropy implies greater uncertainty.
* **Computing Entropy**: Given a probability distribution with probabilities (p\_1, p\_2, \ldots, p\_N), the entropy is the negative sum of these probabilities multiplied by their logarithms (using base 2 logarithm).
* **Example**: For a uniform distribution, where all outcomes are equally likely, entropy is maximized. A 128-bit uniformly distributed key has an entropy of 128 bits.
* **Coin Toss Example**: The entropy of a fair coin toss is 1 bit. If one side (e.g., heads) is more likely (probability 1/4) than the other (e.g., tails with probability 3/4), the entropy decreases.

**2.4 Random Number Generators (RNGs) and Pseudorandom Number Generators (PRNGs)**

Cryptosystems rely on randomness for security, utilizing random number generators (RNGs) and pseudorandom number generators (PRNGs). RNGs harness analog environmental phenomena to produce unpredictable bits, while PRNGs convert this output into high-quality randomness. Despite their efficiency, PRNGs demand careful implementation to avoid vulnerabilities. Quantum random number generators offer promising solutions for enhancing cryptographic security. The symbiotic relationship between RNGs and PRNGs ensures cryptosystems withstand sophisticated attacks.



**How PRNGS Work**

Pseudorandom Number Generators (PRNGs) derive randomness from Random Number Generators (RNGs) to update an entropy pool, akin to an RNG's environmental input. The PRNG employs a Deterministic Random Bit Generator (DRBG) algorithm to expand bits from the entropy pool into longer sequences, ensuring uniqueness with each input. Its three main operations include init(), refresh(R), and next(N), initializing the PRNG, updating the entropy pool with data from an RNG, and returning N pseudorandom bits while updating the entropy pool. The refresh operation, often termed reseeding, involves a seed, which, in the absence of an RNG, may be hardcoded unique values, ensuring cryptographic strength.

**Security Concerns**

PRNGs must ensure backtracking and prediction resistance for robust security. Backtracking resistance requires irreversible transformations during state updates via refresh and next operations. Prediction resistance demands regular refresh calls with unknown R values to prevent future entropy pool prediction. Known R values don't compromise security, as reconstructing the pool necessitates knowledge of call order. These measures bolster cryptographic resilience against potential attacks.

**The PRNG Fortuna**

Fortuna, a PRNG since 2003, builds on Yarrow, used in macOS and iOS. It features 32 entropy pools and an internal state with a key and counter. Operations like init(), refresh(R), and next(N) drive its functionality, ensuring periodic entropy updates and pseudorandom bit generation. However, accurate implementation is challenging, lacking exhaustive test suites, and poses security risks like insufficient bit production and seed file exposure. Careful management is vital to maintain Fortuna's cryptographic integrity.

**Cryptographic vs. Non-Cryptographic PRNGs**

* + **Cryptographic PRNGs**: These are designed for cryptographic purposes, such as generating secure keys or initialization vectors. They emphasize unpredictability, resistance to attacks, and long-term security.
  + **Non-Cryptographic PRNGs**: These serve general-purpose needs (e.g., simulations, games). They prioritize speed and efficiency but lack the security guarantees of cryptographic PRNGs.

**The Uselessness of Statistical Tests**:

* + **Issue**: Statistical tests alone cannot prove the quality of a PRNG.
  + **Why?**: Even a flawed PRNG can pass statistical tests, yet fail in real-world scenarios.
  + **Caution**: Relying solely on statistical tests can lead to vulnerabilities.

**Linearity Insecurity**:

* + **Problem**: Linear congruential generators (LCGs) exhibit linearity.
  + **Consequence**: Predictable patterns emerge, compromising security.
  + **Recommendation**: Avoid LCGs for cryptographic purposes.

**A Popular Non-Crypto PRNG: Mersenne Twister**:

* + **Overview**: Mersenne Twister is widely used due to its long period (2^19937 - 1) and fast generation.
  + **Limitation**: It’s not suitable for cryptographic applications.
  + **Caution**: Avoid using Mersenne Twister for sensitive tasks.

**2.4 Real-World PRNGs**

**2.6 How Things Can Go Wrong**

**Chapiter 3 : Cryptographic Security**

Cryptography’s security definitions differ from general software security standards. Unlike software security, cryptographic security is quantifiable. While software applications are typically labeled as secure or insecure, cryptographic security offers measurable standards. The distinction lies in the precise metrics and criteria used to evaluate cryptographic systems, contrasting with the more subjective assessments in general computer security.

**3.1 : Defining the Impossible**

* + **Concepts**: Two notions define “impossible” in cryptography:
    - **Informational Security**: Theoretical impossibility—whether a cipher can be broken at all, even with unlimited resources.
    - **Computational Security**: Practical impossibility—whether a cipher resists attacks within reasonable time and resources.
  + **Relevance**: While informational security is theoretical, computational security quantifies real-world cipher strength.
    1. **Security in Theory: Informational Security**:
  + **Definition**: A cipher is informationally secure if, given unlimited computation time and memory, it remains unbreakable.
  + **Example**: Even if breaking the cipher would take trillions of years, it’s still informationally insecure.
  + **Theoretical Role**: Important in theory but lacks practical quantification.
    1. **Security in Practice: Computational Security**:

**Definition**: A cipher is computationally secure if it cannot be broken within a reasonable timeframe and available resources.

**Quantification**: Measures real-world strength considering memory, hardware, budget, and energy.

**Practical Relevance**: Crucial for assessing cipher security in practice.

* 1. **Quantifying Security**
* Understanding attack efficiency and cipher resilience.

**Measuring Security in Bits**: Defining cipher strength based on operation complexity.

**Full Attack Cost:** Factors influencing the practicality and expense of attacks.

**Parallelism:** Considering computational efficiency through parallel processing.

**Memory:** Assessing the impact of memory usage on attack feasibility.

**Precomputation:** Exploring one-time calculations and their impact on attacks.

**Number of Targets:** Evaluating the scope and risk associated with multiple targets.

**Choosing and Evaluating Security Levels:** Selecting appropriate security levels for cryptographic algorithms.

* 1. **Achieving Security**

Ensuring cryptographic schemes adhere to chosen security levels.

**Provable Security**: Guaranteeing cipher resilience through complexity theory.

**Proof Relative to a Mathematical Problem:** Establishing the hardness of breaking a scheme compared to solving mathematical challenges.

**Proof Relative to Another Crypto Problem:** Comparing schemes to demonstrate security through cryptographic relationships.

**Caveats:** Acknowledging limitations of security proofs and practical implementation concerns.

**Heuristic Security:** Understanding security without formal proof, common in symmetric ciphers like AES.

* 1. **Generating Keys**
  2. **How Things Can Go Wrong**

**Incorrect Security Proof**:

**Example**: Optimal Asymmetric Encryption Padding (OAEP) used RSA for secure encryption.

**Issue**: An incorrect proof of OAEP’s security against chosen-ciphertext attackers went unnoticed for seven years.

**Result**: OAEP is only almost secure against such attacks.

**Trust**: We now rely on a new proof, hoping it’s flawless.

**Short Keys for Legacy Support**:

**Discovery**: Some HTTPS sites and SSH servers used shorter public-key cryptography keys (512 bits instead of 2048 bits).

**Security Level**: Keys of 512 bits offer a security level of approximately 60 bits.

**Vulnerability**: These keys could be broken within about two weeks using computational resources.

**Impact**: Many websites, including the FBI’s, were affected.

**Resolution**: Patches were applied to fix the issue in software like OpenSSL.

**Chapiter 4 : Block Cipher**

A **block cipher** is a cryptographic algorithm that operates on fixed-size blocks of data. It consists of two essential components:

1. **Encryption Algorithm (E)**:
   * Takes a secret key, denoted as **K**, and a plaintext block, **P**.
   * Produces a corresponding ciphertext block, **C**.
   * Mathematically, this operation is expressed as: **C = E(K, P)**.
2. **Decryption Algorithm (D)**:
   * Acts as the inverse of the encryption algorithm.
   * Given the same key **K** and a ciphertext block **C**, it retrieves the original plaintext block **P**.
   * The decryption operation is written as: **P = D(K, C)**.

**Security Goals**

The definition of a secure block cipher aligns with concepts like randomness and indistinguishability. In essence, we consider a block cipher secure when it exhibits a level of “random-lookingness.”

**Block Size**

Two critical parameters characterize a block cipher:

1. **Block Size**:
   * Refers to the fixed size of data blocks processed by the cipher.
   * Most block ciphers use either 64-bit or 128-bit blocks.
   * For example:
     + DES (Data Encryption Standard) employs 64-bit blocks (2^6 bits).
     + AES (Advanced Encryption Standard) uses 128-bit blocks (2^7 bits).
   * Powers of two block sizes simplify data handling in computing systems.
2. **Key Size**:
   * Determines the length of the secret key.
   * Security depends on both the block size and key size.

**Codebook Attacks**

While choosing an appropriate block size, we must strike a balance. Blocks should not be too large or too small. Smaller blocks can lead to vulnerabilities like **codebook attacks**:

1. **Codebook Attack**:
   * Efficient against block ciphers with smaller block sizes (e.g., 16-bit blocks).
   * Steps:
     1. Collect 65,536 (2^16) ciphertexts corresponding to each 16-bit plaintext block.
     2. Create a lookup table (the codebook) mapping ciphertext blocks to their corresponding plaintext blocks.
     3. To decrypt an unknown ciphertext block, look up its corresponding plaintext block in the table.
   * With 16-bit blocks, the lookup table requires only 220 bits (128 kilobytes) of memory.
   * Larger block sizes (e.g., 32-bit or 64-bit) mitigate this vulnerability.

**4.2 The Advanced Encryption Standard (AES)**

**AES Security: A Closer Look**

AES is widely regarded as one of the most secure block ciphers available. Let’s explore why:

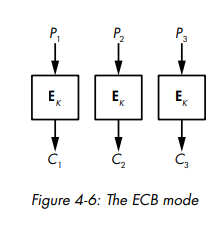
1. **Complex Pseudorandom Behavior**:
   * AES achieves security by ensuring that all output bits depend on all input bits in a complex, pseudorandom manner.
   * Designers meticulously selected components like **MixColumns** (for maximal diffusion) and **SubBytes** (for optimal non-linearity).
   * This composition shields AES against various cryptanalytic attacks.
2. **Lack of Absolute Proof**:
   * Despite its reputation, there’s no absolute proof that AES is immune to all possible attacks.
   * Unknown attack vectors exist, and proving security against every conceivable attack is challenging.
   * Confidence in AES security comes from real-world testing and analysis.
3. **Crowdsourcing Attacks**:
   * To gain confidence, the cryptographic community relies on crowdsourcing.
   * Talented researchers attempt to break AES, and the fact that they haven’t succeeded significantly reinforces its security.
4. **Theoretical Security and Practical Reality**:
   * Over 15 years of research, AES’s theoretical security has been rigorously explored.
   * In 2011, cryptanalysts discovered a way to recover an AES-128 key with fewer operations (a speed-up of 4).
   * However, this attack requires an enormous number of plaintext-ciphertext pairs (about 288 bits’ worth).
   * While interesting, it’s not a practical concern for most scenarios.
5. **Implementation and Deployment**:
   * When implementing and deploying cryptographic systems, focus on many factors.
   * AES’s core algorithm is robust, but the choice of modes of operation matters.
   * Incorrect modes or misuse can compromise even a strong cipher like AES.

**4.3 Modes of Operation**

**Electronic Codebook (ECB) Mode**

The **ECB mode** is the simplest among block cipher encryption modes, although it provides minimal security. Let’s delve into its characteristics:

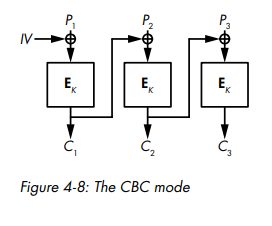
1. **Independence of Blocks**:
   * ECB processes plaintext blocks independently.
   * Given plaintext blocks **P1**, **P2**, …, **PN**, it computes corresponding ciphertext blocks:
     + **C1 = E(K, P1)**
     + **C2 = E(K, P2)**
     + And so on.
2. **Insecurity**:
   * Despite its simplicity, ECB is insecure due to its lack of diffusion and vulnerability to patterns.
   * Identical plaintext blocks result in identical ciphertext blocks.
   * This predictability makes it susceptible to attacks, especially when encrypting repetitive data.



**Cipher Block Chaining (CBC) Mode**

The **CBC mode** builds upon the simplicity of the Electronic Codebook (ECB) mode but introduces a crucial twist that significantly enhances security. Let’s explore its key features:

1. **Block Chaining**:
   * In CBC, each plaintext block **Pi** is processed differently compared to ECB.
   * Instead of directly encrypting **Pi** as **Ci = E(K, Pi)**, CBC computes **Ci = E(K, Pi ⊕ Ci-1)**.
   * Here, **Ci-1** represents the previous ciphertext block, effectively chaining the blocks together.
2. **Initialization Vector (IV)**:
   * When encrypting the first block (**P1**), there is no previous ciphertext block.
   * CBC uses a random initial value called the **Initialization Vector (IV)**.
   * The IV ensures that the encryption process starts securely.
3. **Dependency and Uniqueness**:
   * Each ciphertext block in CBC depends on all previous blocks.
   * Identical plaintext blocks will not result in identical ciphertext blocks.
   * This property enhances security by preventing patterns from emerging.
4. **Distinct Encryption with Different IVs**:
   * The random IV guarantees that two identical plaintexts will encrypt to distinct ciphertexts.
   * When calling the cipher twice with two distinct initial values, the resulting ciphertexts remain different.



**4.4 How Things Can Go Wrong**

**Meet-in-the-Middle Attacks:**

* Exploit reversible block cipher operations.
* Search for matching intermediate results during encryption and decryption.
* Relevant for double encryption scenarios.

**Padding Oracle Attacks:**

* Exploit padding correctness feedback.
* Modify ciphertexts to gain information about plaintext.
* Proper handling of padding errors mitigates this attack.

**Chapiter 5 : Stream Cipher**

* 1. **Block Ciphers**:
  + Block ciphers operate on fixed-size blocks of plaintext (usually 64 or 128 bits) and transform them into corresponding ciphertext blocks.
  + The encryption process involves mixing the plaintext bits with key bits using a specific algorithm (the block cipher).
  + Each block is processed independently, and the same key is used for all blocks.
  + Common block ciphers include **AES (Advanced Encryption Standard)** and **DES (Data Encryption Standard)**.
  1. **Stream Ciphers**:

Stream ciphers, unlike block ciphers, do not divide the plaintext into fixed-size blocks.

Instead, they generate a continuous stream of pseudorandom bits (keystream) from the encryption key.The keystream is then XORed with the plaintext to produce the ciphertext.Stream ciphers are often used for real-time communication, as they can encrypt data bit by bit.

An analogy is the **one-time pad**, where the keystream is truly random and used only once.

**5.3 Hardware-Oriented**

Stream Ciphers rely on dedicated electronic circuits like ASICs, PLDs, and FPGAs for implementation. Initially favored for their efficient bit-level operations, stream ciphers were preferred over block ciphers due to lower hardware costs. Feedback Shift Registers (FSRs) form the core mechanism of hardware stream ciphers, undergoing shift and update operations with a feedback function. The period of an FSR determines its repetition cycle, crucial for security. Despite advancements, stream ciphers' association with hardware persists, though the cost gap with block ciphers has diminished. Today, hardware-friendly block ciphers offer comparable efficiency, impacting the cipher selection landscape.

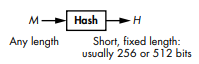
**5.4 Software-Oriented Stream Ciphers**

Software-Oriented Stream Ciphers cater to modern CPUs, operating on bytes or 32- to 64-bit words, leveraging efficiency in arithmetic operations. They find prominence in servers and personal computers, where powerful processors execute them as native software. Interest in software stream ciphers grows as powerful CPUs become more prevalent, diminishing the need for bit-oriented ciphers. Stream ciphers, such as SNOW3G and ZUC in 4G standards, operating on words, have eclipsed older bit-oriented ciphers like A5/1. Following incidents like the padding oracle attack, stream ciphers gain traction over block ciphers due to easier specification and implementation. Designs like RC4 and Salsa20, despite vulnerabilities, remain popular choices in various systems.

**RC4**, developed by Ron Rivest in 1987, gained widespread adoption due to its simplicity and speed. It operates by generating a pseudo-random keystream based on a secret key, which is then XORed with the plaintext to produce ciphertext. Despite its initial popularity, RC4 has been extensively studied and found to have several vulnerabilities, leading to its deprecation in many security applications.

**Salsa20**, designed by Daniel Bernstein in 2005, is a more modern stream cipher known for its high security and performance. It utilizes a 512-bit key and a 64-bit nonce to generate a stream of pseudo-random bytes, which are then XORed with the plaintext to produce ciphertext. Salsa20 is highly regarded for its resistance against cryptanalysis and its efficient implementation in software and hardware. It has been widely adopted in various cryptographic protocols and systems.

**Chapiter 6 : Hash Functions**

 **Hash functions** like MD5, SHA-1, SHA-256, SHA-3, and BLAKE2 serve as versatile tools in cryptography, employed in digital signatures, encryption, integrity verification, and more. For instance, when encrypting emails, browsing HTTPS websites, or connecting via IPSec or SSH, hash functions ensure security behind the scenes. They are essential for identifying files in cloud storage, Git repositories, and intrusion detection systems, playing a crucial role in various real-world applications.

**MD5 (Message Digest Algorithm 5)**:

* MD5 produces a 128-bit hash value.
* It’s no longer secure for cryptographic purposes but can be used for non-cryptographic tasks.

**SHA-1 (Secure Hash Algorithm 1)**:

* SHA-1 produces a 160-bit hash value.
* It’s also deprecated due to vulnerabilities.

**SHA-256 (Secure Hash Algorithm 256)**:

* SHA-256 produces a 256-bit hash value.
* It’s widely used for secure hashing.

**SHA-3 (Secure Hash Algorithm 3)**:

* SHA-3 comes in different variants (e.g., SHA3-224, SHA3-256).

**BLAKE2**:

* BLAKE2 is a fast and secure hash function.

**Chapiter 7 : Keyed Hashing**

**Keyed hashing** involves using secret keys to hash messages, enabling authentication and integrity protection. It forms the basis of cryptographic algorithms like message authentication codes (**MACs**) and pseudorandom functions (**PRFs**). MACs authenticate messages, while PRFs generate hash-sized values. These algorithms, whether based on hash functions or block ciphers, ensure message integrity and confidentiality in cryptographic protocols.

**Message Authentication Codes (MACs)** verify message integrity and authenticity using a key-generated authentication tag. Secure communication protocols like IPSec, SSH, and TLS employ MACs to ensure message security. MACs prevent forgeries through secret key protection and defend against replay attacks by incorporating message numbers.

**Pseudorandom Functions (PRFs)** use secret keys to generate outputs indistinguishable from random values. They're utilized in cryptographic protocols for key derivation, identification, and authentication. Unlike MACs, PRFs demand outputs that are indistinguishable from randomness for security. HMAC-SHA-256, used in TLS, serves as both a secure MAC and PRF.

**CMAC (Cipher-based MAC)** constructs MACs using block ciphers like AES, distinct from simpler CBC-MACs. It addresses CBC-MAC vulnerabilities by processing the last block with a separate key, fixing forgery risks. CMAC computes two distinct keys, K1 and K2, from the main key K, enhancing security. Unlike CBC encryption, CMAC is deterministic, ensuring consistent tags for given messages without needing random initialization vectors (IVs).

**Chapiter 8 : Authenticated Encryption**

**Authenticated encryption** combines message confidentiality and authenticity. Unlike message authentication codes (**MACs**) which only create a tag for authenticity, authenticated encryption algorithms also encrypt the message. This dual functionality makes authenticated encryption a versatile tool for securing data. Throughout this chapter, various methods of combining ciphers with MACs for authenticated encryption will be explored. Notable authenticated ciphers include block cipher-based constructions like AES-GCM, and ciphers utilizing permutation algorithms.

**8.1 Authenticated Encryption Using MACs**:

* + **Purpose**: Ensures both integrity and authenticity of messages.
  + **MAC (Message Authentication Code)**: A fixed-length checksum computed from a secret key and the message.
  + **Usage**: Alice computes a MAC on the message and sends it along with the message to Bob. Bob verifies the MAC to detect tampering or spoofing.
  + **Example**: Protecting stored files on a USB flash drive using MACs.

**8.2 Authenticated Ciphers**:

* + **Definition**: Authenticated ciphers combine encryption and authentication in a single step.
  + **AES-GCM (Galois/Counter Mode)**: An example of an authenticated cipher that provides confidentiality and authenticity.
  + **Inputs**: AES-GCM takes an AES key, an initialization vector (IV), plaintext, and optional additional authenticated data (AAD).

**8.3 AES-GCM: The Authenticated Cipher Standard**

**AES-GCM (Galois/Counter Mode)**:

1. **Purpose**: AES-GCM is an **authenticated encryption mode** that combines confidentiality and data origin authentication.
2. **Efficiency**: It is **faster than GCM**, even when GCM has hardware support.
3. **Inputs**: AES-GCM takes an AES key, an initialization vector (IV), plaintext content, and optional additional authenticated data (AAD).

**8.4 OCB: An Authenticated Cipher Faster than GCM**

**OCB (Offset Codebook Mode)**:

1. **Purpose**: OCB is an **authenticated encryption mode** that provides both privacy and authenticity.
2. **Efficiency**: OCB is **faster than GCM** (Galois/Counter Mode), even when GCM has hardware support.
3. **Block-Cipher Calls**: OCB uses nearly optimal block-cipher calls for encryption and authentication.

**Chapiter 9 : Hard Problems**

1. **Factoring**:
   * **Description**: Factoring large numbers into their prime factors.
   * **Challenge**: Given a number (n = p \cdot q), where (p) and (q) are large prime numbers, finding (p) and (q) is computationally difficult.
   * **Importance**: Factoring forms the basis for asymmetric encryption (e.g., RSA), and its hardness ensures security.
2. **Decisional Diffie-Hellman (DDH)**:
   * **Description**: Given (g^a), (g^b), and (g^c), determine whether (c = a \cdot b) (mod (p)).
   * **Challenge**: Deciding whether the discrete logarithm problem is easy or hard.
   * **Usage**: DDH assumption underlies security in various cryptographic protocols.

**Chapiter 10 : RSA**

1. **Algorithm Description**:
   * RSA is an **asymmetric** encryption algorithm, meaning it uses **two different keys**: a **public key** for encryption and a **private key** for decryption.
   * The algorithm relies on the **practical difficulty of factoring large numbers** into their prime factors.
2. **Key Components**:
   * **Public Key**: Created by an RSA user and published. It consists of two large prime numbers and an auxiliary value.
   * **Private Key**: Kept secret by the user. It allows decryption of messages encrypted with the public key.
3. **Encryption and Decryption**:
   * **Encryption**: Anyone can encrypt messages using the recipient’s public key.
   * **Decryption**: Only the recipient with the private key can decrypt the messages.
4. **Security Basis**:
   * RSA’s security relies on the **difficulty of factoring the product of two large prime numbers** (the “factoring problem”).
   * Breaking RSA encryption is known as the **RSA problem**.
   * The security remains intact if a **large enough key** is used.

**RSA Key Generation and Security**

1. **Key Pair Generation**:
   * RSA uses a pair of keys: **Public Key** and **Private Key**.
   * The **Public Key** is shared openly, while the **Private Key** remains confidential.
   * Here’s how the key pair is generated:
     + Choose two large prime numbers, say **P** and **Q**.
     + Calculate the modulus **n** as the product of **P** and **Q**: (n = P \cdot Q).
     + Select a small positive integer **e** (usually 3 or 65537) such that it is coprime with (\phi(n)) (Euler’s totient function).
     + Compute (\phi(n) = (P-1)(Q-1)).
     + Find the private exponent **d** such that (d \cdot e \equiv 1 \mod \phi(n)).
     + The public key is ((n, e)), and the private key is ((n, d)).
2. **Encryption with RSA**:
   * To encrypt a message **M** using someone’s public key:
     + Compute the ciphertext **C** as (C \equiv M^e \mod n).
     + Send **C** to the recipient.
3. **Decryption with RSA**:
   * To decrypt the ciphertext **C** using the recipient’s private key:
     + Compute the plaintext **M** as (M \equiv C^d \mod n).
4. **Security Considerations**:
   * RSA’s security relies on the difficulty of factoring large integers.
   * The strength of encryption increases exponentially with the key size.
   * Common key sizes are 1024 or 2048 bits.
   * Experts recommend using 2048-bit keys or higher.
   * RSA is widely used for secure data transmission and digital signatures.

**Encrypting with RSA**

When encrypting data with RSA, the sender uses the recipient’s **public key**. The recipient can then decrypt the data using their **private key**.

**Signing with RSA**

RSA is also used for **digital signatures**. A sender signs a message using their private key, and the recipient verifies the signature using the sender’s public key. This ensures data integrity and authenticity.

**Chapiter 11 : Diffie Hellman**

**The Diffie–Hellman Key Exchange**

* **Diffie–Hellman key exchange** is a mathematical method that allows two parties to establish a **shared secret key** without directly transmitting it across a public channel.
* Named after its inventors, **Whitfield Diffie** and **Martin Hellman**, DH was one of the earliest practical examples of **public-key cryptography**.
* Traditionally, secure communication required exchanging keys through secure physical means (like trusted couriers). DH revolutionized this process by allowing parties with no prior knowledge of each other to jointly create a shared secret key.
* The shared secret key can then be used for subsequent communication using a **symmetric-key cipher**.

**How Diffie–Hellman Works**

1. **Key Generation**:
   * Both parties agree on large prime numbers (**P** and **Q**).
   * They calculate the modulus **n** as (n = P \cdot Q).
   * Each party selects a small positive integer **e** (usually 3 or 65537) coprime with (\phi(n)).
   * (\phi(n)) is Euler’s totient function, computed as (\phi(n) = (P-1)(Q-1)).
   * The private exponent **d** is found such that (d \cdot e \equiv 1 \mod \phi(n)).
   * The public key is ((n, e)), and the private key is ((n, d)).
2. **Key Exchange**:
   * Parties exchange their public keys.
   * Using the other party’s public key, they compute a shared secret key.
3. **Security Considerations**:
   * DH’s security relies on the difficulty of factoring large integers.
   * The strength increases exponentially with the key size.
   * DH is used to secure various Internet services.

**Key Agreement Protocols**

* Although DH itself is a non-authenticated key-agreement protocol, it forms the basis for various authenticated protocols.
* It provides **forward secrecy** in **Transport Layer Security (TLS)** ephemeral modes (EDH or DHE).

**How Things Can Go Wrong**

* **Weak Parameters**: Insecure DH parameters can compromise security.
* **Man-in-the-Middle Attacks**: Without authentication, an attacker could intercept and modify exchanged keys.
* **Implementation Flaws**: Incorrect implementations may weaken security.

**Chapiter 12 : Hard Elliptic Curve**

**What Is an Elliptic Curve?**

* An **elliptic curve** is a smooth, projective, algebraic curve of genus one. It has a specified point called **O**.
* Defined over a field **K**, it describes points in **K²**, which is the Cartesian product of **K** with itself.
* The curve equation typically takes the form: (y^2 = x^3 + ax + b), where (a) and (b) are coefficients in **K**.
* Elliptic curves are essential in number theory, cryptography, and even played a role in Andrew Wiles’s proof of **Fermat’s Last Theorem**.

**The ECDLP Problem**

* The **Elliptic Curve Discrete Logarithm Problem (ECDLP)** is a special case of the discrete logarithm problem.
* Given an elliptic curve **E** defined over a finite field **Fq**, and points **P** and **Q** in the group generated by **P**, find the integer (l) such that (Q = lP).
* ECDLP’s apparent intractability forms the basis for the security of **Elliptic Curve Cryptography (ECC)**.

**Diffie–Hellman Key Agreement over Elliptic Curves**

* **ECDH (Elliptic-curve Diffie–Hellman)** allows two parties with elliptic-curve public–private key pairs to establish a shared secret over an insecure channel.
* The shared secret can be directly used as a key or used to derive another key.
* ECDH is a variant of the Diffie–Hellman protocol using elliptic-curve cryptography.

**Choosing a Curve**

* When using ECDH, selecting the right elliptic curve is crucial.
* The curve parameters (such as prime or binary fields) must be agreed upon.
* The public keys can be either static (trusted via certificates) or ephemeral (ECDHE).
* Static keys lack forward secrecy, while ephemeral keys provide better security properties.

**How Things Can Go Wrong**

* Weak curve parameters or incorrect modeling of the curve can compromise security.
* Man-in-the-middle attacks are thwarted by using static public keys.
* Proper validation and secure key derivation are essential.

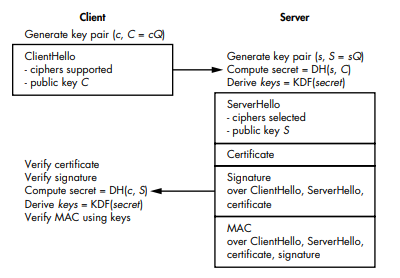
**Chapiter 13 : TLS**

**Target Applications and Requirements**

* **TLS** is a cryptographic protocol designed to provide **communications security** over computer networks.
* It is widely used in various applications, including:
  + **Web browsers**: For secure communication between web applications and servers (e.g., loading websites over HTTPS).
  + **Email**: Encrypting email communication.
  + **Messaging**: Ensuring privacy and integrity in messaging services.
  + **Voice over IP (VoIP)**: Securing voice communication.
* TLS aims to achieve **privacy**, **integrity**, and **authentication** through cryptography.

**The TLS Protocol Suite**

* **TLS** builds on the now-deprecated **SSL (Secure Sockets Layer)** specifications.
* It operates in the **presentation layer** and consists of two main protocols:
  + **TLS record protocol**: Handles data fragmentation, compression, and encryption.
  + **TLS handshake protocol**: Establishes a secure connection, negotiates cipher suites, and generates session keys.
* **TLS 1.3** is the most recent version, published in 2018.



**The Strengths of TLS Security**

1. **Encryption**: Hides data from third parties during transmission.
2. **Authentication**: Ensures that communicating parties are who they claim to be.
3. **Integrity**: Verifies that data remains unaltered during transmission.

**How Things Can Go Wrong**

* **Vulnerabilities**: TLS has had its share of flaws, including attacks like **POODLE**, **BEAST**, and **Heartbleed**.
* **Downgrade Attacks**: Attackers can force a downgrade to weaker protocols (e.g., SSL 3.0) to exploit vulnerabilities.
* **Configuration Mistakes**: Incorrect settings or outdated protocols can compromise security.

**Chapiter 14 : Quantum and Post-Quantum**

**How Quantum Computers Work**

* **Quantum computers** leverage the principles of **quantum mechanics** to perform computations.
* Unlike classical computers that use bits (0s and 1s), quantum computers use **qubits** (quantum bits).
* Key features of qubits:
  + **Superposition**: A qubit can exist in multiple states simultaneously.
  + **Entanglement**: Qubits can be correlated even when separated by large distances.
* Quantum gates manipulate qubits, allowing complex operations.
* Quantum algorithms exploit superposition and entanglement for specific tasks.

**Quantum Speed-Up**

* Quantum computers offer the potential for **exponential speed-up** in solving certain problems.
* **Shor’s algorithm** can factor large numbers efficiently, threatening classical RSA encryption.
* **Grover’s algorithm** accelerates searching unsorted databases.

**Why Is It So Hard to Build a Quantum Computer?**

1. **Decoherence**: Qubits are sensitive to their environment and easily lose coherence.
2. **Error Correction**: Quantum error correction is challenging due to noise and interactions.
3. **Physical Implementation**: Building stable qubits (e.g., superconducting circuits, trapped ions) is complex.
4. **Scalability**: Scaling up to many qubits while maintaining coherence is difficult.

**Post-Quantum Cryptographic Algorithms**

* **Post-quantum cryptography** aims to develop algorithms resistant to quantum attacks.
* NIST has selected the first group of **quantum-resistant encryption algorithms** based on structured lattices and hash functions.
* These algorithms will become part of NIST’s post-quantum cryptographic standard.

**How Things Can Go Wrong**

* **Vulnerabilities**: Quantum computers could break existing cryptographic systems.
* **Downgrade Attacks**: Attackers might force a downgrade to weaker protocols.
* **Implementation Flaws**: Incorrect implementations may weaken security.

**Why I Chose This Book :**

***first* : To Understand**

1. **Security Protocols : How it works**
2. **Data Protection : To understand how data protection is happening**
3. **Authentication and Authorization : Privecy**
4. **Quantum Threats : Quantum Computers is the next Generation of Tech**

***second* : I want to make a python software for chat with best encryption algorithm**

**Resources that helped me to Understand Some Concepts in This Book:**

1. **Youtube : iTeam Acedemy, Python Arabic Community**
2. **Websites :** [**www.deviceauthority.com**](http://www.deviceauthority.com) **,** [**www.freecodecamp.org**](http://www.freecodecamp.org)

**MINI PROJECT USING PYTHON PRORAMMING LANGUAGE**

****

**Code with example : https://github.com/senani-derradji/Encrypt\_Decrypt/blob/main/Algo.ipynb**