www.ijifr.org Impact Factor(2025)::8.057 ISSN: 2347-1697
International Journal of Informative & Futuristic Research (IJIFR)
Volume - 12, Issue -10, June 2025
An Open-Access, Double-Blind, Peer-Reviewed, Refereed International Journal of Multidisciplinary Research

Unique Research Paper Identification (URPI)::IJIFR/V12/E10/005

Comprehensive Survey of Symmetric and Public-Key Cryptographic Algorithms: Foundations, Attacks, and Applications

Ami M. Shah[1] and Ashishkumar Gor[2]
[1,2] Assistant Professor,
Department of Computer Engineering,
Dharmsinh Desai University, Nadiad, Gujarat, India

Abstract: *This survey provides a comprehensive examination of modern cryptographic algorithms, spanning both symmetric-key and public-key paradigms. It delves into the structural design, mathematical foundations, operational principles, and known cryptanalytic vulnerabilities of widely adopted algorithms such as AES, ChaCha20, RSA, and Elliptic Curve Cryptography (ECC). The report highlights the trade-offs between computational efficiency and security strength across different use cases and deployment environments. Additionally, it analyzes modern modes of operation (e.g., GCM, Poly1305), digital signature schemes (e.g., ECDSA, EdDSA), and secure key exchange protocols. Emphasis is placed on emerging trends, including post-quantum cryptographic schemes like Kyber and Dilithium, which aim to ensure resilience against quantum adversaries. Real-world applications in secure messaging, VPNs, blockchain, and IoT are examined to demonstrate how cryptographic choices are influenced by domain-specific constraints such as latency, energy efficiency, and trust models. This survey aims to serve as a foundational reference for researchers and practitioners seeking to understand the evolving landscape of algorithmic cryptography and its practical implications.*

Keywords: Elliptic Curve Cryptography (ECC), Key Cryptography, Public Key Cryptography, RSA, Cryptanalysis, Secure Communication Protocols, Post-Quantum Cryptography (PQC)

1. Introduction

Modern cryptography underpins the security of digital communications, protecting everything from personal messages to financial transactions. This report provides a comprehensive survey of cryptographic algorithms, covering symmetric-key (secret-key) ciphers and public-key (asymmetric) techniques, along with their mathematical foundations, cryptanalytic vulnerabilities, performance considerations, and real-world use cases. Symmetric ciphers (like AES and ChaCha20) use a single shared secret key for encryption and decryption, offering high speed and efficiency for bulk data encryption [3][6]. Public-key algorithms (like RSA and Elliptic Curve Cryptography) use key pairs (public/private) to enable encryption, digital signatures, and key exchange without a pre-shared secret, at the cost of higher computational complexity. We review block and stream cipher constructions, modes of operation, and prominent algorithms in each class. We then examine asymmetric algorithms including RSA, DSA, ElGamal, and ECC-based schemes,

and how they are used in protocols (e.g., SSL/TLS, PKI, secure messaging)[6][26][27]. The mathematical underpinnings in number theory and algebra (modular arithmetic, discrete logarithms, finite fields, elliptic curves) are summarized to illuminate why these algorithms are secure[27]. We survey cryptanalysis techniques ranging from brute force and classical attacks (linear, differential) to advanced methods (algebraic attacks, side-channels, padding oracle exploits), highlighting known vulnerabilities and defenses. We compare algorithm performance and scalability, including security level vs. key size trade-offs and efficiency in hardware/software contexts, and provide a comparative table of key sizes and characteristics [27][28]. Finally, we discuss several application domains (secure messaging [1][5], VPNs[6][7], blockchain[9][11], IoT[16]) to illustrate how cryptographic algorithms are deployed in practice, and we include an overview of post-quantum cryptography to contextualize emerging trends [21][22].

2. Symmetric Key Cryptography

Symmetric-key cryptography uses a single secret key for both encryption and decryption of messages. It remains the workhorse of data encryption due to its speed and efficiency, especially for bulk data [27]. Symmetric ciphers are broadly classified into block ciphers (which operate on fixed-size blocks of bits using rounds of substitution and permutation) and stream ciphers (which generate a keystream to encrypt data bit-by-bit or byte-by-byte) [28]. To securely encrypt arbitrarily long messages, block ciphers are used with modes of operation that specify how blocks are chained or combined, while stream ciphers inherently produce a continuous stream of key material. We review prominent symmetric algorithms, common modes of operation, and their security and usage.

2.1 Block Ciphers

Block ciphers transform fixed-length plaintext blocks into ciphertext blocks of the same length under a secret key, through multiple rounds of mixing operations (substitutions and permutations). The **Advanced Encryption Standard (AES)** is the de facto symmetric cipher standard worldwide, selected by NIST in 2001 to replace the aging DES/3DES ciphers [3]. AES is a 128-bit block cipher with key lengths of 128, 192, or 256 bits (providing ~128, 192, or 256 bits of security respectively). It uses a substitution-permutation network with 10, 12, or 14 rounds (for 128/192/256-bit keys) and was designed for both security and performance. AES encryption is extremely fast especially with hardware support (e.g., Intel AES-NI instructions [29] give an 8× speedup, from ~28 cycles/byte down to ~3.5 cycles/byte). AES is widely regarded as highly secure; no practical attacks exist against the full 10-round AES-128 or its larger-key variants, aside from brute-force which is computationally infeasible (2^{128} possibilities for a 128-bit key). Indeed, 128-bit keys are considered *computationally secure* against brute force – trying all 2^{128} keys would require on the order of 10^{18} joules of energy (about 30 GW-years), far beyond realistic limits. AES's design includes strong nonlinear components: its S-box substitution was *specifically designed to resist linear and differential cryptanalysis* by minimizing correlations and differential propagation probabilities. This non-linearity and a complex key schedule make AES robust against known cryptanalytic attacks [27][30].

By contrast, **Triple DES (3DES)** is an older block cipher based on the DES algorithm (56-bit block cipher) applied three times. Standard 3DES uses three independent 56-bit DES keys (168-bit total, but with only 112 bits of effective security due to meet-in-the-middle attacks) [31]. 3DES was long used in finance and legacy systems but is now considered outdated: its small 64-bit block size makes it vulnerable to birthday attacks on large volumes of data (the Sweet32 attack) and its performance is much slower than AES. In fact, 3DES is being **deprecated and retired** from use – NIST has officially disallowed new uses of 3DES after 2023. Industry standards have moved to

AES for virtually all new systems, as AES-128 is both stronger and *faster* than 3DES on modern hardware.

Another notable block cipher is Blowfish [32], designed by Bruce Schneier in 1993 as a free alternative to DES. Blowfish has a 64-bit block size and a variable key length (32 up to 448 bits). It uses a 16-round Feistel network with complex key-dependent S-boxes. Blowfish gained popularity in software implementations and is still found in some encryption products. To date, the full 16-round Blowfish has no known practical cryptanalytic attacks (cryptanalysis has broken reduced-round versions). However, Blowfish's 64-bit block makes it susceptible to the same Sweet32 birthday attack issues if large amounts of data are encrypted under the same key. Consequently, Blowfish is not recommended for new designs encrypting very large files (>4 GB). Its successor Twofish (128-bit block, 128–256-bit keys) was a finalist in the AES competition but AES (Rijndael) was ultimately selected as the standard [32]. In summary, modern practice has consolidated around AES for block cipher needs, with older ciphers like 3DES being phased out. Block ciphers like AES provide the core primitives for many encryption protocols, but they must be combined with appropriate modes of operation to securely handle arbitrary message lengths and patterns.

2.2 Stream Ciphers

Stream ciphers generate a pseudorandom keystream that is XORed with plaintext bytes to produce ciphertext. They are useful for applications requiring continuous encryption of streaming data or where data arrives in unpredictable lengths. A historically prominent stream cipher is RC4 [33], designed in 1987, which became widely used (e.g., in early SSL/TLS and WEP for Wi-Fi). RC4 is simple and fast in software, but it has well-known weaknesses: biases in its output keystream (especially in the initial bytes) lead to practical attacks when used in protocols without careful mitigation. Over the past decade, **RC4 was found to be insecure** in contexts like TLS (e.g., the RC4 bias attacks that recover plaintext from ciphertext). This led to RC4 being formally prohibited in TLS 1.2 and later; researchers and standards bodies have **deprecated RC4** entirely.

In response to RC4's shortcomings, modern designs favour stream ciphers with provable security properties. One such algorithm is **ChaCha20 [3]**, a 20-round stream cipher designed by Daniel J. Bernstein. ChaCha20 (and its variant XChaCha20) is an improved variant of the earlier Salsa20 cipher. It takes a 256-bit key and 96-bit nonce to generate a keystream with excellent statistical properties and no known attacks better than brute force. **ChaCha20 has become a widely adopted modern stream cipher**, especially in TLS and secure messaging. A notable advantage of ChaCha20 is its performance on systems without AES hardware acceleration: ChaCha20 is designed to leverage common CPU vector instructions and is extremely fast in software. Google and others introduced ChaCha20-Poly1305 cipher suites for TLS to improve performance on mobile devices, where AES (if not hardware-accelerated) can be slow [6][29]. In fact, ChaCha20 with the Poly1305 authenticator can be **three times faster on mobile devices** than AES-GCM, according to measurements on Android devices. This makes ChaCha20 highly attractive for smartphone applications and VPNs. ChaCha20 is also immune to the specific TLS attacks (BEAST, Lucky13, etc.) that affected older cipher configurations [34]. Stream ciphers like ChaCha20 are typically used in an *Authenticated Encryption* construction (e.g., ChaCha20-Poly1305) to provide both confidentiality and integrity in one step, similar to AES-GCM. In summary, stream ciphers remain important for high-speed encryption, especially on constrained devices, but algorithms like ChaCha20 (with robust security and performance) have displaced legacy ciphers like RC4.

2.3 Modes of operations

Block ciphers such as AES operate on fixed-size blocks (typically 128 bits), which necessitates the use of modes of operation to securely encrypt longer messages and to provide additional features such as authentication. Different modes offer varying security guarantees and performance characteristics. The simplest among them is the Electronic Codebook (ECB) mode, where each block is encrypted independently. However, ECB is not semantically secure—identical plaintext blocks produce identical ciphertext blocks, leading to visible data patterns (such as outlines in images). Consequently, ECB is rarely used in practice except in niche cases where data patterns are irrelevant. Cipher Block Chaining (CBC) mode improves upon ECB by XORing each plaintext block with the previous ciphertext block before encryption, using an initialization vector (IV) for the first block. CBC effectively hides plaintext patterns and was historically used in protocols like TLS 1.0/1.1 and IPsec. However, CBC alone does not provide integrity and is vulnerable to padding oracle attacks if improperly implemented. It can still be secure when used with proper padding and combined with an integrity check such as a Message Authentication Code (MAC), though modern systems often avoid CBC in favour of authenticated encryption.

Counter (CTR) mode is another widely used approach that turns a block cipher into a stream cipher by encrypting successive values of a counter and XORing the result with plaintext blocks. CTR offers benefits such as parallelizability and resistance to error propagation, making it efficient and practical. However, it lacks built-in integrity protection and is critically vulnerable if counters or IVs are reused, as this can expose plaintext via keystream reuse. To mitigate this, CTR is often paired with an external MAC like HMAC or PMAC [27][28]. Modern cryptographic systems typically prefer authenticated encryption with associated data (AEAD) modes, which combine confidentiality and integrity in a single pass. One such mode is Galois/Counter Mode (GCM), which builds on CTR mode and adds integrity using Galois field multiplication. AES-GCM is widely adopted in protocols such as TLS 1.2/1.3 and IPsec due to its high performance, especially when hardware acceleration for AES is available. It efficiently provides both encryption and authentication without needing a separate MAC. On platforms lacking AES acceleration, alternatives like ChaCha20-Poly1305 may offer better performance [6].

Other modes like Cipher Feedback (CFB) and Output Feedback (OFB) are older methods that convert block ciphers into stream ciphers, while XTS mode is used for encrypting disk sectors with tweakable encryption to prevent data relocation attacks. Overall, modes of operation are critical to deploying block ciphers securely. Current best practices recommend avoiding ECB entirely, using CBC only with great caution and integrity checks, and preferring AEAD modes such as AES-GCM or ChaCha20-Poly1305 for robust and efficient security in modern applications [3][6][35]. Other legacy modes like CFB and OFB are rarely used today. **XTS mode** is deployed in disk encryption software such as BitLocker and VeraCrypt [36].

2.4 Security Analysis and Applications of Symmetric Ciphers

Symmetric algorithms are valued for their **speed and high throughput**, making them suitable for encrypting large data volumes, VPN tunnels, disk encryption, etc. A 128-bit key cipher like AES-128 is estimated to offer ~128-bit security, which is currently considered safe against any brute-force attack [27][30]. Thus, attacks on symmetric ciphers typically focus on finding structural weaknesses (cryptanalysis) rather than brute force. The best-known classical attacks are **linear** and **differential cryptanalysis**, which exploit statistical biases in cipher rounds. AES was explicitly designed to withstand these: for instance, its S-box's nonlinearity ensures minimal linear correlations and differential probabilities. As a result, no shortcut attacks have been found on full

AES – cryptanalysis has only broken reduced-round versions or weakened ciphers with related structures.

Some earlier symmetric algorithms did succumb to cryptanalysis. The original DES (56-bit key) was vulnerable to brute force (2^{56} possibilities) – indeed the EFF's "DES cracker" machine demonstrated a DES key could be found in days by 1998. DES was also attacked by differential cryptanalysis (Biham and Shamir in the 1990s), although DES's designers had unbeknownst to the public tweaked it to resist those attacks to an extent. The small 56-bit key of DES was the bigger issue, and Triple-DES extended its life for a while by effectively using 112-bit keys. As noted, however, 3DES and other 64-bit block ciphers now fall to large-scale "birthday" collision attacks if too much data is encrypted; this is why NIST has retired 3DES after 2023. **Blowfish**, despite having up to 448-bit keys, has a 64-bit block and thus shares the large-data vulnerability (Sweet32), so it should not be used for very large file encryption without re-keying. Modern block ciphers use 128-bit blocks as a minimum to mitigate this.

In practical deployments, symmetric ciphers are usually combined with public-key algorithms: for example, in SSL/TLS or PGP, a symmetric key (for AES, etc.) is exchanged or wrapped using asymmetric cryptography, and then the symmetric cipher handles the bulk data encryption due to its efficiency. Symmetric encryption is also the core of **secure storage** (filesystems encryption, database encryption) and **communications** (VPNs, Wi-Fi WPA3 uses AES-CCM or GCMP, etc.). A specific application class is **authenticated secure messaging**: protocols like Signal (used in WhatsApp, Signal Messenger, etc.) rely on symmetric ciphers for message privacy (often AES or XChaCha20 for the message encryption) combined with message authentication codes (or built-in AEAD tags) to ensure integrity. Symmetric ciphers are further used in constructing cryptographic primitives like pseudorandom generators and hash functions (e.g., SHA-3's Keccak is built from a symmetric permutation). Overall, symmetric key cryptography provides the *confidentiality layer* in most systems, and its security hinges on using strong algorithms (AES, ChaCha20) in correct modes with proper key management.

Applications: Virtually every secure application uses symmetric cryptography. For example, in the **Signal/WhatsApp protocol**, once the two parties have established a shared secret via public-key key exchange, they derive symmetric session keys which are then used with AES-256 (in CBC or CTR with HMAC, or in newer implementations, XChaCha20-Poly1305) to encrypt each message. Similarly, **VPNs** use symmetric ciphers for encrypting data packets: the WireGuard VPN protocol uses ChaCha20 for encryption (with Poly1305 for authentication) as a core design choice for high performance and security on all platforms, whereas IPsec VPNs commonly use AES-GCM for the ESP payload encryption. In the realm of **storage**, tools like VeraCrypt, BitLocker, and LUKS rely on AES (often in XTS mode) to encrypt disk volumes. **IoT devices** and embedded systems, which often have constrained CPUs and no hardware AES support, also utilize symmetric encryption (sometimes favoring lighter algorithms or smaller key sizes for speed). NIST has recently standardized the Ascon family [16] as a lightweight authenticated cipher for IoT, due to its efficiency on small microcontrollers. In summary, symmetric cryptography is ubiquitous, from securing web traffic to protecting data at rest, and continues to evolve (e.g., the rise of ChaCha20 in mobile and certain TLS configurations for speed). The next sections contrast this with public-key cryptography, which addresses different security needs like key exchange and digital signatures.

3. Public Key Cryptography

Public-key (asymmetric) cryptography uses **pairs of keys**: a public key (which can be openly shared) and a private key (kept secret by the owner). This enables powerful capabilities such as

letting anyone encrypt a message or verify a signature using the public key, but only the holder of the corresponding private key can decrypt or sign, respectively. Since its inception in the late 1970s, public-key cryptography has become the backbone of secure key exchange, digital signatures for authenticity, and public key infrastructure (PKI) for identity and certificates. Asymmetric algorithms rely on one-way mathematical problems (trapdoor functions) that are easy to compute in one direction but believed infeasible to invert without the private key (e.g., factoring large integers or computing discrete logarithms). We survey the major public-key algorithms: the classic **RSA [9]** and related schemes (Diffie–Hellman, DSA, ElGamal [27]) based on modular arithmetic, and the more modern **Elliptic Curve Cryptography (ECC)** algorithms like ECDSA and EdDSA based on elliptic curve groups [25] [27]. We also discuss their use in protocols (SSL/TLS, SSH, PGP, cryptocurrency systems, etc.) and compare their security and performance.

3.1 RSA, DSA, and ElGamal (Classical Public-Key Algorithms)

The foundation of practical public-key cryptography began with the Diffie–Hellman (DH) key exchange protocol introduced in 1976. DH revolutionized secure communication by allowing two parties to establish a shared secret over an insecure channel without transmitting the key itself. Importantly, DH is used purely for key agreement, not for direct encryption or digital signatures, and its security relies on the computational hardness of the discrete logarithm problem in a finite group (commonly modulo a large prime). Shortly after, in 1978, Rivest, Shamir, and Adleman introduced RSA—still one of the most widely used public-key algorithms. RSA supports both encryption and digital signatures, with its security based on the difficulty of factoring large semiprime integers. The public key includes a large modulus $n=p\cdot q$ (where p and q are large primes) and a public exponent, typically e=65537, while the private key is derived using Euler's totient function. RSA is extensively used in Public Key Infrastructure (PKI), such as digital certificates in TLS, and in secure email systems like PGP. Today, a 2048-bit RSA key offers around 112 bits of security, while a 3072-bit key is recommended for 128-bit security. RSA offers fast encryption and signature verification due to the small public exponent but suffers from slower decryption and signature generation because of expensive private-key operations.

The Digital Signature Algorithm (DSA), standardized by NIST in the 1990s (FIPS 186), was designed exclusively for digital signatures and not encryption. It shares mathematical roots with the ElGamal signature scheme and is based on the discrete logarithm problem. DSA operates over a subgroup of a finite prime field, with a key consisting of large parameters p, q, and a generator g. A DSA signature consists of two numbers (r,s) and critically depends on using a unique, uniformly random nonce for each message; nonce reuse can catastrophically expose the private key—a flaw that has occurred in real-world implementations. While DSA provides similar security to RSA for comparable key sizes (e.g., 2048-bit DSA $\approx$ 2048-bit RSA), its usage has declined outside government systems. DSA signatures are smaller and faster to generate than RSA, but verification tends to be slower, and DSA lacks encryption or key agreement capabilities, limiting its use in broader cryptographic protocols like TLS.

ElGamal encryption, another significant early asymmetric scheme, is an encryption method derived from the principles of Diffie–Hellman. It encrypts messages by combining a random exponent with the recipient's public key, producing ciphertexts made of two group elements. Its security also stems from the hardness of discrete logarithms, and while it provides semantic security under chosen-plaintext attacks (assuming the Decisional Diffie–Hellman assumption), it is malleable and not secure under adaptive chosen-ciphertext attacks without additional protection. ElGamal is not as widely used as RSA due to its larger ciphertext size and computational demands, but it remains

foundational in modern cryptographic constructions. It is used in hybrid encryption schemes (e.g., in some PGP variants) and forms the basis of many signature and key exchange protocols. For instance, Diffie–Hellman can be seen as an ElGamal encryption of a random key, and DSA is essentially a modified ElGamal signature scheme.

Together, RSA, Diffie–Hellman, DSA, and ElGamal constitute the first generation of public-key cryptographic algorithms based on number-theoretic hardness assumptions, specifically integer factorization and discrete logarithms. These algorithms have remained secure against classical attacks but require very large key sizes—typically 2048 to 3072 bits—to remain resistant to modern cryptanalysis and computational advances. The resulting performance issues, especially in resource-constrained environments, have led to growing adoption of elliptic curve cryptography (ECC), which offers comparable security with significantly smaller key sizes and improved efficiency.

3.2 Elliptic Curve Cryptography: ECDSA and EdDSA

Elliptic Curve Cryptography (ECC) offers strong public-key security with significantly smaller keys than RSA or DSA, thanks to the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP). Unlike classical discrete log settings, ECDLP lacks known sub-exponential attacks, allowing a 256-bit ECC key (e.g., Curve25519 or P-256) to provide security comparable to a 3072-bit RSA key. This key size reduction translates to faster computations, lower bandwidth use, and reduced storage overhead—making ECC ideal for constrained environments like mobile and IoT devices. ECC algorithms include ECDH for key exchange and ECDSA for digital signatures. ECDH enables two parties to derive a shared secret using elliptic curve key pairs, while ECDSA produces compact signatures (e.g., 64 bytes on 256-bit curves) and is widely used in TLS certificates, Bitcoin transactions, and secure communications. Compared to RSA, ECC provides similar or better security with improved signing speed and smaller keys, though signature verification may be slightly slower.

A modern ECC signature scheme is EdDSA, notably the Ed25519 [25] [27] variant, which enhances performance and security by avoiding per-message randomness and using deterministic nonces. EdDSA is now standard in OpenSSH, Tor, DNSSEC, and secure messaging protocols due to its simplicity, efficiency, and resistance to misuse. It produces 64-byte signatures and is designed for ease of secure implementation. ECC has become the preferred choice for public-key cryptography in modern systems. TLS 1.3 exclusively uses ECC-based ephemeral key exchange (e.g., X25519, P-256), and protocols increasingly favor ECDSA or EdDSA over RSA for digital signatures. While ECC requires careful implementation to prevent side-channel attacks, its advantages in speed, size, and security have made it the state-of-the-art for public-key cryptography in the pre–post-quantum era.

3.3 Key Exchange Protocols: Diffie–Hellman and ECDH

Secure key exchange between parties without prior contact is a central challenge in cryptography, elegantly solved by the **Diffie–Hellman (DH)** protocol. In DH, two parties exchange values derived from private secrets and a common group base, allowing them to compute a shared secret over an insecure channel. The security of DH is based on the difficulty of the discrete logarithm problem. Although an eavesdropper can observe the exchanged public values, recovering the secret keys is computationally infeasible with proper parameters. Originally proposed in 1976, DH was the first practical public-key algorithm and remains fundamental in secure communication protocols such as

TLS**,** IPsec, and SSH, especially in its ephemeral form which offers forward secrecy (ensuring past communications remain secure even if long-term keys are compromised).

The elliptic curve variant, Elliptic Curve Diffie–Hellman (ECDH), follows the same principle but operates on elliptic curve groups. It enables key exchange by multiplying private scalars with a base point on the curve. ECDH provides equivalent security with much smaller key sizes and faster computations than traditional DH. For example, a 256-bit ECDH key (e.g., Curve25519) matches the security of a 3072-bit DH key, yet is more efficient in terms of bandwidth and processing. Consequently, TLS 1.3, modern VPNs (e.g., WireGuard), and secure messaging protocols (e.g., Signal's X3DH) all rely on ECDH for their initial key establishment.

However, DH and ECDH [1][6][26] alone do not authenticate the parties involved, leaving them vulnerable to man-in-the-middle attacks. Therefore, real-world implementations must pair DH/ECDH with authentication mechanisms, such as digital signatures or trusted public key certificates. When authenticated, DH-based exchanges not only establish confidentiality but also ensure forward secrecy—critical for long-term data protection. Variants like ElGamal key exchange and multi-stage DH protocols (as in Signal) build upon the core DH concept to enable secure and flexible key agreements. In summary, DH and its elliptic curve counterparts remain cornerstones of secure key exchange, enabling encrypted communication across untrusted networks in a scalable and efficient manner.

3.4 Applications in PKI, SSL/TLS, and Digital Signatures

Public-key cryptography underpins modern digital security, with its primary applications in secure key exchange, digital identity verification, and data integrity. One of its most visible uses is in **Public Key Infrastructure (PKI)**, which secures the web via **X.509 certificates**. These certificates bind a public key to an entity's identity and are signed by trusted **Certificate Authorities (CAs)** using algorithms like RSA or ECDSA. When a user connects to a secure website, the server presents such a certificate; the browser verifies its signature using the CA's public key, ensuring the server's authenticity. After identity verification, **key exchange** occurs—typically via **ephemeral ECDH**—to derive a symmetric session key for encryption.

In **SSL/TLS protocols**, public-key cryptography plays multiple roles: the server's certificate is signed by a CA, and key exchange is performed using ephemeral **Diffie–Hellman** or **ECDH**. While older versions (like TLS 1.2) sometimes used RSA to encrypt session keys directly, **TLS 1.3** streamlines the process: it relies exclusively on ECDH (e.g., X25519, P-256) for key exchange and uses digital signatures only for authenticating the handshake, providing both **forward secrecy** and **authenticated key agreement**. Beyond TLS, **digital signatures** are widely used to verify software integrity (code signing), secure firmware boot, sign cryptocurrency transactions (e.g., ECDSA in Bitcoin), and authenticate users in secure messaging and email (e.g., PGP, GPG). In all these cases, asymmetric signatures ensure **non-repudiation** and **data integrity**, as only the private key holder can produce a valid signature verifiable with the corresponding public key.

PKI relies primarily on **RSA and ECC**. RSA (2048 or 3072 bits) remains common for certificates, but ECC (P-256, P-384) is increasingly favoured due to smaller key sizes and better efficiency. Algorithms like **DSA** are largely deprecated in public PKI, while **Ed25519**—a fast and secure signature scheme—is gaining traction in SSH, secure messaging, and specialized applications, though not yet widely used in X.509 certificates. **Hybrid encryption** is often employed in secure communications (e.g., PGP), where a symmetric key (e.g., for AES encryption) is encrypted with

public-key algorithms like RSA or ElGamal for each recipient. This approach combines the efficiency of symmetric ciphers with the flexibility of public-key encryption for multi-recipient communication.

Other applications include **authentication protocols** (e.g., FIDO2/WebAuthn, where devices sign challenges with private keys) and **blockchains**, where public-key cryptography secures transactions. In systems like Bitcoin and Ethereum, users sign transactions with ECDSA over the secp256k1 curve, and the public keys verify authenticity without exposing private information.

In summary, **public-key cryptography** is central to establishing trust and enabling secure communications across the internet. RSA and ECC (especially ECDSA/ECDH) remain dominant in 2025, with ECC becoming the preferred standard due to its compactness and performance advantages. From websites and VPNs to firmware and financial transactions, asymmetric cryptography is the backbone of secure infrastructure. Looking ahead, the field is preparing to transition to **post-quantum algorithms**, as both RSA and ECC would be vulnerable to sufficiently powerful quantum attacks—a topic explored in subsequent sections.

4. Cryptanalysis

No cryptographic algorithm is immune to attack, and the field of **cryptanalysis** is devoted to finding weaknesses in cryptographic primitives. Over the years, cryptanalysts have developed a variety of techniques to break or weaken ciphers. In the early 1990s, Eli Biham and Adi Shamir unveiled *differential cryptanalysis*, a powerful chosen-plaintext attack that exploits patterns in how differences in inputs can affect differences in outputs through the cipher. Originally a classified technique, differential cryptanalysis was shown publicly to be effective against many cipher designs and was famously applied to a reduced-round variant of DES. Soon after, in 1993 Mitsuru Matsui introduced *linear cryptanalysis*, an attack that uses linear approximations to describe a cipher's behaviour and requires a large number of known plaintexts to derive key bits. Matsui's method was used to analytically break DES (again in a reduced-round scenario), and its description marked the first successful cryptanalysis of DES purely by academic research (he was able to recover a DES key with about 2^{43} known plaintexts). These techniques forced cipher designers to adjust parameters (e.g., increasing the number of rounds) and to analyze new algorithms against such advanced methods.

Beyond block ciphers, cryptanalysis has targeted other components too. The hardness of RSA relies on the difficulty of prime factorization; as factorization algorithms improved (for example, the Number Field Sieve), RSA keys needed to grow in size. By 2010 researchers factored a 768-bit RSA modulus, a feat that underscored the recommendation to use RSA keys of 2048 bits or more for long-term security. For discrete-log-based systems (like DSA or ECC), improved algorithms (pollard rho, index calculus, etc.) similarly influence required key sizes. Another important class of attacks are **side-channel attacks**, which bypass purely mathematical weaknesses by exploiting physical implementations. In 1996, Paul Kocher demonstrated that the time taken to perform private key operations could leak information about the key – these timing attacks were able to extract RSA and Diffie–Hellman keys from software implementations by carefully measuring operation latency. Subsequent research showed that power consumption (power analysis) and other side-channel emissions from cryptographic devices could be analyzed to recover secrets, forcing developers to implement constant-time algorithms and other countermeasures in cryptographic libraries and hardware. Side-channel attacks remind us that even a theoretically strong algorithm can be undermined by a clever attack on its real-world usage.

Cryptanalysis also extends to **cryptographic protocols** and systems. For example, weaknesses in the way random numbers are generated have led to catastrophic breaks (as seen in some public-key implementations where poor randomness yielded predictable keys). Protocol attacks, like man-in-the-middle or downgrade attacks, target how cryptographic algorithms are used in context. A historical example of protocol-level cryptanalysis was the work done at Bletchley Park during World War II, where allied cryptanalysts (including Alan Turing) broke the Enigma cipher by exploiting procedural flaws and repetitive aspects of the German encryption protocols – underscoring Kerckhoffs's principle that the *security of a cipher must depend only on the key, not on the obscurity of the algorithm*. Through continuous scrutiny by cryptanalysts, the community gains confidence in algorithms that withstand years of attack. When weaknesses are found – as with DES, MD5, or SHA-1 – those algorithms are phased out and replaced by newer, stronger designs. This evolutionary pressure is essential to maintain privacy and security in the face of evolving threats.

5. Performance and Scalability

The choice of cryptographic algorithms in practice is often dictated not just by security but by performance, resource constraints, and scalability (supporting many users or high data throughput). Here we compare symmetric vs asymmetric algorithms in terms of speed, key size, and operational cost, and examine how these factors play out in various environments (servers, mobile, embedded/IoT). We also provide a comparative table of algorithm characteristics, especially relating key sizes to security levels, which is crucial for long-term security planning.

Key Sizes vs. Security Levels: Cryptographers measure security in bits – e.g., "128-bit security" means an attacker would need about 2^{128} operations to break the scheme (equivalent to brute-forcing a 128-bit key). Symmetric ciphers define the baseline: AES-128 is considered 128-bit secure. Asymmetric schemes require much larger key sizes to reach the same security because known attacks are faster than brute force. The Table1 below (based on NIST and NSA guidelines) shows the rough equivalences:

Security Level (bits)	**Symmetric Cipher (key size)**	**RSA / DH (modulus size)**	**ECC (field size)**
~80 (deprecated)	2-key Triple DES (112-bit key)	1024-bit RSA/DH	~160-bit curve (e.g., secp160)
112	3-key Triple DES (168-bit key, ≈112-bit security)	2048-bit RSA/DH	224-bit curve (e.g., secp224)
128	AES-128 (128-bit key)	3072-bit RSA/DH	256-bit curve (secp256r1, Curve25519)
192	AES-192 (192-bit key)	7680-bit RSA/DH	384-bit curve (secp384r1)
256	AES-256 (256-bit key)	15360-bit RSA/DH	521-bit curve (secp521r1)

Table 1:: Comparable strengths for symmetric and asymmetric algorithms. Approximate recommendations from NIST SP 800-57 and related sources.

From the table, for example, a 256-bit ECC key (like secp256r1) provides about the same security as a 3072-bit RSA key. This dramatic difference is why ECC is preferred for higher security with less computational load. It is evident that asymmetric keys must grow much faster to maintain

security. RSA-2048 is sufficient for now (~112-bit security), but to reach 128-bit security one must go to RSA-3072. In constrained environments, RSA-3072 or RSA-4096 can be problematic (in terms of CPU and memory), whereas ECC-256 or ECC-384 is manageable and offers equal or better security.

- **Hardware vs. Software Efficiency**: Symmetric ciphers (AES, ChaCha20) are generally orders of magnitude faster than public-key operations. AES can often be executed in a few cycles per byte with modern CPU instructions. Asymmetric ops like RSA or ECDSA involve big integer math (modular exponentiation or scalar multiplication) which is much slower. For perspective, on a typical CPU, AES throughput (with AES-NI) might be 5–10 GB/s, whereas RSA-2048 can do only a few thousand operations per second. ECC is faster than RSA at equivalent security: for example, a 256-bit ECDSA signing might take on the order of microseconds to a millisecond, whereas a 3072-bit RSA signing could take a few milliseconds or more. On embedded devices, RSA's disparity is more pronounced – RSA key generation in particular is extremely slow (finding random primes). A study noted RSA key generation can be 100–1000 times slower than ECC key generation for embedded systems. Also, RSA private operations scale roughly with the cube of the modulus length (if using basic algorithms), while ECC scales more gently with key size.

  Many modern CPUs have hardware accelerators for AES (AES-NI on Intel, ARMv8 crypto extensions) which significantly boosts symmetric encryption speed (4×–8× improvement). Likewise, some chips accelerate SHA-256, GCM multiplication, etc. Asymmetric operations can also be accelerated (some CPUs have big-integer arithmetic units or there are dedicated cryptographic co-processors in smart cards and HSMs). Nevertheless, the difference in throughput remains: symmetric encryption can handle data streams at network line rates or storage speeds, whereas asymmetric crypto is typically used sparingly (e.g., one RSA op to set up a session, then stream with symmetric).

- **Mobile and Embedded Constraints:** On smartphones, not all devices had AES hardware until recent years. This is where ChaCha20 shined – it's a software-efficient cipher not needing specialized hardware and was shown to significantly outperform AES-256 on mobile CPUs without AES-NI. Many mobile TLS stacks prefer ChaCha20-Poly1305 when AES hardware support is absent, to reduce latency and battery usage. Embedded IoT devices often run on low-power microcontrollers (no hardware crypto, limited MHz). On such devices, performing a 2048-bit RSA handshake could be very slow (hundreds of milliseconds or more), whereas an ECC handshake (Curve25519) might complete in a few milliseconds, making ECC much more suitable for IoT authentication. Additionally, memory constraints make large RSA keys problematic: an RSA-2048 key pair might require several kilobytes of storage and buffers, while a Curve25519 key is 32 bytes and signatures 64 bytes, much more compact. As Microchip's whitepaper noted, at 128-bit security, RSA public keys and signatures are about 6× larger than ECC's, and private keys 12× larger, which matters for storing keys securely and transmitting them (e.g., certificates). Therefore, for protocols like Zigbee, Bluetooth, and others targeting constrained devices, ECC is the preferred choice.
- **Throughput vs. Latency:** Symmetric ciphers have negligible latency overhead – encrypting a 16-byte block with AES might be <1 microsecond on modern processors. Asymmetric ops have significant latency: a single RSA-2048 sign might take say 1–5 ms on a server (much more on a microcontroller). This is why protocols minimize the number of public-key operations per session. In TLS, typically only two to three asymmetric ops occur (server signature, optionally client signature, and key exchange), everything else is symmetric. Another example: PGP email encryption uses one RSA/ECC operation per recipient to encrypt the session key, but then uses

fast symmetric crypto for the bulk message. In blockchain, verifying an ECDSA signature (which every node does for every transaction) is optimized with algorithms and sometimes hardware because tens of thousands of signature verifies per second may be needed system-wide – yet ECC is still manageable (verification is faster than signing and libraries like libsecp256k1 are highly optimized).

- **Parallelism and Scalability**: Symmetric crypto can take advantage of parallelism (e.g., encrypting multiple blocks in parallel in CTR or GCM mode, using multiple cores or SIMD instructions). Asymmetric crypto typically involves large serial computations that are harder to parallelize per operation (though you can of course do many separate RSA/ECDSA ops in parallel if you have many cores). There is research on using GPUs for RSA/ECC at high throughput (e.g., TLS terminators), and FPGA or ASIC acceleration for these tasks.

Cryptographic agility and algorithm choices: Performance considerations sometimes drive algorithm choices. For instance, some organizations prefer Ed25519 (EdDSA) over RSA for signing because it's much faster at high security and keys are small – important for things like DNSSEC or Certificate Transparency logs that need to handle huge volumes of signatures. Similarly, some VPNs (WireGuard) chose ChaCha20 over AES for simplicity and performance across platforms. On servers with dedicated AES instructions, AES-GCM might achieve ~10 Gbit/s per core, which is sufficient for most use, but on devices without, ChaCha20 is more consistent. Comparative Table of Algorithms is presented below (Table 2) with summary of key characteristics of some representative algorithms:

Algorithm	**Type**	**Key Sizes (bits)**	**Security (bits)**	**Notable Features & Usage**
AES	Symmetric Block Cipher	128, 192, 256 key; 128-bit block	128, 192, 256	**Standard for encryption** (fast in hardware/software); AES-256 used for top security. Resistant to known cryptanalysis. Widely used in TLS, IPsec, disk encryption.
ChaCha20-Poly1305	Symmetric Stream AEAD	256-bit key; 96-bit nonce	~256	**Modern stream cipher with MAC**; very fast in software (3× faster than AES-GCM on some mobiles). Used in TLS 1.3 as an alternative cipher, in WireGuard VPN, etc.
3DES (TDEA)	Symmetric Block Cipher	112-bit (effective, 168-bit key)	~112	Legacy triple-DES. 64-bit blocks (Sweet32 vulnerable). **Deprecated** (disallowed after 2023) due to slow speed and small block size.

RSA	Public-key (IFP)	2048, 3072, 4096 ...	~112 (@2048) , ~128 (@3072)	**Dominant legacy public-key** for encryption & signatures. Security based on factoring. Large keys (2048+ bits). Fast public ops, slow private ops. Still widely used in certificates/PKI, but slowly being replaced by ECC. Vulnerable to quantum (Shor's algorithm).
Diffie–Hellman	Key Exchange (DLP)	2048, 3072 mod p (group size)	~112, ~128	**Classic key agreement**. Often used with ephemeral keys (DHE). Large p for security. Now mostly replaced by ECDH in new systems for efficiency.
DSA	Signature (DLP)	2048-bit p, 224/256-bit q	~112 (@2048/224)	**Digital Signature Standard** (FIPS 186). Similar to ElGamal signature. Smaller signatures than RSA. Requires good randomness. Rare in modern use outside government compatibility. Superseded by ECDSA.
ECDH (Curve25519)	Key Exchange (ECDLP)	256-bit field (curve)	~128	**Efficient ECC Diffie–Hellman**. Curve25519 (Montgomery) offers speed and security (no known weakness). Used in TLS 1.3 (X25519) and many protocols (Signal, WireGuard). Very fast even on embedded.
ECDSA (P-256)	Signature (ECDLP)	256-bit curve (secp256r1)	~128	**Elliptic Curve Signature**. Much smaller keys/signatures than RSA. Widely used in TLS certs, Bitcoin, etc. Requires secure random nonce.
Ed25519	Signature (ECDLP)	256-bit curve (twisted Edwards)	~128	**EdDSA (Ed25519)**: Schnorr-based signature. Fast, no bias issues (deterministic nonce). 64-byte signature. Used in SSH, modern applications. Growing popularity for security and simplicity.
ElGamal	Encryption (DLP)	2048+ bit p	~112 (@2048)	**ElGamal encryption** (homomorphic property: multiplicative). Not common standalone, but underlies many systems (e.g., PGP uses ElGamal or RSA for session key encryption). Ciphertext is double the size of plaintext + overhead.

SHA-256	Hash	– (256-bit output)	128 (collision)	Cryptographic hash (not an encryption algorithm, but key in integrity). Mentioned because in performance context, hashing is often alongside encryption. SHA-256 can be hardware accelerated; required in many protocols.

Table 2 :: Comparative Table of Algorithms: Below is a summary of key characteristics of some representative algorithms

(Here IFP = Integer Factorization Problem; DLP = Discrete Log Problem; ECDLP = Elliptic Curve Discrete Log Problem.)

From a performance perspective, symmetric algorithms vastly outperform asymmetric for equivalent security. That is why designs use asymmetric crypto sparingly (for initial key exchange or signing a digest) and switch to symmetric for bulk encryption. For example, a secure channel setup might involve a few asymmetric operations (taking a few milliseconds total), after which gigabytes of data can flow encrypted by AES at negligible CPU cost. Hardware support has narrowed some gaps. With AES-NI, AES encryption on even a modest CPU can exceed 5 Gbps throughput, making it essentially free relative to typical network speeds. GCM authentication is also hardware-optimized on many platforms. On the other hand, an RSA-2048 sign operation might be ~0.1 ms on a high-end CPU (tens of thousands per second), which is fine for a single SSL handshake but would be expensive if you had to do it for every network packet.

Finally, consider **scalability** in multi-user systems: On a server handling 10,000 TLS handshakes per second, RSA would impose a heavy load (10k RSA decrypts/sec). Some large platforms like Cloudflare switched to ECDSA and ECDH to drastically reduce CPU usage per handshake, thus scaling to more connections with the same hardware. In blockchain, thousands of signatures need verifying per block – ECC makes that feasible, whereas RSA-size signatures and keys would bloat the system and slow down processing. In **embedded/IoT**, limited battery and CPU means heavy asymmetric crypto can shorten device battery life or make operations sluggish. ECC at 256-bit is feasible on small microcontrollers (taking maybe tens of milliseconds for a signature), while RSA-2048 might take seconds on the same device – unacceptable for user experience or real-time constraints. That is why protocols like Thread and Zigbee exclusively use ECC for joining networks, and why protocols like WireGuard opted for an all-ECC approach to be future-proof for IoT use.

In conclusion, performance considerations strongly favour: use symmetric crypto for data, use the "lightest" asymmetric crypto that meets security (hence ECC over RSA in new designs), offload to hardware where possible, and be mindful of key sizes relative to security to avoid undue overhead. We have seen a clear trend: industry moving to ECC for better performance at high security, and standardized security levels aligning with those shown in the table (128-bit being a common target, which means 3072-bit RSA or 256-bit ECC, the latter being far more efficient). Next, we will see how these choices manifest in real-world applications.

6. Application Use-cases

In this section, we consider several real-world application domains and how they employ cryptographic algorithms: secure messaging, VPNs, blockchain/cryptocurrencies, and IoT. Each use case has specific requirements and constraints that influence the choice of ciphers and protocols.

Cryptographic algorithms are foundational to securing a wide range of modern digital systems. This section explores how different real-world domains—secure messaging, VPNs, blockchain, and

IoT—employ cryptographic primitives based on their specific performance and security requirements. In the domain of secure messaging, applications like Signal and WhatsApp utilize the Signal Protocol, which combines asymmetric and symmetric cryptography to ensure end-to-end encryption [1]. Key exchange is performed using the X3DH protocol, based on elliptic-curve Diffie–Hellman over Curve25519, establishing a shared secret between users [2]. Once a secure channel is established, symmetric ciphers like AES-256-CBC or XChaCha20 are used for efficient message encryption, alongside HMAC-SHA256 for authentication [3]. A key innovation is the double ratchet mechanism, which ensures forward and future secrecy by frequently updating keys using symmetric and asymmetric ratchets [4]. This ensures that compromising one message key does not reveal past or future messages. Users verify identities through static public keys (Curve25519) using out-of-band verification methods, providing protection against man-in-the-middle attacks. This hybrid model achieves strong security with high performance, setting a benchmark for secure messaging platforms [5].

For Virtual Private Networks (VPNs), cryptographic performance is critical due to the need to encrypt entire network traffic in real time. Modern VPN protocols like WireGuard exemplify a minimal and efficient cryptographic design, using Curve25519 for key exchange, ChaCha20-Poly1305 for authenticated encryption, and BLAKE2s for hashing [6]. WireGuard avoids the complexity of algorithm negotiation by fixing its algorithm suite, which enhances performance and security while maintaining a small codebase. Its use of AEAD (Authenticated Encryption with Associated Data) modes ensures confidentiality and integrity with minimal overhead, especially on devices lacking AES hardware. In contrast, IPsec, a more established suite often used in enterprise environments, relies on RSA or (EC)DH for key negotiation through the IKE protocol, and commonly uses AES-GCM for packet encryption [7]. AES-GCM is preferred for its speed and built-in authentication, with configurations supporting AES-GCM-128 or AES-GCM-256. Although more flexible, IPsec is relatively complex and less efficient than WireGuard. Nonetheless, both VPN types uphold key cryptographic principles: secure key exchange via public-key cryptography, symmetric encryption for bulk data, and forward secrecy through ephemeral key mechanisms [8].

In blockchain and cryptocurrencies, cryptography ensures both transactional integrity and decentralized trust. Digital signatures are central: Bitcoin and Ethereum use ECDSA over the secp256k1 elliptic curve to prove ownership and authorize transactions [9]. A user's public key becomes their identity on the blockchain, and transaction validation depends on the unforgeability of ECDSA. Signature size efficiency and verification speed make ECC the default, as public ledgers must store and validate thousands of signatures with minimal space and computation [10]. Hash functions also play a key role: Bitcoin uses SHA-256 (often double-hashed) for mining and block linking, while Ethereum employs Keccak-256 (a SHA-3 variant) for address generation and mining [11]. Most blockchains do not encrypt data, instead relying on pseudonymity for privacy. However, privacy-centric blockchains like Monero and Zcash incorporate advanced cryptography such as ring signatures and zero-knowledge proofs (e.g., zk-SNARKs) to achieve anonymity and confidentiality [12]. The blockchain setting exemplifies cryptography's power in decentralized environments, emphasizing ECC for compact, verifiable identity and hashing for integrity and consensus mechanisms.

In the Internet of Things (IoT), cryptography faces unique challenges due to device constraints like limited power, memory, and computational resources. Asymmetric cryptography in IoT typically uses ECC (e.g., Curve P-256) for key exchange and device authentication, since ECC offers small key sizes and fast computation [13]. Symmetric encryption is predominantly handled by AES-128 due to its adequate security and widespread hardware support in microcontrollers. When AES

hardware is unavailable, lightweight ciphers such as ChaCha20, PRESENT, or NIST's Ascon (standardized in 2023 for lightweight AEAD) are preferred [14]. Communication protocols like Bluetooth LE and Zigbee already use AES-128-CCM or ECDH for security, with newer systems integrating cryptographic accelerators for ECC and AES directly in hardware. To reduce computational burden, some IoT systems opt for pre-shared keys (PSKs) or symmetric group keys, although these approaches lack scalability. Secure firmware updates—crucial for IoT—employ ECC-based digital signatures (e.g., Ed25519 or ECDSA) to ensure authenticity [15]. Moreover, bandwidth constraints in IoT networks necessitate compact cryptographic constructs, making ECC-based schemes favorable due to their smaller overhead compared to RSA. The adoption of standards like COSE and Ascon further reflects the cryptographic shift toward lightweight and efficient solutions tailored to embedded environments. Overall, cryptography in IoT balances performance and security through tailored use of established primitives, ensuring robustness without overburdening constrained devices [16].

In summary, across diverse application domains, cryptographic implementations follow a common pattern: asymmetric schemes (typically ECC-based) for key establishment or identity, symmetric encryption (like AES or ChaCha20) for efficiency, and secure hashing for integrity. Each domain optimizes these components based on operational needs—secure messaging favours agility and secrecy, VPNs demand high throughput, blockchains prioritize signature verification and data integrity, while IoT solutions emphasize lightweight operations with minimal resource usage. This adaptability illustrates cryptography's foundational yet flexible role in securing modern digital ecosystems [17].

7. Post Quantum Cryptography Context

The looming advent of quantum computers threatens to upend many of the cryptographic systems in use today. In the 1990s, Peter Shor discovered a quantum algorithm that can factor large integers and compute discrete logarithms in polynomial time on a hypothetical quantum computer [18]. Shor's algorithm means that RSA, Diffie–Hellman, and elliptic-curve cryptosystems—all of whose security rests on those two hard problems—would be effectively broken if sufficiently powerful quantum computers are realized. Additionally, Lov Grover developed a quantum search algorithm that accelerates brute-force search, providing a quadratic speed-up for attacking symmetric ciphers or hash functions [19]. In practice, Grover's algorithm means that a quantum computer could brute-force a key space of size N in roughly $\sqrt{N}$ operations, so a 128-bit key would have the effective security of a 64-bit key against a quantum attacker. This quantum threat prompted the field of post-quantum cryptography (PQC), which seeks cryptographic algorithms that can resist quantum attacks. These are often based on mathematical problems believed to be hard for quantum computers, such as lattice problems, error-correcting codes, multivariate polynomial equations, and hash-based constructions [20].

Beginning in 2016, NIST initiated an open competition to develop and standardize quantum-resistant cryptography. After multiple rounds of evaluation, NIST announced the first group of winning algorithms in July 2022 [21]. Notably, all of the initial selections for public-key encryption and key establishment are lattice-based: the algorithm CRYSTALS-Kyber was chosen for general-purpose encryption/KEM (Key Encapsulation Mechanism). For digital signatures, NIST selected CRYSTALS-Dilithium (lattice-based) as the primary algorithm, along with FALCON (lattice-based) and SPHINCS+ (hash-based) as alternatives for specific use-cases [22]. These algorithms are based on problems like structured lattice challenges and hash-based combinatorial structures that even quantum computers should struggle to solve, according to current knowledge. A few other candidates (such as the code-based Classic McEliece) are being considered in ongoing evaluation

rounds, reflecting a desire for diversity in defense in case any one problem class is weakened by future advances [23].

Deploying post-quantum cryptography will be a long and complex process. Cryptographic standards and protocols must be updated, software and hardware implementations optimized, and interoperability ensured—all while maintaining security against both quantum and classical attackers [24]. Moreover, care is needed to avoid introducing new vulnerabilities; for instance, some PQC algorithms have much larger key sizes or ciphertext sizes, which can pose practical challenges [25]. Nevertheless, the transition to quantum-safe algorithms is underway. As one observer noted in Communications of the ACM, the road to post-quantum cryptography will require concerted effort from academia, industry, and government to prepare systems before large-scale quantum computers arrive [26]. This proactive migration is vital to ensure that data encrypted today remains secure for years to come, in the era of quantum computing.

8. Conclusion

Cryptography has come a long way from its early roots – evolving from simple ciphers to a sophisticated science underpinning the security of the modern Internet. We have robust symmetric ciphers (AES and others) to protect data at rest and in transit, and powerful public-key systems (RSA, ECC, DH) enabling secure key exchange and digital signatures that form the backbone of digital trust. At the same time, the continual efforts of cryptanalysts have kept the field dynamic: any weakness discovered in an algorithm leads to improved designs and standards (for example, the retirement of SHA-1 in favour of SHA-2/SHA-3, or the replacement of DES with AES). As we stand on the threshold of the quantum era, the cryptographic community is proactively developing and standardizing new tools to ensure security against quantum attacks. The coming years will see a gradual but critical migration to these post-quantum algorithms, alongside the cryptographic best practices already in place. Cryptography is an arms race between designers and attackers – but with rigorous analysis, peer-reviewed research, and prudent standards, cryptography will continue to enable private and authenticated communication in the face of whatever computational advancements lie ahead.

9. References

[1] M. Marlinspike and T. Perrin, “The Signal Protocol,” Open Whisper Systems Technical Report, (2016).
[2] T. Perrin and M. Marlinspike, “X3DH: Extended Triple Diffie–Hellman Key Agreement,” Signal Technical Specification, (2016).
[3] A. Langley, M. Hamburg and S. Turner, “ChaCha20 and Poly1305 for IETF Protocols,” RFC 8439, IETF, (2018).
[4] T. Perrin, “The Double Ratchet Algorithm,” Open Whisper Systems Specification, (2016).
[5] A. Green and R. Crews, “End-to-End Encryption in WhatsApp,” WhatsApp Security Whitepaper, Facebook Inc., (2018).
[6] J. Donenfeld, “WireGuard: Next Generation Kernel Network Tunnel,” WireGuard Whitepaper, (2017).
[7] S. Kent and K. Seo, “Security Architecture for the Internet Protocol,” RFC 4301, IETF, (2005).
[8] D. Harkins and D. Carrel, “The Internet Key Exchange (IKE),” RFC 2409, IETF, (1998).
[9] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” (2008), https://bitcoin.org/bitcoin.pdf.
[10] G. Maxwell, A. Poelstra and P. Wuille, “libsecp256k1: Optimized ECC Library for Bitcoin,” Bitcoin Core Documentation, (2015).
[11] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Ethereum Yellow Paper, (2014).

[12] M. Noether, "Ring Signature Confidential Transactions for Monero," Monero Research Lab Technical Report, (2015).
[13] M. Scott, "Implementing Cryptographic Pairings," Proceedings of the 2010 International Conference on Cryptology in Africa, Springer, Berlin, pp 177–195 (2010).
[14] National Institute of Standards and Technology, "Lightweight Cryptography Finalists: Ascon," NIST Announcement, https://csrc.nist.gov, visited on 12/6/2025.
[15] ARM Ltd., "Mbed OS: Security Features and Firmware Verification," Technical Reference Manual, ARM, Cambridge (2020).
[16] Bluetooth SIG, "Bluetooth Core Specification v5.3," (2021), https://www.bluetooth.com/specifications.
[17] D.J. Bernstein, "Curve25519: New Diffie–Hellman Speed Records," Lecture Notes in Computer Science, vol. 3958, Springer, Berlin, pp 207–228 (2006).
[18] P.W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, IEEE Press, Santa Fe, NM, pp 124–134 (1994).
[19] [L.K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, Philadelphia, PA, pp 212–219 (1996).
[20] D.J. Bernstein, J. Buchmann and E. Dahmen, *Post-Quantum Cryptography*, Springer, Berlin (2009).
[21] NIST, "Post-Quantum Cryptography Standardization," *NIST PQC Project Overview*, https://csrc.nist.gov/Projects/post-quantum-cryptography.
[22] NIST, "NIST Selects Quantum-Resistant Encryption Algorithms," *NIST News Release*, July 2022, https://www.nist.gov/news-events/news/2022/07/nist-announces-first-post-quantum-cryptography-standards.
[23] R. Misoczki, J. Tillich, N. Sendrier and P.S.L.M. Barreto, "Classic McEliece: conservative code-based cryptography," *PQCrypto 2013*, Springer, pp 23–45 (2013).
[24] E. Alkim, L. Ducas, T. Pöppelmann and P. Schwabe, "Post-quantum key exchange—a new hope," *Proceedings of the 25th USENIX Security Symposium*, Austin, TX, pp 327–343 (2016).
[25] D. Moody, R. Peralta and D. Smith-Tone, "Challenges in migrating cryptographic infrastructure to post-quantum algorithms," *IEEE Security & Privacy*, vol. 17, no. 4, pp 48–54 (2019).
[26] S. Garfinkel, "The Road to Post-Quantum Cryptography," *Communications of the ACM*, vol. 64, no. 3, pp 29–31 (2021).
[27] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton (1996).
[28] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer, Berlin (2010).
[29] S. Gueron, "Intel Advanced Encryption Standard (AES) Instructions Set," *Intel White Paper*, (2010), https://www.intel.com/content/www/us/en/docs.
[30] V. Rijmen and J. Daemen, "AES Proposal: Rijndael," *NIST AES Competition Submission*, (1999).
[31] S. Josefsson and B. Kaliski, "PKCS #1: RSA Cryptography Specifications Version 2.2," *RFC 8017*, IETF, (2016).
[32] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, New York (1996).
[33] M. AlFardan, D. Bernstein, K. Paterson, B. Poettering and J. Schuldt, "On the Security of RC4 in TLS," *USENIX Security Symposium*, pp 305–320 (2013).
[34] A. Langley, "Prohibiting RC4 Cipher Suites," *Google Security Blog*, (2016), https://security.googleblog.com/2016/09.
[35] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," *NIST Special Publication 800-38D*, (2007).
[36] [36] D. A. Cooper, S. Santesson, S. Farrell, R. Housley and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," *RFC 5280*, IETF, (2008).

10. BIOGRAPHIES

Ami M. Shah, is the faculty of Computer Engineering at Dharmsinh Desai University since July 2007. She completed her B.E. Computer Engineering from Government Engineering College Modasa in 2005 and M.Tech Computer Engineering from Dharmsinh Desai University in 2016. She teaches Artificial Intelligence, Database Management Systems, Network and Information Security at UG Level. She also teaches Soft Computing at Post Graduate Level. She has guided various UG Level projects. Her research area includes Image Processing, Soft Computing, and Cryptography.

Ashishkumar Gor, is the faculty of Computer Engineering at Dharmsinh Desai University since July 2010. He completed his B.E. from Gujarat University in 2009. He has done his M.Tech from Dharmsinh Desai University in 2014. His areas of research include Image Processing, Deep Learning, Mathematical Optimization, Cryptography and Network Security. He teaches algorithms, compilers, cryptography, and Mathematical foundation of Computer Science courses in UG and PG Level. He is pursuing his Ph.D. from the same university and working on Self-Supervised approaches for solving Image Restoration problems.

PAPER CITATION: Shah, M.A. , Gor, A .(2025) :: ***Comprehensive Survey of Symmetric and Public-Key Cryptographic Algorithms: Foundations, Attacks, and Applications. International Journal of Informative & Futuristic Research (IJIFR), Volume - 12, Issue -10, June 2025, Pg. No. 20-39, URPI-IJIFR/V12/E10/005***