🔐CRYPTOGRAPHY

Cryptography is the study and art of hiding meaningful information in an unreadable format. Cryptography and cryptographic (“crypto”) systems help in securing data from interception and compromise during online transmissions. Cryptography enables one to secure transactions, communications, and other processes performed in the electronic world, and is additionally used to protect confidential data such as email messages, chat sessions, web transactions, personal data, corporate data, e-commerce applications, etc.

Types of Keys in Cryptography

Cryptographic keys are fundamental to the security of cryptographic systems. They are secret values used by algorithms to transform plaintext into ciphertext and vice versa. The type of key used depends on the cryptographic method and the security requirements.

---

1. Symmetric Keys (Secret Keys)

Definition:

A **symmetric key** is a single key used for both encryption and decryption. Both the sender and receiver must share this key securely before communication.

Characteristics:

- Same key for encryption and decryption.
- Key must be kept secret by both parties.
- Key distribution is a major challenge.
- Typically used for bulk data encryption due to efficiency.

Examples:

- AES key (128, 192, or 256 bits)
- DES key (56 bits)
- Blowfish key (variable length)

Use Cases:

- Encrypting files or databases.
- Securing communication channels (e.g., VPNs).
- Wireless network encryption (WPA2 uses symmetric keys).

Key Management Issues:

- Securely sharing the key before communication.

- Key storage and protection.
- Key rotation and expiration policies.

---

2. Asymmetric Keys (Public and Private Keys)

Definition:

Asymmetric cryptography uses a **key pair**: a **public key** and a **private key**. The public key is shared openly, while the private key is kept secret.

Characteristics:

- Public key encrypts data; private key decrypts it (or vice versa for signatures).
- Solves the key distribution problem of symmetric keys.
- Computationally more intensive than symmetric keys.
- Enables digital signatures and secure key exchange.

Key Pair Components:

- **Public Key:** Used to encrypt data or verify signatures. Can be distributed widely.
- **Private Key:** Used to decrypt data or create signatures. Must be kept secret.

Examples:

- RSA key pair (commonly 2048 or 4096 bits)
- ECC key pair (e.g., 256-bit keys providing strong security)
- DSA key pair (used for digital signatures)

Use Cases:

- Secure email (PGP/GPG).
- SSL/TLS certificates for websites.
- Digital signatures for document authentication.
- Secure key exchange protocols (e.g., Diffie-Hellman).

Key Management:

- Private key protection is critical.
- Public keys are often distributed via certificates.
- Revocation and renewal of keys via PKI.

---

3. Session Keys

Definition:

A **session key** is a temporary symmetric key used for encrypting data during a single communication session.

Characteristics:

- Generated for each session.
- Used to encrypt data efficiently during the session.
- Discarded after the session ends.
- Often exchanged securely using asymmetric cryptography.

Use Cases:

- SSL/TLS sessions use asymmetric cryptography to exchange a session key.
- Secure messaging apps generate session keys for each chat session.
- VPN tunnels use session keys for data encryption.

Benefits:

- Limits the amount of data encrypted with one key.
- Reduces risk if a session key is compromised.
- Enhances forward secrecy when combined with ephemeral key exchange.

---

4. Private Keys

Definition:

In asymmetric cryptography, the **private key** is the secret key used to decrypt data or create digital signatures.

Characteristics:

- Must be kept confidential.
- Stored securely, often in hardware security modules (HSMs) or encrypted storage.
- Loss or compromise leads to loss of security.

Use Cases:

- Decrypting messages encrypted with the corresponding public key.
- Signing documents or transactions to prove authenticity.

---

5. Public Keys

Definition:

The **public key** is the non-secret key in asymmetric cryptography used to encrypt data or verify signatures.

Characteristics:

- Can be freely distributed.
- Often embedded in digital certificates.
- Used by anyone to send encrypted messages or verify signatures.

---

6. Private/Public Key Pairs in Detail

How They Work Together:

- **Encryption:** Sender encrypts data with receiver's public key; only receiver's private key can decrypt.
- **Digital Signature:** Sender signs data with private key; anyone can verify with sender's public key.

Key Generation:

- Generated together mathematically.
- Must be large enough to prevent brute force or cryptanalysis attacks.

---

7. Other Specialized Keys

7.1. Master Key

- A key used to derive other keys.
- Common in hierarchical key management systems.
- Example: In disk encryption, a master key encrypts volume keys.

7.2. Key Encryption Key (KEK)

- Used specifically to encrypt other keys.
- Helps protect keys during storage or transmission.

7.3. Initialization Vector (IV) / Nonce

- Not a key but used alongside keys.
- Ensures that encryption produces different ciphertexts even for identical plaintexts.
- Prevents replay attacks and pattern analysis.

---

8. Key Length and Security

- Key length directly impacts security.
- Longer keys generally mean stronger security but slower performance.
- Symmetric keys: 128 bits or more recommended.

- Asymmetric keys: RSA 2048 bits or ECC 256 bits considered secure.
- Key length must balance security and performance.

9. Summary Table

Key Type	Usage	Key Sharing	Example Algorithms	Key Length Typical
Symmetric Key	Encrypt/decrypt data	Shared secretly	AES, DES, Blowfish	128, 192, 256 bits
Public Key	Encrypt data, verify signature	Publicly shared	RSA, ECC, DSA	RSA: 2048-4096 bits, ECC: 256 bits
Private Key	Decrypt data, sign data	Kept secret	RSA, ECC, DSA	Same as public key
Session Key	Temporary symmetric key	Shared per session	AES (usually)	128-256 bits
Master Key	Derive other keys	Kept secret	Used in key management	Varies
Key Encryption Key	Encrypt other keys	Kept secret	Used in key wrapping	Varies

Encryption and Types of Encryption

1. What is Encryption?

Encryption is the process of converting plaintext (readable data) into ciphertext (unreadable data) using an algorithm and a key. The purpose of encryption is to protect the confidentiality of data by making it inaccessible to unauthorized users.

- **Plaintext:** Original readable data.
- **Ciphertext:** Encrypted, unreadable data.
- **Encryption Algorithm (Cipher):** The mathematical procedure used to transform plaintext into ciphertext.
- **Key:** A secret value used by the encryption algorithm to perform the transformation.

Purpose of Encryption:

- Protect data confidentiality.
- Secure communication over insecure channels.
- Prevent unauthorized access to sensitive information.

2. How Encryption Works (Basic Process)

1. **Encryption:**

   Plaintext + Key → Encryption Algorithm → Ciphertext

2. **Decryption:**

   Ciphertext + Key → Decryption Algorithm → Plaintext

The key used in encryption and decryption depends on the type of encryption (symmetric or asymmetric).

---

3. Types of Encryption

Encryption can be broadly classified into two main types:

3.1 Symmetric Encryption (Secret Key Encryption)

Overview:

- Uses the **same key** for both encryption and decryption.
- Both sender and receiver must share the secret key securely.
- Fast and efficient, suitable for encrypting large volumes of data.

How It Works:

- Sender encrypts plaintext using the secret key.
- Receiver decrypts ciphertext using the same secret key.

Common Algorithms:

- **AES (Advanced Encryption Standard):** Most widely used symmetric cipher; supports 128, 192, and 256-bit keys.
- **DES (Data Encryption Standard):** Older standard with 56-bit key; now considered insecure.
- **3DES (Triple DES):** Applies DES three times for better security.
- **Blowfish:** Fast and flexible key length.
- **RC4:** Stream cipher, now deprecated due to vulnerabilities.

Use Cases:

- Encrypting files and databases.
- Securing communication channels (VPNs, Wi-Fi).
- Bulk data encryption.

Advantages:

- High speed and efficiency.
- Simple to implement.

Disadvantages:

- Key distribution problem: securely sharing the key is challenging.
- If the key is compromised, all communication is compromised.

3.2 Asymmetric Encryption (Public Key Encryption)

Overview:

- Uses a **pair of keys**: a **public key** and a **private key**.
- Public key encrypts data; private key decrypts it.
- Solves the key distribution problem of symmetric encryption.
- Computationally more intensive and slower than symmetric encryption.

How It Works:

- Sender encrypts data using the receiver's public key.
- Receiver decrypts data using their private key.
- For digital signatures, sender signs data with private key; receiver verifies with public key.

Common Algorithms:

- **RSA (Rivest-Shamir-Adleman):** Based on factoring large integers.
- **ECC (Elliptic Curve Cryptography):** Uses elliptic curves for smaller keys with equivalent security.
- **ElGamal:** Based on discrete logarithms.
- **DSA (Digital Signature Algorithm):** Used for digital signatures.

Use Cases:

- Secure email (PGP/GPG).
- SSL/TLS for secure web browsing.
- Digital signatures and certificates.
- Secure key exchange.

Advantages:

- Solves key distribution problem.
- Enables digital signatures and authentication.

Disadvantages:

- Slower than symmetric encryption.
- Computationally intensive.

3.3 Hybrid Encryption

Overview:

- Combines symmetric and asymmetric encryption.
- Asymmetric encryption is used to securely exchange a symmetric session key.
- Symmetric encryption is used for encrypting the actual data.

How It Works:

- Sender generates a random symmetric session key.
- Session key is encrypted with receiver's public key.
- Receiver decrypts session key with private key.
- Both use session key for fast symmetric encryption of data.

Use Cases:

- SSL/TLS protocols.
- Secure messaging apps.
- Encrypted file transfer.

Advantages:

- Efficient and secure.
- Solves key distribution problem.
- Fast encryption/decryption for large data.

---

4. Types of Encryption Based on Data Processing

4.1 Block Cipher Encryption

- Encrypts data in fixed-size blocks (e.g., 64 or 128 bits).
- Each block is encrypted separately.
- Common block cipher modes include ECB, CBC, CFB, OFB, and GCM.

Examples:

- AES (128-bit block size)
- DES (64-bit block size)
- 3DES

Characteristics:

- Suitable for encrypting large data.
- Modes of operation add security and flexibility.

4.2 Stream Cipher Encryption

- Encrypts data one bit or byte at a time.
- Generates a pseudorandom keystream combined with plaintext using XOR.
- Typically faster and used for real-time data encryption.

Examples:

- RC4 (deprecated)
- Salsa20
- ChaCha20

Characteristics:

- Suitable for streaming data (e.g., video, audio).
- Requires a unique key and initialization vector (IV) for each session.

---

5. Modes of Operation for Block Ciphers

Block ciphers require modes of operation to securely encrypt data larger than one block.

Common Modes:

- **ECB (Electronic Codebook):** Encrypts each block independently; vulnerable to pattern attacks.
- **CBC (Cipher Block Chaining):** Each plaintext block is XORed with the previous ciphertext block before encryption; requires an IV.
- **CFB (Cipher Feedback):** Converts block cipher into a stream cipher.
- **OFB (Output Feedback):** Similar to CFB but generates keystream blocks independently.
- **GCM (Galois/Counter Mode):** Provides confidentiality and data integrity (authenticated encryption).

---

6. Key Concepts in Encryption

6.1 Key Length

- Longer keys provide stronger security.
- Symmetric keys: 128 bits or more recommended.
- Asymmetric keys: RSA 2048 bits or ECC 256 bits considered secure.

6.2 Initialization Vector (IV)

- A random or unique value used with encryption algorithms to ensure different ciphertexts for identical plaintexts.
- Prevents replay and pattern attacks.

6.3 Salt

- Random data added to plaintext before encryption or hashing to prevent dictionary attacks.

7. Summary Table

Encryption Type	Key Type	Data Processing	Speed	Use Cases	Examples
Symmetric Encryption	Single secret key	Block or Stream	Fast	File encryption, VPNs	AES, DES, Blowfish
Asymmetric Encryption	Public/private key	Block	Slower	Secure email, SSL/TLS	RSA, ECC, DSA
Hybrid Encryption	Both symmetric & asymmetric	Block/Stream	Efficient	Secure web browsing, messaging	SSL/TLS
Block Cipher	Secret or public/private key	Fixed-size blocks	Moderate	Bulk data encryption	AES, DES, 3DES
Stream Cipher	Secret key	Bit/byte stream	Very fast	Real-time data encryption	RC4 (deprecated), ChaCha20

HASH FUNCTIONS

Hash functions calculate a unique fixed-size bit string representation, called a message digest, of any arbitrary block of information. Message digest (One-way Hash) functions distill the information contained in a file (small or large) into a single fixed-length number, typically between 128 and 256 bits. If any given bit of the function's input is changed, every output bit has a 50% chance of changing. Given an input file and its corresponding message digest, it should be nearly impossible to find another file with the same message digest value, as it is computationally infeasible to have two files with the same message digest value.

MD2, MD4, MD5, and MD6 are message digest algorithms used in digital signature applications to compress documents securely before the system signs it with a private key. The algorithms can be of variable length, but the resulting message digest is always 128 bits.

The MD5 algorithm is a widely used cryptographic hash function that takes a message of arbitrary length as input and outputs a 128-bit (16-byte) fingerprint or message digest of the input. The MD5 algorithm is used in a wide variety of cryptographic applications and is useful for digital signature applications, file integrity checking, and storing passwords.

Self-signed certificates are widely used for testing servers. In self-signed certificates, a user creates a pair of public and private keys using a certificate creation tool such as Adobe Acrobat Reader, Java's keytool, Apple's Keychain, etc. and signs the document with the public key. The recipient requests the private key from the sender in order to verify the certificate. However, certificate verification rarely occurs due to the necessity of disclosing the private key: this makes self-signed certificates useful only in a self-controlled testing environment.

Email encryption hides the email content from eavesdroppers by encrypting it into an unreadable form. Emails can be encrypted and decrypted by means of a digital signature mechanism that uses public and private keys: the public key is shared, while the private key is kept private.

There are numerous methods that can be employed for email encryption, including:

- Digital Signature: Uses asymmetric cryptography to simulate the security properties of a signature in digital, rather than written form
- Secure Sockets Layer (SSL): Uses RSA asymmetric (public key) encryption to encrypt data transferred over SSL connections
- Transport Layer Security (TLS): Uses a symmetric key for bulk encryption, an asymmetric key for authentication and key exchange, and message authentication codes for message integrity
- Pretty Good Privacy (PGP): Used to encrypt and decrypt data that provides authentication and cryptographic privacy
- GNU Privacy Guard (GPG): Software replacement of PGP and free implementation of the OpenPGP standard that is used to encrypt and decrypt data

Hands-On Labs and Practical Exercises on Encryption

Lab 1: Symmetric Encryption with AES (Python)

Objective:

Encrypt and decrypt a message using AES symmetric encryption.

Tools:

- Python 3
- `pycryptodome` library (`pip install pycryptodome`)

Steps:

```
pythonCopy
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Generate a random 16-byte key (128-bit)
key = get_random_bytes(16)

# Create AES cipher in CBC mode
cipher = AES.new(key, AES.MODE_CBC)
```

```
# Plaintext message
plaintext = b'This is a secret message.'

# Pad plaintext to block size (16 bytes)
padded_text = pad(plaintext, AES.block_size)

# Encrypt
ciphertext = cipher.encrypt(padded_text)

print("Ciphertext (hex):", ciphertext.hex())

# Decrypt
decipher = AES.new(key, AES.MODE_CBC, cipher.iv)
decrypted_padded = decipher.decrypt(ciphertext)

# Unpad decrypted text
decrypted = unpad(decrypted_padded, AES.block_size)

print("Decrypted text:", decrypted.decode())
```

What You Learn:

- Key generation
- Padding for block ciphers
- Encryption and decryption with AES in CBC mode

Lab 2: Asymmetric Encryption with RSA (Python)

Objective:

Generate RSA keys, encrypt a message with the public key, and decrypt with the private key.

Tools:

- Python 3
- `pycryptodome` library

Steps:

```
pythonCopy
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# Generate RSA key pair
key = RSA.generate(2048)
```

```
private_key = key.export_key()
public_key = key.publickey().export_key()

print("Private Key:", private_key.decode())
print("Public Key:", public_key.decode())

# Encrypt with public key
message = b'Encrypt this message with RSA.'
public_key_obj = RSA.import_key(public_key)
cipher_rsa = PKCS1_OAEP.new(public_key_obj)
ciphertext = cipher_rsa.encrypt(message)

print("Ciphertext (hex):", ciphertext.hex())

# Decrypt with private key
private_key_obj = RSA.import_key(private_key)
decipher_rsa = PKCS1_OAEP.new(private_key_obj)
decrypted = decipher_rsa.decrypt(ciphertext)

print("Decrypted message:", decrypted.decode())
```

What You Learn:

- RSA key pair generation
- Public key encryption and private key decryption
- Use of padding scheme (OAEP)

Lab 3: Hashing and Integrity Check (Python)

Objective:

Generate a SHA-256 hash of a message and verify integrity.

Tools:

- Python 3
- `hashlib` (built-in)

Steps:

```
pythonCopy
import hashlib

message = b'This message needs integrity check.'

# Generate SHA-256 hash
```

```
hash_object = hashlib.sha256(message)
hash_digest = hash_object.hexdigest()

print("SHA-256 Hash:", hash_digest)

# Verify integrity by hashing again and comparing
received_message = b'This message needs integrity check.'
received_hash = hashlib.sha256(received_message).hexdigest()

if received_hash == hash_digest:
    print("Integrity verified: message is unchanged.")
else:
    print("Integrity check failed: message has been altered.")
```

What You Learn:

- Hash generation
- Integrity verification using hashes

Lab 4: Hybrid Encryption (Combining RSA and AES)

Objective:

Encrypt a message using AES symmetric encryption, then encrypt the AES key with RSA public key.

Tools:

- Python 3
- `pycryptodome`

Steps:

```
pythonCopy
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP, AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad

# Generate RSA keys
rsa_key = RSA.generate(2048)
public_key = rsa_key.publickey()
private_key = rsa_key

# Generate AES session key
session_key = get_random_bytes(16)
```

```
# Encrypt the session key with RSA public key
cipher_rsa = PKCS1_OAEP.new(public_key)
enc_session_key = cipher_rsa.encrypt(session_key)

# Encrypt the message with AES session key
data = b'Hybrid encryption example message.'
cipher_aes = AES.new(session_key, AES.MODE_CBC)
ciphertext = cipher_aes.encrypt(pad(data, AES.block_size))

print("Encrypted session key (hex):", enc_session_key.hex())
print("Ciphertext (hex):", ciphertext.hex())

# Decrypt the session key with RSA private key
cipher_rsa_dec = PKCS1_OAEP.new(private_key)
dec_session_key = cipher_rsa_dec.decrypt(enc_session_key)

# Decrypt the message with AES session key
cipher_aes_dec = AES.new(dec_session_key, AES.MODE_CBC, cipher_aes.iv)
plaintext = unpad(cipher_aes_dec.decrypt(ciphertext), AES.block_size)

print("Decrypted message:", plaintext.decode())
```

What You Learn:

- Combining asymmetric and symmetric encryption
- Secure key exchange using RSA
- Efficient data encryption using AES

Lab 5: Using OpenSSL Command Line for Encryption

Objective:

Encrypt and decrypt files using OpenSSL symmetric and asymmetric encryption.

Tools:

- OpenSSL installed on your system (Linux, macOS, Windows with WSL or native)

Symmetric Encryption (AES-256-CBC):

```
bashCopy
# Encrypt a file
openssl enc -aes-256-cbc -salt -in plaintext.txt -out encrypted.bin
```

```
# Decrypt the file
openssl enc -d -aes-256-cbc -in encrypted.bin -out decrypted.txt
```

Asymmetric Encryption (RSA):

```
bashCopy
# Generate RSA private key
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

# Extract public key
openssl rsa -pubout -in private_key.pem -out public_key.pem

# Encrypt file with public key
openssl rsautl -encrypt -inkey public_key.pem -pubin -in plaintext.txt -out encrypted.bin

# Decrypt file with private key
openssl rsautl -decrypt -inkey private_key.pem -in encrypted.bin -out decrypted.txt
```

What You Learn:

- File encryption and decryption using OpenSSL
- RSA key generation and usage
- Practical command-line cryptography

Cryptanalysis: Overview and Tools

1. What is Cryptanalysis?

Cryptanalysis is the science and art of analyzing and breaking cryptographic systems. The goal is to find weaknesses or vulnerabilities in encryption algorithms or implementations to recover plaintext, keys, or other secret information without having direct access to the key.

- It is the counterpart to cryptography (which is about creating secure systems).
- Cryptanalysis helps evaluate the strength of cryptographic algorithms.
- It is used by security researchers, penetration testers, and attackers.

---

2. Goals of Cryptanalysis

- **Recover plaintext** from ciphertext without the key.
- **Recover the secret key** used in encryption.
- **Find weaknesses** in cryptographic algorithms or protocols.
- **Bypass authentication** or digital signatures.

- **Exploit implementation flaws** (side-channel attacks).

---

3. Types of Cryptanalysis Attacks

3.1 Ciphertext-only Attack

- Attacker has access only to ciphertext.
- Attempts to deduce plaintext or key from ciphertext patterns.

3.2 Known-plaintext Attack

- Attacker has pairs of plaintext and corresponding ciphertext.
- Uses this information to deduce the key or decrypt other ciphertexts.

3.3 Chosen-plaintext Attack

- Attacker can choose arbitrary plaintexts and obtain their ciphertexts.
- Useful for analyzing how plaintext affects ciphertext.

3.4 Chosen-ciphertext Attack

- Attacker can choose ciphertexts and obtain their decrypted plaintexts.
- Used to exploit decryption oracles.

3.5 Side-channel Attack

- Exploits physical implementation leaks (timing, power consumption, electromagnetic leaks).

---

4. Common Cryptanalysis Techniques

- **Brute Force:** Trying all possible keys.
- **Frequency Analysis:** Analyzing letter or bit frequency in ciphertext (effective against classical ciphers).
- **Differential Cryptanalysis:** Studying how differences in plaintext affect ciphertext differences.
- **Linear Cryptanalysis:** Using linear approximations to describe the behavior of the block cipher.
- **Algebraic Attacks:** Using algebraic equations to solve for keys.
- **Statistical Attacks:** Using statistical properties of plaintext and ciphertext.
- **Side-channel Attacks:** Timing, power analysis, fault injection.

---

5. Popular Cryptanalysis Tools

5.1 Hashcat

- **Purpose:** Password cracking tool using brute force, dictionary, and rule-based attacks.
- **Use:** Cracks hashed passwords (MD5, SHA, bcrypt, etc.).
- **Platform:** Cross-platform (Windows, Linux, macOS).
- **Website:** **https://hashcat.net/hashcat/**

5.2 John the Ripper

- **Purpose:** Password cracker supporting many hash types.
- **Use:** Dictionary attacks, brute force, and hybrid attacks.
- **Platform:** Cross-platform.
- **Website:** **https://www.openwall.com/john/**

5.3 Aircrack-ng

- **Purpose:** Wireless network security tool.
- **Use:** Cracks WEP and WPA/WPA2-PSK keys using captured packets.
- **Platform:** Linux, Windows.
- **Website:** **https://www.aircrack-ng.org/**

5.4 Cain and Abel

- **Purpose:** Password recovery tool for Windows.
- **Use:** Cracks various password hashes, performs network sniffing, and cryptanalysis.
- **Platform:** Windows.
- **Note:** Development discontinued but still used.

5.5 CrypTool

- **Purpose:** Educational tool for learning cryptography and cryptanalysis.
- **Use:** Visualizes cryptographic algorithms and attacks.
- **Platform:** Windows.
- **Website:** **https://www.cryptool.org/en/**

5.6 GPG/PGP Tools

- **Purpose:** Encrypt/decrypt and analyze PGP encrypted data.
- **Use:** Can be used to test cryptographic implementations.
- **Platform:** Cross-platform.

5.7 RsaCtfTool

- **Purpose:** Automated tool to find vulnerabilities in RSA keys.
- **Use:** Factorization, common modulus attacks, low exponent attacks.

- **Platform:** Python-based.
- **GitHub:** https://github.com/Ganapati/RsaCtfTool

5.8 CyberChef

- **Purpose:** Web-based tool for cryptanalysis and data transformation.
- **Use:** Encoding/decoding, encryption/decryption, hash cracking.
- **Platform:** Browser-based.
- **Website:** https://gchq.github.io/CyberChef/

---

6. Practical Cryptanalysis Examples

- **Frequency Analysis on Classical Ciphers:** Using letter frequency to break substitution ciphers.
- **Brute Force on Weak Keys:** Trying all keys on DES (56-bit key) to recover plaintext.
- **Side-channel Timing Attack:** Measuring decryption time to infer private key bits.
- **RSA Key Factorization:** Using tools like RsaCtfTool to factor weak RSA keys.

---

7. Summary Table of Tools

Tool Name	Purpose	Platform	Notes
Hashcat	Password hash cracking	Cross-platform	GPU accelerated
John the Ripper	Password cracking	Cross-platform	Supports many hash types
Aircrack-ng	Wireless key cracking	Linux, Windows	Focus on Wi-Fi security
Cain and Abel	Password recovery	Windows	Network sniffing, cracking
CrypTool	Educational cryptanalysis	Windows	Visualizes crypto algorithms
RsaCtfTool	RSA vulnerability analysis	Python	Automated RSA attacks
CyberChef	Data transformation & analysis	Browser	Easy to use, many functions

---

Practical Cryptanalysis Labs

Lab 1: Password Cracking with Hashcat

Objective:

Crack a hashed password using Hashcat with a dictionary attack.

Prerequisites:

- Hashcat installed (**Download**)
- A hash to crack (e.g., MD5 hash of a password)

- A wordlist (e.g., rockyou.txt)

Steps:

1. **Get a sample hash**

   Example MD5 hash for password "password123":
   `482c811da5d5b4bc6d497ffa98491e38`

2. **Prepare your wordlist**

   Download the rockyou.txt wordlist (commonly used in password cracking).

3. **Run Hashcat**

   Open terminal/command prompt and run:

   ```
   bashCopy
   hashcat -m 0 -a 0 482c811da5d5b4bc6d497ffa98491e38 /path/to/rockyou.txt
   ```

   - `m 0` specifies MD5 hash type.
   - `a 0` specifies dictionary attack.
   - Replace `/path/to/rockyou.txt` with the actual path.

4. **View results**

   Hashcat will try passwords from the wordlist and display the cracked password if found.

Lab 2: Password Cracking with John the Ripper

Objective:

Crack a password hash using John the Ripper.

Prerequisites:

- John the Ripper installed (**Download**)
- A file containing the hash (e.g., `hashes.txt`)

Steps:

1. **Create a file with the hash**

   Create `hashes.txt` containing:

   ```
   Copy
   482c811da5d5b4bc6d497ffa98491e38
   ```

2. **Run John the Ripper**

   In terminal:

```
bashCopy
john --format=raw-md5 hashes.txt --wordlist=/path/to/rockyou.txt
```

3. **Check cracked passwords**

   After some time, run:

   ```
   bashCopy
   john --show hashes.txt
   ```

   It will display the cracked password.

Lab 3: Frequency Analysis on a Substitution Cipher (Python)

Objective:

Analyze ciphertext letter frequency to help decrypt a substitution cipher.

Steps:

1. Use this Python script:

```
pythonCopy
from collections import Counter

ciphertext = "Wklv lv d vhfuhw phvvdjh."

# Count letter frequency
freq = Counter(c.lower() for c in ciphertext if c.isalpha())

# Sort by frequency
sorted_freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

print("Letter frequencies:")
for letter, count in sorted_freq:
    print(f"{letter}: {count}")
```

1. Run the script and observe the frequency of letters.
2. Compare with typical English letter frequency (e.g., ETAOIN SHRDLU) to guess substitutions.

Lab 4: RSA Key Vulnerability Testing with RsaCtfTool

Objective:

Test an RSA key for common vulnerabilities and attempt to recover the private key.

Prerequisites:

- Python 3 installed
- RsaCtfTool installed (`git clone https://github.com/Ganapati/RsaCtfTool.git` and install dependencies)

Steps:

1. **Obtain an RSA public key** (or generate a weak one for testing).
2. **Run RsaCtfTool**:

```
bashCopy
python3 RsaCtfTool.py --publickey public.pem --attack all
```

1. The tool will try multiple attacks (factoring, common modulus, low exponent) to recover the private key.
2. If successful, it will output the private key and decrypted messages.

Overview of Cryptanalysis

Cryptanalysis can be performed using various methods, including the following:

- Linear Cryptanalysis: A known plaintext attack that uses a linear approximation to describe the behavior of the block cipher
- Differential Cryptanalysis: The examination of differences in an input and how this affects the resultant difference in the output
- Integral Cryptanalysis: This attack is useful against block ciphers based on substitution-permutation networks and is an extension of differential cryptanalysis

Perform Cryptanalysis using CrypTool

- **CrypTool** is a freeware program that enables you to apply and analyze cryptographic mechanisms, and has the typical look and feel of a modern Windows application. CrypTool includes a multitude of state-of-the-art cryptographic functions and allows you to both learn and use cryptography within the same environment. CrypTool is a free, open-source e-learning application used in the implementation and analysis of cryptographic algorithms.

Perform Cryptanalysis using AlphaPeeler

- **AlphaPeeler** is a powerful tool for learning cryptology. It can be useful as an instructor's teaching aid and to create assignments for classical cryptography. You can easily learn classical techniques such as frequency analysis of alphabets, mono-alphabetic substitution, Caesar cipher, transposition cipher, Vigenere cipher, and Playfair cipher. AlphaPeeler Professional (powered by crypto++ library) also includes DES, Gzip/Gunzip, MD5, SHA-1, SHA-256, RIPEMD-16, RSA key generation, RSA crypto, RSA signature & validation, and generation of secret share files.