🔐 Cryptography & John the Ripper - Complete Guide

> 🎯 Comprehensive hash cracking and cryptography reference for security testing and password recovery

---

📋 Table of Contents

🔢 Basic Cryptography Calculations

🔑 RSA Key Generation

Foundational RSA Formulas Essential mathematical foundations for RSA encryption

Formula 1: Modulus Calculation

```
n = p × q
```

Definition Where:

- `p` = First large prime number
- `q` = Second large prime number
- `n` = RSA modulus (public)

Example:

```
p = 61
q = 53
n = 61 × 53 = 3233
```

Formula 2: Euler's Totient Function

```
ϕ(n) = (p−1) × (q−1)
```

ⓘ **Definition Used to calculate the totient of n for key generation**

Example:

```
p = 61, q = 53
ϕ(n) = (61−1) × (53−1)
ϕ(n) = 60 × 52 = 3120
```

Purpose:

- 🔑 Determines valid encryption/decryption exponents
- 🔐 Essential for private key calculation
- 🧮 Foundation of RSA security

🤝 Diffie-Hellman Key Exchange

✓ **Secure Key Exchange Protocol Allows two parties to establish shared secret over insecure channel**

Public Parameters

ⓘ **Shared Publicly p = Prime number (shared) g = Generator (shared)**

Both parties know p and g, but these are public information.

Private Keys

⚠ Keep Secret a = Alice's private key (secret) b = Bob's private key (secret)

Public Key Calculation

Alice calculates:

```
A = g^a mod p
```

Bob calculates:

```
B = g^b mod p
```

Exchange:

- Alice sends A to Bob → Bob receives A
- Bob sends B to Alice → Alice receives B

Shared Secret Calculation

Alice computes:

```
Secret = B^a mod p
```

Bob computes:

```
Secret = A^b mod p
```

Result:

```
Both arrive at same shared secret!
Secret = g^(ab) mod p
```

Example with Numbers

Public Parameters:

```
p = 23 (prime)
g = 5 (generator)
```

Private Keys:

```
Alice's private: a = 6
Bob's private: b = 15
```

Public Keys:

```
Alice: A = 5^6 mod 23 = 8
Bob: B = 5^15 mod 23 = 19
```

Shared Secret:

```
Alice: 19^6 mod 23 = 2
Bob: 8^15 mod 23 = 2

Shared Secret = 2 ✅
```

🔍 Hash Analysis Tools

📁 File Hash Generation

Hexdump Analysis

```
hexdump -C file1.txt
```

Purpose:

- 👁️ Display file content in hexadecimal
- 🔍 Analyze binary data
- 🐛 Debug file corruption
- 🔎 Examine file structure

Output Example:

```
00000000  48 65 6c 6c 6f 20 57 6f  72 6c 64 0a              |Hello World.|
0000000c
```

MD5 Hash Generation

```
# Single file
md5sum file1.txt

# Output
d41d8cd98f00b204e9800998ecf8427e  file1.txt
```

⚠ MD5 Deprecated MD5 is **not secure** for cryptographic purposes. Use for file integrity checks only.

SHA1 Hash Generation

```
# Multiple files
sha1sum *.txt

# Output
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d  file1.txt
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8  file2.txt
```

⚠ SHA-1 Deprecated SHA-1 is vulnerable to collision attacks. Use SHA-256 or higher.

🌐 Hash Cracking Resources

Online Hash Databases

✓ Quick Lookup Services Large databases of pre-computed hash lookups

Popular Services:

- 🌐 **hashes.com** - Multi-algorithm database
- 🔓 **crackstation.net** - Fast hash lookup
- 🔍 **md5decrypt.net** - MD5 focused
- 📊 **hashkiller.io** - Community database

Usage:

```
1. Copy hash: 5f4dcc3b5aa765d61d8327deb882cf99
2. Paste into website
3. Get result: password
```

Best For Common passwords and weak hashes. Won't work for salted or complex passwords.

🔐 Secure Hash Functions

Modern Recommended Algorithms Password hashing functions designed to resist cracking

Argon2 ⭐ Winner

Characteristics:

- 🏆 Winner of Password Hashing Competition (2015)
- 💪 Memory-hard algorithm
- ⚡ Configurable time and memory costs
- 🛡️ Resistant to GPU/ASIC attacks

Types:

- **Argon2d** - Maximizes resistance to GPU attacks
- **Argon2i** - Optimized for password hashing
- **Argon2id** - Hybrid (recommended)

Scrypt

Characteristics:

- 🧠 Memory-hard function
- 💾 Requires large amounts of memory
- 🛡️ Resistant to hardware attacks
- 🔐 Used by cryptocurrencies

Bcrypt

Characteristics:

- ⏰ Adaptive hash function
- 🔄 Adjustable cost factor
- 🐢 Intentionally slow
- ✅ Time-tested and reliable

Example:

```
$2b$10$N9qo8uLOickgx2ZMRZoMye IIvVyjrSSoD50ByxirDazFyRoNWZX2i
| |  |
| |  └─ Cost factor (2^10 iterations)
| └──── Bcrypt version
└────── Identifier
```

PBKDF2

Characteristics:

- 📜 RSA PKCS #5 standard
- 🔄 Configurable iterations
- 🏢 Widely used in enterprise
- ✅ FIPS approved

⚡ Hashcat - GPU Accelerated Cracking

✓ **High-Performance Password Recovery Industry-standard tool with GPU acceleration**

Basic Syntax

```
hashcat -m <hash_type> -a <attack_mode> hashfile wordlist
```

Parameters:

- `-m` → Hash type mode
- `-a` → Attack mode
- `hashfile` → File containing hashes
- `wordlist` → Dictionary file

Common Attack Modes

Mode	Name	Description
`0`	**Straight**	Direct wordlist attack
`1`	**Combination**	Combine words from wordlists
`3`	**Brute-force**	Try all character combinations
`6`	**Hybrid Wordlist + Mask**	Wordlist with character patterns
`7`	**Hybrid Mask + Wordlist**	Pattern with wordlist append

Practical Examples

MD5 Cracking:

```
hashcat -m 0 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt
```

NTLM Cracking:

```
hashcat -m 1000 -a 0 ntlm_hashes.txt rockyou.txt
```

Brute Force (8 chars, lowercase):

```
hashcat -m 0 -a 3 hashes.txt ?l?l?l?l?l?l?l?l
```

Masks:

```
?l = lowercase (a-z)
?u = uppercase (A-Z)
?d = digits (0-9)
?s = special characters
?a = all characters
```

Hash Type Reference (Common)

Hash Type	-m Value	Example
MD5	0	5f4dcc3b5aa765d61d8327deb882cf99
SHA-1	100	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
SHA-256	1400	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d
NTLM	1000	b4b9b02e6f09a9bd760f388b67351e2b
bcrypt	3200	LhayLxezLhK1LhWvKxCyLO...

🔍 Hash Identifier

ⓘ Automatic Hash Type Detection Identifies hash types to determine correct cracking approach

```
python3 hash-id.py
```

Interactive Mode:

```
HASH: 5f4dcc3b5aa765d61d8327deb882cf99

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))
```

Alternative Tools:

```
# hashid
hashid '5f4dcc3b5aa765d61d8327deb882cf99'

# hash-identifier (Kali)
hash-identifier
```

⚠ **Note Hash identification requires educated guessing. Context matters!**

⚔️ John the Ripper - Complete Guide

✓ **The Swiss Army Knife of Password Cracking Free, open-source password security auditing and recovery tool**

🎯 Basic John the Ripper Syntax

Standard Command Structure

```
john [options] [file path]
```

Common Options:

- `--wordlist=<path>` → Dictionary attack
- `--format=<type>` → Specify hash format
- `--show` → Display cracked passwords
- `--list=formats` → List supported formats

Essential Commands

Basic Wordlist Attack:

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
```

Specify Format:

```
john --format=raw-md5 --wordlist=rockyou.txt hash1.txt
```

Show Cracked Passwords:

```
john --show hash.txt
```

List Status:

```
john --show --left hash.txt
```

🔍 Format Discovery

> **Finding the Right Format John supports hundreds of hash formats**

```
# List all formats
john --list=formats

# Search for specific hash type
john --list=formats | grep -iF "md5"
john --list=formats | grep -iF "sha1"
john --list=formats | grep -iF "ntlm"
```

Example Output:

```
descrypt, bsdicrypt, md5crypt, md5crypt-long, bcrypt, scrypt, LM, AFS,
tripcode, AndroidBackup, adxcrypt, agilekeychain, aix-ssha1, aix-ssha256,
aix-ssha512, andOTP, ansible, argon2, as400-des, as400-ssha1, asa-md5,
AxCrypt, AzureAD, BestCrypt, bfegg, Bitcoin, BitLocker, bitshares, Bitwarden,
BKS, Blackberry-ES10, WoWSRP, Blockchain, chap, Clipperz, cloudkeychain,
...
```

📝 Practical Examples

MD5 Hash Cracking

```
# Create hash file
echo "5f4dcc3b5aa765d61d8327deb882cf99" > md5_hash.txt

# Crack with wordlist
john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt md5_hash.txt

# Show result
john --show --format=raw-md5 md5_hash.txt
```

SHA-1 Hash Cracking

```
# Find format
john --list=formats | grep -iF "sha1"

# Crack
john --format=raw-sha1 --wordlist=rockyou.txt sha1_hash.txt
```

Multiple Hash Types in One File

```
# John can auto-detect
john --wordlist=rockyou.txt mixed_hashes.txt

# Or specify format
john --format=dynamic --wordlist=rockyou.txt hashes.txt
```

🗂️ Hash Types & Specialized Attacks

🪟 NTLM/NTHash (Windows Systems)

ⓘ **Windows Authentication Hash Used by modern Windows operating systems**

Characteristics

Key Information:

- 🪟 Modern Windows OS default

- 💾 Stored in SAM database
- 🆔 Format: "NT" prefix
- 🔍 Easy to identify (no tool needed)
- ⚡ Fast to crack (no salting)

Hash Format:

```
Username:RID:LM_Hash:NTLM_Hash:::
```

Example:

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:b4b9b02e6f09a9bd760f388b673
51e2b:::
```

Attack Strategy

⚠ **Strategic Approach Don't waste time on strong passwords**

Decision Tree:

```
Check Password Policy
    │
    ├─ Weak Policy? → Attempt cracking
    │                  └─ Use rockyou.txt
    │                  └─ Try mutations
    │
    └─ Strong Policy? → Pass-the-Hash attack
                         └─ Don't crack, use hash directly
```

Pass-the-Hash:

```
# Using hash directly for authentication
pth-winexe -U
Administrator%aad3b435b51404eeaad3b435b51404ee:b4b9b02e6f09a9bd760f388b67351e2
b //10.10.10.10 cmd
```

Cracking NTLM with John

```
# Format for John
john --format=NT --wordlist=rockyou.txt ntlm_hashes.txt

# Show results
john --show --format=NT ntlm_hashes.txt
```

🐧 Linux Shadow Files

ⓘ **Linux Password Storage Passwords stored in /etc/shadow with various hash algorithms**

File Locations

System Files:

- 📁 `/etc/passwd` - User account information
- 🔐 `/etc/shadow` - Password hashes and aging info

File Permissions:

```
-rw-r--r--   /etc/passwd   (world readable)
-rw-------   /etc/shadow   (root only)
```

Understanding /etc/passwd

Format:

```
username:x:UID:GID:comment:home:shell
```

Example:

```
root:x:0:0:root:/root:/bin/bash
user:x:1000:1000:User Name:/home/user:/bin/bash
```

Fields:

- `x` → Password in /etc/shadow

- `0` → UID (0 = root)
- `0` → GID (primary group)

Understanding /etc/shadow

Format:

```
username:$id$salt$hash:lastchange:min:max:warn:inactive:expire
```

Example:

```
root:$6$Ha.d5nGup$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJ...:18576:::::
```

Hash ID Types:

```
$1$ = MD5
$2a$ = Bcrypt
$2y$ = Bcrypt
$5$ = SHA-256
$6$ = SHA-512
$y$ = yescrypt
```

🔧 Unshadow Process

✓ **Combining Files for John John requires merged passwd + shadow format**

Step-by-Step Process

Step 1: Use unshadow utility

```
unshadow [path to passwd] [path to shadow] > output.txt
```

Step 2: Practical example

```
unshadow /etc/passwd /etc/shadow > hash.txt
```

Step 3: Crack with John

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
```

Before Unshadow

/etc/passwd:

```
root:x:0:0::/root:/bin/bash
user:x:1000:1000::/home/user:/bin/bash
```

/etc/shadow:

```
root:$6$Ha.d5nGup$yugXSk24ZljLTAZZ...:18576::::::
user:$6$Bm29pY$HICSm3LJiRJpKfIf7lZQ...:18577::::::
```

After Unshadow

Combined format:

```
root:$6$Ha.d5nGup$yugXSk24ZljLTAZZ...:0:0::/root:/bin/bash
user:$6$Bm29pY$HICSm3LJiRJpKfIf7lZQ...:1000:1000::/home/user:/bin/bash
```

✓ **Ready for John Combined file contains all necessary information for cracking**

Complete Example

```
# Step 1: Copy files from target
scp root@target:/etc/passwd .
scp root@target:/etc/shadow .

# Step 2: Combine files
unshadow passwd shadow > linux_hashes.txt

# Step 3: Identify format (usually auto-detected)
```

```
head -1 linux_hashes.txt
# If starts with $6$, it's SHA-512

# Step 4: Crack
john --wordlist=/usr/share/wordlists/rockyou.txt linux_hashes.txt

# Step 5: Show cracked passwords
john --show linux_hashes.txt
```

🎯 Single Crack Mode

✓ **Intelligence-Based Attack John creates custom wordlist from available information**

How It Works

Process:

```
1. John analyzes username
2. Generates mutations:
   - mike → Mike, MIKE, miKE
   - mike → mike123, mike2023
   - mike → ekim (reverse)
   - mike → m1k3 (leetspeak)
3. Tries thousands of variations
4. No external wordlist needed
```

Command Syntax

```
john --single --format=[format] [path to file]
```

Examples:

```
# Automatic format detection
john --single hash.txt

# Specify format
john --single --format=raw-md5 hash.txt
```

```
# With specific format (Linux shadow)
john --single --format=sha512crypt linux_hashes.txt
```

Why Use Single Mode?

When to Use Best for:

- ✅ Personal accounts (username-based passwords)
- ✅ When no wordlist available
- ✅ Quick initial attempt
- ✅ Users who use their name in password

Common Patterns Caught:

```
john → john, John, JOHN, john123, john2023, john!
admin → admin, Admin, admin123, administrator
mike → mike, Mike, michael, mike1, mike_2023
```

Advantages

- 🚀 **Fast** - No huge wordlist to process
- 🧠 **Smart** - Intelligent mutations
- 📝 **Targeted** - Username-based guessing
- ✅ **Effective** - Catches weak passwords

🔧 File Format Conversion Tools

Specialized Crackers Convert protected files to John-compatible format

📦 Archive File Crackers

ZIP Files - zip2john

Password-Protected ZIP Archives Extract password hashes from ZIP files

Syntax:

```
zip2john [options] [zip file] > [output file]
```

Complete Process:

```
# Step 1: Convert ZIP to John format
zip2john protected.zip > zip_hash.txt

# Step 2: Crack the hash
john --wordlist=/usr/share/wordlists/rockyou.txt zip_hash.txt

# Step 3: Show password
john --show zip_hash.txt

# Step 4: Extract ZIP with found password
unzip protected.zip
```

Example Output:

```
protected.zip:$pkzip2$1*2*2*0*2a*1e*5c5a7b3f*0*42*0*2a*5c5a*3e6b*$/pkzip2$::pr
otected.zip
```

RAR Files - rar2john

Password-Protected RAR Archives Extract password hashes from RAR files

Syntax:

```
rar2john [rar file] > [output file]
```

Complete Process:

```
# Step 1: Convert RAR to John format
rar2john protected.rar > rar_hash.txt

# Step 2: Crack
```

```
john --wordlist=/usr/share/wordlists/rockyou.txt rar_hash.txt

# Step 3: Show password
john --show rar_hash.txt

# Step 4: Extract RAR
unrar x protected.rar
```

RAR Extraction:

```
# Extract with password
unrar x -p"password123" protected.rar

# List contents
unrar l protected.rar

# Test archive
unrar t protected.rar
```

🔑 SSH Private Key Cracking

ssh2john

⚠️ **Password-Protected SSH Keys Crack encrypted SSH private key passphrases**

Background:

- 🔐 SSH keys can be password-protected
- 📁 Usually named `id_rsa`, `id_dsa`, `id_ecdsa`, `id_ed25519`
- 🔒 Passphrase encrypts the private key

Syntax:

```
ssh2john [id_rsa private key file] > [output file]
```

Complete Process:

```
# Step 1: Convert SSH key to John format
ssh2john id_rsa > ssh_hash.txt

# Step 2: Crack the passphrase
john --wordlist=/usr/share/wordlists/rockyou.txt ssh_hash.txt

# Step 3: Show password
john --show ssh_hash.txt

# Step 4: Use SSH key with found passphrase
ssh -i id_rsa user@hostname
# Enter recovered passphrase when prompted
```

Example Encrypted SSH Key:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,A1B2C3D4E5F6...

MIIEowIBAAKCAQEA1234567890ABCDEF...
-----END RSA PRIVATE KEY-----
```

ⓘ **Protected Key Indicators If SSH key has `Proc-Type: 4,ENCRYPTED`, it's password-protected**

📊 Additional Converters

office2john

```
# Microsoft Office documents
office2john document.docx > office_hash.txt
john office_hash.txt
```

---

pdf2john

```
# Password-protected PDFs
pdf2john protected.pdf > pdf_hash.txt
john pdf_hash.txt
```

---

keepass2john

```
# KeePass database files
keepass2john Database.kdbx > keepass_hash.txt
john keepass_hash.txt
```

🖼️ Steganography Tools

ⓘ **Hidden Data Extraction Files hidden within other files**

🎨 Steghide

✓ **Multi-Format Steganography Tool Extract hidden files from images and audio**

Supported Formats

- 📸 **JPEG** - Images
- 🖼️ **BMP** - Bitmaps
- 🎵 **WAV** - Audio files
- 🔊 **AU** - Audio files

Basic Commands

Extract Hidden File:

```
steghide extract -sf image.jpg
```

With Password:

```
steghide extract -sf image.jpg -p password123
```

Get Information:

```
steghide info image.jpg
```

Embed File:

```
steghide embed -cf image.jpg -ef secret.txt
```

Complete Extraction Process

```
# Step 1: Check if file has embedded data
steghide info image.jpg

# Output if data present:
# "image.jpg":
#   format: jpeg
#   capacity: 3.5 KB
# Try to get information about embedded data ? (y/n) y
# Enter passphrase:

# Step 2: Extract without password
steghide extract -sf image.jpg

# Step 3: If password protected, try cracking
stegseek image.jpg /usr/share/wordlists/rockyou.txt

# Step 4: Manual extract with found password
steghide extract -sf image.jpg -p "foundpassword"
```

Additional Steganography Tools

Stegseek (Fast Steghide Cracker):

```
# Crack steghide password
stegseek image.jpg rockyou.txt
```

```
# Much faster than manual attempts
```

Binwalk (File Analysis):

```
# Analyze file for embedded content
binwalk image.jpg

# Extract all found files
binwalk -e image.jpg
```

Strings (Text Extraction):

```
# Extract readable strings
strings image.jpg | less

# Look for hidden messages or flags
strings image.jpg | grep -i "flag"
```

📚 Best Practices & Advanced Techniques

📖 Wordlist Management

Common Wordlist Locations

✓ **Standard Kali/Parrot Paths Pre-installed wordlists on security distros**

Standard Locations:

```
# RockYou - Most popular (14 million passwords)
/usr/share/wordlists/rockyou.txt

# SecLists - Comprehensive collection
/usr/share/seclists/Passwords/
/usr/share/seclists/Passwords/Common-Credentials/
/usr/share/seclists/Passwords/Leaked-Databases/

# Dirb wordlists
/usr/share/wordlists/dirb/common.txt
/usr/share/wordlists/dirb/big.txt
```

```
# FastTrack
/usr/share/wordlists/fasttrack.txt

# Metasploit
/usr/share/metasploit-framework/data/wordlists/
```

Wordlist Statistics

Wordlist	Lines	Size	Best For
rockyou.txt	14M	134MB	General purpose
fasttrack.txt	222	1.9KB	Quick wins
common.txt	4.7K	41KB	Common passwords
10-million-password-list-top-1000000.txt	1M	8.2MB	Top passwords

---

🛠️ Custom Wordlist Creation

CeWL (Website Scraper)

Custom Wordlists from Websites Generate targeted wordlists by scraping websites

Basic Usage:

```
cewl https://example.com -w custom_wordlist.txt
```

Advanced Options:

```
# Minimum word length 6, depth 2
cewl -m 6 -d 2 https://example.com -w wordlist.txt

# Include email addresses
cewl -e https://example.com -w wordlist.txt

# Follow external links
cewl -o https://example.com -w wordlist.txt
```

Parameters:

- `-m` → Minimum word length
- `-d` → Depth to spider
- `-w` → Output file
- `-e` → Include emails
- `-o` → Offsite links

Crunch (Pattern Generator)

✓ **Generate Custom Character Sets Create wordlists based on patterns and character sets**

Basic Syntax:

```
crunch <min> <max> [charset] [options]
```

Examples:

```
# 8-12 character lowercase
crunch 8 12 abcdefghijklmnopqrstuvwxyz -o wordlist.txt

# 6-8 digits
crunch 6 8 0123456789 -o pins.txt

# Pattern-based (@ = lowercase, , = uppercase, % = digit, ^ = special)
crunch 8 8 -t admin%%% -o admin_passwords.txt
# Generates: admin000, admin001, ..., admin999
```

Character Sets:

```
# Lowercase
crunch 8 8 -f /usr/share/crunch/charset.lst lalpha -o lower.txt

# Uppercase
crunch 8 8 -f /usr/share/crunch/charset.lst ualpha -o upper.txt

# Mixed
crunch 8 8 -f /usr/share/crunch/charset.lst mixalpha -o mixed.txt
```

File Size Warning Crunch can generate HUGE files! Use carefully:

```
crunch 8 8 -c 1000000  # Limit to 1M lines per file
```

Performance Optimization

John Configuration Tips

Speed Up Cracking Optimize John for better performance

1. Specify Format:

```
# Slower (auto-detect)
john hash.txt

# Faster (specified)
john --format=raw-md5 hash.txt
```

2. Use Multiple CPU Cores:

```
# Fork 4 processes
john --fork=4 --format=raw-md5 hash.txt

# Use all available cores
john --fork=$(nproc) hash.txt
```