Journal homepage: https://itce.vn.ua/en

Information Technologies and Computer Engineering **Vol. 23, No. 1. 2026**

Article's History: Received: 28.10.2025 Revised: 30.01.2026
Accepted: 26.03.2026 Published: 20.04.2026

ISSN 1999-9941; e-ISSN 2078-6387
UDC 004.056
DOI: 10.31649/vitce/1.2026.09

ChaCha: Development and modification of Salsa20 in modern cryptographic systems

Oleksii Palii*

Postgraduate Student
Vinnytsia National Technical University
21021, 95 Khmelnytske shose Str., Vinnytsia, Ukraine
https://orcid.org/0009-0006-8387-3609

Oleksandr Dudnyk

PhD in Technical Sciences, AssociateProfessor
Vinnytsia National Technical University
21021, 95 Khmelnytske shose Str., Vinnytsia, Ukraine
https://orcid.org/0009-0005-3684-965X

Abstract. The study reviewed the ChaCha20 stream cypher as the successor to the Salsa20 algorithm, emphasising development, technical features, and application in modern cryptosystems. The research relevance is determined by the widespread implementation of ChaCha20 in security protocols (TLS 1.3, VPN, etc.) due to its high performance in software implementations and resistance to cryptanalysis. The study aimed to analyse the evolution of ChaCha from Salsa20, compare it with other cyphers, and summarise the latest achievements in terms of modifications and performance. The study used methods of analysing literary sources and experimental data on the speed and resistance of cyphers. The main results included a highlighted history of ChaCha's creation based on Salsa20 and improving diffusion per round, a detailed description of the algorithm's structure (4×4 state matrix, addition-rotation-XOR operations) and its cryptographic strength (no practical attacks on the full 20-round version). The advantages of ChaCha20 over the Advanced Encryption Standard (AES) in a software environment are demonstrated; in particular, on platforms without AES hardware acceleration, ChaCha20 runs up to 3 times faster with an equivalent level of security. The implementation of ChaCha20-Poly1305 in TLS and WireGuard is considered, as well as the use of XChaCha for extended nonces and the Adiantum algorithm for disk encryption on mobile devices. Modern modifications of ChaCha (e.g., increasing the number of rounds) and their impact on performance and security were analysed. The practical value of the review is determined by a summary of modern experience with ChaCha20, which can be used for the selection of cryptographic algorithms in resource-constrained systems and for further research in the field of stream cyphers

Keywords: stream cipher; Advanced Encryption Standard; TLS 1.3; cryptanalysis; resource-constrained systems

Introduction

Stream cyphers remain a substantial class of symmetric encryption algorithms, especially in situations where high performance at the software level is required. Early implementations, such as RC4, have been officially banned in modern protocols and are considered unreliable in practice due to numerous statistical biases and side-channel attacks (Datadog Security, n.d.). Block cyphers (e.g., AES Advanced Encryption Standard), on the other hand, while reliable, rely heavily on specialised instructions for hardware acceleration. This creates difficulties when using them on devices without such support, such as mobile gadgets, IoT devices, or in resource-constrained environments. In software implementations, block cyphers often demonstrate slower performance and vulnerability to timing attacks due to the use of S-Box tables. The problem of AES vulnerability to side-channel attacks was analysed in

Suggested Citation:
Palii, O., & Dudnyk, O. (2026). ChaCha: Development and modification of Salsa20 in modern cryptographic systems. *Information Technologies and Computer Engineering*, 23(1), 9-21. doi: 10.31649/vitce/1.2026.09

*Corresponding author

detail by B. Gülmezoglu *et al.* (2019), who describe successful attacks using CPU cache observation. In the context of these challenges, research has emphasised new stream cyphers, in particular ChaCha20, a modification of Salsa20 that combines high cryptographic strength with efficiency. ChaCha20 has been successfully implemented in TLS 1.3 (Transport Layer Security), VPN protocols (Virtual Private Network, in particular WireGuard) and security systems as an effective alternative to AES-GCM (Advanced Encryption Standard in Galois/Counter Mode) in conjunction with Poly1305. Its structure avoids the typical problems of previous cyphers and ensures high performance even without hardware acceleration, which makes further study of this algorithm relevant.

The ChaCha20 algorithm continues to attract interest from researchers and developers of cryptographic systems. B. Rashidi (2024) conducted an experimental comparison of the effectiveness of ChaCha20-Poly1305 and AES-GCM in the context of the TLS 1.3 protocol. The study demonstrated that ChaCha20 provides stable performance on devices without AES hardware support and reduces power consumption, which is critical for mobile platforms. R. Serrano *et al.* (2022) emphasised the development of the ChaCha20-Poly1305 cryptographic core for TLS protocols, emphasising protection against side-channel attacks, such as energy consumption analysis and cache attacks. V.R. Kebande (2023) proposed the use of ChaCha20 in the new WireGuard VPN protocol, arguing that it offers high performance and ease of implementation even in resource-constrained environments. Proposed variant of the algorithm with an extended number of rounds (EChaCha20) increased resistance to statistical attacks without a noticeable decrease in performance. In addition, J.P. Degabriele *et al.* (2021) investigated the impact of nonce parameters on the robustness of ChaCha20 implementations, particularly when the key is reused in IoT devices.

Z. Najm *et al*. (2018) compared the power consumption and side-channel attack vulnerability between ChaCha20-Poly1305 and AES-GCM implementations on microcontrollers. The results demonstrated that ChaCha20-Poly1305 consumes approximately 7 μW per 50 bytes, which is significantly less than the 27 μW consumed by AES-GCM. In addition, ChaCha20-Poly1305 proved to be more resistant to attacks targeting time variations and power consumption, in particular due to more effective countermeasures against side channels. M. Polubelova *et al.* (2020) presented a compact, time-attack-resistant implementation of ChaCha20-Poly1305 on ARM Cortex-M4, targeting IoT devices without AES hardware support, with practical measurements of delays and throughput, reducing the risk of errors associated with manual optimisation.

Thus, the scientific community is actively researching the ChaCha20 algorithm in terms of both theoretical stability and practical implementation, confirming the relevance of further study of its modifications and areas of application. Despite considerable attention to individual aspects of ChaCha20, there is a lack of comprehensive studies that comprehensively cover the stages of its development, types of modifications, and practical implementation experience. The purpose of this article was to conduct a systematic review of the ChaCha20 algorithm, analyse its main modifications, and evaluate its scope of application in modern cryptographic systems. To achieve this goal, the following tasks were set: to analyse the origin of the ChaCha algorithm and its differences from its predecessor, Salsa20; to conduct a comparative analysis of ChaCha20 with other symmetric ciphers (in particular, AES) in terms of performance and cryptographic strength; to summarise the current experience of using and modifying ChaCha20 (TLS, VPN, XChaCha20, AEAD (Authenticated Encryption with Associated Data) mode).

Materials and Methods

The study was based on a comparative analysis of the ChaCha20 stream cypher and AES block cypher algorithms in CTR (Counter Mode) and GCM modes. The methodological basis covered several complementary approaches: analysis and synthesis of scientific and technical literature, systematisation and comparison of empirical data, qualitative and quantitative evaluation of various implementations, interpretation of experimental reports, and adaptation of known methods to the specifics of microcontroller platforms. The main approach was comparative analysis, which makes it possible to identify the strengths and weaknesses of the algorithms under study in terms of their performance, cryptographic strength, and suitability for practical application in various operating scenarios.

The research was based on peer-reviewed scientific articles published between 2008 and 2024, international standards, in particular RFC 7539 (Nir & Langley, 2015), and technical reports from companies such as Google and Cloudflare (Krasnov, 2016). The sources were selected from the Scopus and Google Scholar scientific databases using the Google search engine according to clearly defined criteria: relevance to the topic, availability of quantitative and qualitative test results, openness of experimental methods, relevance to the present technical context, and the authority of the sources from the point of view of the scientific community and industrial standards.

Analysis of testing results for ChaCha20 and AES algorithm performance on different hardware platforms was emphasised. Intel Haswell processors with AES-NI instructions (Intel, USA), ARM Cortex-A53 mobile processors installed in Google Pixel smartphones (ARM Holdings, UK), as well as popular STM32 and ESP32 microcontroller platforms (Espressif, China) were studied. The set of comparable metrics included the following indicators: small data block processing latency, processor cycle speed per byte, throughput, power consumption, side-channel attack resistance, and parallel data processing capability for performance improvement.

The analysis was performed solely based on secondary data obtained from official reports, benchmarks, and independent studies. It is based on the results of comprehensive

tests that demonstrated the superiority of ChaCha20-Poly1305 over AES-GCM in environments without specialised hardware support (De Santis *et al.*, 2017). Adapted methodologies from publications on WireGuard and Noise Framework were also used to evaluate the effectiveness of implementations on microcontrollers (Donenfeld, 2017). No laboratory experiments were conducted; conclusions were formed solely based on a thorough analysis and generalisation of available empirical results.

The sequence of the study was determined following the tasks set. The first stage was an analysis of the history of the ChaCha20 algorithm and its fundamental differences from its predecessor, Salsa20. Next, a comparative analysis of ChaCha20 with other common symmetric cyphers, in particular AES, was conducted in terms of performance and cryptographic strength. The next step was to summarise the modern experience of the use of ChaCha20 in security protocols such as TLS and VPN. The final stage considered the latest modifications to the algorithm, including XChaCha20, AEAD mode and an increase in the number of rounds. This evaluated the effectiveness of the algorithm in resource-constrained environments and investigated energy consumption and throughput in real-world applications. All these stages of the research were based on comparative analysis, which identified the strengths and weaknesses of different cryptographic algorithms and concluded on their practical application in modern cryptosystems.

Results and Discussion

History of the development of the ChaCha algorithm

The ChaCha algorithm is derived from the Salsa20 stream cypher, developed by D.J. Bernstein in 2005 for the eSTREAM competition (Bernstein, 2008). Salsa20 was one of the finalists in eSTREAM and proved itself to be a fast and secure cypher. In a typical Salsa20/20 implementation (20 rounds), it runs faster than AES and was considered by the cryptographic community to be quite reliable. The researcher also proposed reduced versions of Salsa20/12 and Salsa20/8 (12 and 8 rounds, respectively) for scenarios where speed is prioritised over maximum resistance. Salsa20/12 was included in the final eSTREAM portfolio in 2008 as a promising stream cypher for widespread use in software implementations.

In 2008, D.J. Bernstein published a new version of Salsa20 called ChaCha. The goal of ChaCha was to increase diffusion (bit mixing) during each round without losing performance. The main change concerned the round function: ChaCha retained the basic structure of Salsa20 (16 state words, mod 2^{32} addition, XOR, and cyclic shift operations), but reorganised the sequence of operations in the quarter-round, which is the basic state transformation step. In contrast to Salsa20, where each 32-bit word is updated once per quarter-round, in ChaCha each word is updated twice, which significantly improves the distribution of changes across bits. In other words, ChaCha can use each input word to affect all output words in a single round, whereas in Salsa20 the effect was more limited. This doubling of diffusion is a key difference that, as analysis has shown, can be used in ChaCha to mix bits faster: a single 1-bit change at the input of a ChaCha quarter-round affects an average of 12.5 output bits (in the absence of carry), while in Salsa20 it affects only 8 bits. In addition to rearranging the operations, ChaCha used slightly different cyclic shift constants. Salsa20 used shifts of 7, 9, 13, and 18 bits, while ChaCha uses shifts of 16, 12, 8, and 7 bits. This change was less significant for diffusion (the difference is considered insignificant in terms of cryptographic strength) and only slightly affects the speed on certain platforms. Thus, the main improvements in ChaCha emphasise the state update scheme.

ChaCha was presented in three versions based on the number of rounds: ChaCha8, ChaCha12, and ChaCha20 (similar to the Salsa20/8, /12, /20 line). The full version of ChaCha20 (20 rounds) is designed for maximum security and replaces Salsa20/20 without compromising overall performance. The reduced versions ChaCha8 and ChaCha12 were offered for environments where a lower margin of security is acceptable in favour of higher speed, similar to Salsa20/8 and Salsa20/12. However, in practice, the ChaCha20 version with the maximum number of rounds has become established in standardised applications, as even this version's performance remains high.

The cryptography community quickly analysed the new algorithm. J.P. Aumasson *et al.* (2008) published the results of cryptanalysis of Salsa20 and ChaCha (called "Latin Dances"), showing that ChaCha does indeed have higher round resistance: the best attack on ChaCha was able to cryptanalytically cover one round less than for Salsa20. It was possible to break (recover the key) for Salsa20, reduced to 8 rounds, with a complexity of ~ 2^{251} operations in the presence of ~ 2^{31} pairs of keystream outputs. For full-round Salsa20/20 and ChaCha20, no practical attacks are known except for key brute force. Over more than two decades of research into the cryptographic strength of ChaCha20, no effective practical attacks against the full 20-round version of the algorithm have been found. Known cryptanalytic results apply only to simplified versions of the algorithm with no more than 7–8 rounds and are mainly theoretical in nature, posing no real threat to the practical security of the algorithm. This indicates the high level of cryptographic strength of ChaCha20 and its significant security margin, which explains its widespread use in modern encryption systems.

Technical aspects of ChaCha: structure and cryptographic strength

Structure. ChaCha20 is a stream cypher based on addition-rotation-XOR (ARX) that transforms a 256-bit key, a 32-bit block counter, and a 96-bit nonce (a one-time random value) into a 512-bit block of pseudo-random keystream. The algorithm operates on an internal state of 16 words of 32 bits each (512 bits in total), which can be conveniently represented as a 4×4 matrix of words. The initial state is formed from the key, nonce, and constants as follows: the first row contains 4 fixed 32-bit constants

(ASCII string “expand 32-byte k”), the next eight words are a 256-bit key, followed by two words – the block counter (initialised to zero for the first block), and the last two words are 64 bits of “nonce” (in the original ChaCha, as in Salsa20, “nonce” is 64 bits; in Internet Engineering Task Force (IETF) implementations, 96 bits are used, with a slightly different breakdown). Thus, the initial 4×4 matrix is conditionally shown in Table 1.

Table 1. Initial state of the ChaCha20 algorithm matrix

const	const	const	const
key	key	key	key
key	key	key	key
ctr	ctr	nonce	nonce

Note: const – constant words “expa”, “nd 3”, “2-by”, “te k”; key – 256-bit key (divided into 8 words); ctr – counter; nonce – unique two-word number for each encryption stream

Source: A. Langley *et al.* (2016)

This state undergoes a series of rearrangement rounds, after which it is summed (component-wise in 32-bit chunks) with the initial state, forming a 512-bit keystream output block, which is then XORed byte-by-byte with the plaintext to obtain the ciphertext. When encrypting large amounts of data, the block counter automatically increments, generating subsequent 64-byte stream blocks. The main state transformation occurs through the repeated application of a quarter-round basic operation on four 32-bit words. Four words (labelled a, b, c, d) are taken as input to the quarter-round, and they are updated at the output. The sequence of operations in the ChaCha quarter-round is shown in the pseudocode “Quarter-round function of the ChaCha20 algorithm”:

```
QuarterRound(a, b, c, d)
a += b; d ^= a; d <<<= 16,
c += d; b ^= c; b <<<= 12,
a += b; d ^= a; d <<<= 8,
c += d; b ^= c; b <<<= 7.
```

Quarter-round processes a quarter of the entire 4×4 state matrix. ChaCha20 consists of 20 rounds, with quarter-round applied to four columns of the matrix in parallel in odd rounds (1, 3, 5, ...) and to four diagonals of the matrix in even rounds (2, 4, 6, ...). This alternating scheme (column round + diagonal round) ensures that the data is shuffled both by columns and by rows of the matrix, which in total over two rounds (double round) affects all 16 words. After executing the specified number of rounds (for ChaCha20, 20, i.e. 10 double rounds), the resulting state is added modulo 2^{32} to the initial state (this is the so-called Feed-Forward operation, inherited from the Salsa20 design). The result is interpreted as 64 bytes of key stream, ready for XOR with plaintext. The decryption process is identical to encryption, since XOR with the same key stream restores the plaintext. The structure of the algorithm is shown in Figure 1.

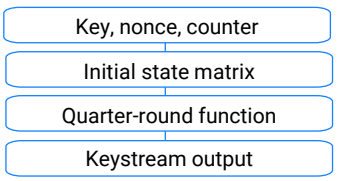

Figure 1. General diagram of the ChaCha20 algorithm
Source: A. Langley *et al.* (2016)

Notably, ChaCha uses only simple integer operations (addition, XOR, shift) that are implemented in hardware on all modern processors and are data-independent (no conditional jumps or indexed memory accesses). This renders ChaCha implementations resistant to timing attacks and side-channel attacks such as cache timing, which can be dangerous for cyphers with substitution tables (e.g., AES without hardware AES-NI). The algorithm is also not patented; the author has published several high-performance implementations in the public domain, which have facilitated its acceptance by the community.

Cryptographic resilience. The ChaCha20 algorithm is designed to provide 256-bit security, meaning that brute-forcing the key requires 2^{256} attempts, which is an astronomically large number. All known cryptanalysis methods failed to crack the full 20-round version. The best results relate to simplified versions with fewer rounds. For example, differential cryptanalysis and its variations have managed to find weaknesses in up to 7 rounds of ChaCha. For example, in 2018-2023, improved differential-linear attacks were reported that can distinguish a 7-round ChaCha permutation from a random one (selecting the output stream with a certain statistical difference) with a complexity significantly lower than brute force. However, these attacks do not lead to key recovery and do not scale to a larger number of rounds. For 8 rounds of ChaCha, the attack complexity is estimated to be around 2^{251}, which is only insignificantly better than a full key search (2^{256}) and is purely theoretical. Thus, from a practical point of view, ChaCha20 is considered reliable: even with the use of modern supercomputers or quantum accelerations (Grover’s algorithm provides insignificant acceleration compared to quadratic acceleration, effectively reducing the complexity to ~2_{128}), breaking the algorithm is unrealistic when used properly (Barbero *et al.*, 2022).

A substantial security requirement for ChaCha (as for most stream cyphers) is that the nonce must be unique for each key. If the same set {key, nonce, initial counter} is accidentally used twice to encrypt different messages, then due to the property of XOR with the same key stream, certain information regarding the two plaintexts (in particular, their XOR with each other) can be obtained. In the worst case, a repeated nonce compromises the messages.

Therefore, protocols with ChaCha20 require that nonce values for a single key never be repeated. The standard version of ChaCha20 (as defined in RFC 7539) uses a 96-bit nonce, which gives an astronomical number of possible values (around 2^{96}) – this is sufficient for most applications, although theoretically there is a small probabilistic chance of a nonce collision when generating random 96-bit numbers. To eliminate the risk of repetition, a modification of XChaCha20 called ChaCha20 "extended nonce" has been proposed, which supports a 192-bit nonce (Arciszewski, 2019). XChaCha20 first passes the key along with the first 128 bits of the nonce through HChaCha20, a 256-bit ChaCha20-based auxiliary permutation, generating a new internal key, after which encryption continues with regular ChaCha20 using this key and the remaining 64 bits of the nonce. This can be used for the safe use of extra-long nonces (24 bytes) with a collision probability of practically zero, which is beneficial for long-term sessions or systems where it is difficult to guarantee the uniqueness of short nonces. In terms of robustness, XChaCha20 maintains the same cryptographic assumptions as ChaCha20.

In summary, ChaCha20 has a simple and robust ARX structure, is well-suited for secure implementation in software, and demonstrates a high margin of safety against known attacks. Thanks to its combination of high performance, resistance to parallel computing, and well-analysed cryptographic security, ChaCha20 has become a popular choice in modern data protection protocols such as TLS, WireGuard, and QUIC. Its widespread adoption demonstrates the algorithm's recognition as an effective solution that meets modern security requirements.

Comparison of ChaCha with other cyphers (Salsa20 and AES).

ChaCha vs Salsa20. Since ChaCha is a direct descendant of Salsa20, it is necessary to consider their differences and similarities. Both algorithms use the same basic operations (32-bit addition, XOR, cyclic shifts) and work with a 512-bit state block formed from a 256-bit key and a 64-bit (or 96-bit) nonce. The performance of both ciphers is substantial: Salsa20 in software implementation achieves ~4-14 cycles per byte on standard CPUs, ChaCha20 has a similar order of magnitude. In fact, when designing ChaCha, D.J. Bernstein sought not to compromise speed relative to Salsa20. As stated in technical report, one round of ChaCha performs the same number of operations (16 additions, 16 XORs, 16 shifts) as a round of Salsa20 and maintains the same level of parallelism. In some architectures, ChaCha even saves one CPU register compared to the "native" implementation of Salsa20. Theoretically, ChaCha was expected to have a similar performance to Salsa, and possibly even better on certain platforms. Practical measurements confirmed this: for 8-round versions (ChaCha8 vs Salsa20/8), ChaCha showed the same or slightly better speed on most of the tested processors (for example, on 32-bit PowerPC G4 and x86 Pentium M, ChaCha8 was 6-8% faster than Salsa20/8). Only on some older 32-bit CPUs, such as Pentium 4, did Salsa20 outperform ChaCha (up to 30% faster), which is due to the architectural features of that platform. For full-round implementations, the difference is even smaller: ChaCha20 and Salsa20/20 have almost identical performance, with differences within a few percent depending on the environment. The results of the comparison between ChaCha and Salsa20 are shown in Table 2.

Table 2. Comparative table of ChaCha and Salsa20 cyphers

Characteristic	ChaCha	Salsa20
Algorithm type	Streaming, ARX	Streaming, ARX
Bloc size	512 bit	512 bit
Key size	256 bit	256 bit
Nonce size	64 or 96 bit (XChaCha: 192 bit)	64 or 96 bit (XSalsa: 192 bit)
Operations per round	16 additions, 16 XORs, 16 cyclic shifts	16 additions, 16 XORs, 16 cyclic shifts
Round types	Quarter-round with a diagonal column structure	Quarter-round with row-column structure
Typical performance values	~4-14 CPU cycles/bytes	~4-14 CPU cycles/bytes
Cryptographic resilience	Higher diffusion, fewer rounds to achieve equivalent security	Requires more rounds

Source: D.J. Bernstein (2008)

Hence, the transition from Salsa20 to ChaCha did not worsen performance but brought gains in robustness: ChaCha requires fewer rounds to achieve an equivalent level of security. Therefore, with the same security (e.g., 8 rounds of ChaCha vs 8 rounds of Salsa20), ChaCha is slightly better in terms of speed, and when using the full 20-round version, a greater margin of security is obtained with virtually no loss of performance. As a result, ChaCha has effectively replaced Salsa20 in the latest protocols: although Salsa20 is also considered secure, ChaCha20 has become predominant in implementations due to its better diffusion and community support.

ChaCha vs AES. These algorithms compete in real-world applications (e.g., AES-256 in CTR or GCM mode vs ChaCha20-Poly1305). AES is a 128-bit block cypher with a multi-round SP network (substitutions and permutations), optimised for hardware execution: modern processors contain AES-NI instructions that can encrypt an AES

block in 1-2 cycles. Thanks to this, AES-128-GCM is fast on desktop and server CPUs, often less than 1 clock cycle per byte when streaming large volumes. However, in environments without hardware support (this includes most mobile devices, many embedded systems, and some CPUs of other architectures), AES in software implementation runs significantly slower and is even potentially vulnerable to external attacks due to its dependence on S-box tables. In such cases, ChaCha20 demonstrates a clear advantage: it is designed specifically for efficient operation on general-purpose CPUs without special instructions. ChaCha20 consists entirely of operations that are performed uniformly fast on any processor: addition, XOR, shift (they do not require memory and do not cause cache misses) on simple ARM cores. The ChaCha20 cypher can be several times faster than AES-256 without AES-NI. A practical example: on a Galaxy Nexus smartphone (ARM Cortex-A9, without AES acceleration), decrypting 1 MB of data with the AES-128-GCM algorithm took ~41.6 ms, while ChaCha20-Poly1305 required only ~13.2 ms. This means that ChaCha20-Poly1305 is approximately three times faster than AES-128-GCM on this mobile device. This saves a significant amount of time and, crucially for portable devices, energy consumption (less CPU load means longer battery life) (Sullivan, 2014). AES-128-GCM and ChaCha20-Poly1305 provide a comparable level of security (~128 bits of effective strength, given the 128-bit randomness of authentication in GCM and Poly1305), so the comparison is valid.

In contrast, on platforms with hardware support for AES, the situation is opposite – hardware instructions make AES-GCM faster. For example, on Intel Haswell (and newer) AES-128-GCM outperforms ChaCha20-Poly1305 in throughput, especially on large data blocks. ChaCha20-Poly1305 uses only general-purpose SIMD instructions, and with the development of instruction sets (e.g., the proliferation of 512-bit AVX-512 vector instructions), the difference may narrow (Cai, 2022). On some ARM64 processors with neon optimisation, ChaCha20 performs at the level of AES. Therefore, when choosing between AES and ChaCha, the availability of hardware acceleration should be considered: for servers and desktops with AES-NI, AES-GCM is often more appropriate, while for mobile, IoT, and other devices without such acceleration, ChaCha20-Poly1305 offers a significant speed advantage without compromising security. Therefore, modern protocols provide for the possibility of using both, such as TLS 1.3, which defines cypher suites based on both AES-GCM (Rescorla, 2018) and ChaCha20-Poly1305, providing the optimal choice in a particular case (for example, the Chrome browser on Android will prefer ChaCha if the CPU does not have AES-NI). The results of the comparison between ChaCha and AES are shown in Table 3.

Table 3. Comparative table of ChaCha and AES cyphers

Characteristic	ChaCha	AES
Algorithm type	Streaming, ARX	Bloc, SP-network
Bloc size	512 bits	128 bits
Key size	256 bits	128/192/256 bit
Nonce size	64 or 96 bits (XChaCha: 192 bit)	96 bits
Operations per round	16 additions, 16 XORs, 16 cyclic shifts	Substitutions (S-Box), XOR, permutations
Round types	Quarter-round with a diagonal column structure	SP-round
Typical performance values	~4-14 CPU cycles/bytes	~20-60 CPU cycles/bytes
Cryptographic resilience	Higher diffusion, fewer rounds to achieve equivalent security	Requires AES-NI for speed and protection against cache attacks

Source: compiled by the authors based on D.J. Bernstein (2008)

Compared to the security of AES and ChaCha20, both algorithms have no practical vulnerabilities when used correctly. AES has a longer history of research, but ChaCha20 has also gained significant trust among experts. One of the advantages of ChaCha is its less complex and easier to implement structure – ~4,000 lines of code in a typical implementation versus tens of thousands in AES-based protocols (such as OpenVPN with different modes). Smaller and simpler code reduces the risk of implementation errors and simplifies security audits. On the other hand, AES, as a widely used standard, has a hardware implementation that minimises the risk of software bugs. Overall, both cyphers are considered reliable and are recommended by standards (for example, TLS 1.3 recommends the use of AES-GCM and ChaCha20-Poly1305). In 2014, A. Langley publicly endorsed ChaCha20-Poly1305 as an alternative to AES for mobile devices without hardware acceleration (Langley *et al.*, 2016). Thus, ChaCha20 has successfully complemented AES, providing an alternative that improves resistance to potential attacks on homogeneity (crypto-monoculture) and increases the flexibility of security systems.

Use of ChaCha in modern cryptographic systems

Since its inception, ChaCha (especially ChaCha20) has gradually gained recognition and integration into various protocols and standards. Below are the most relevant areas of application for this algorithm in modern systems. One of the first major implementations of ChaCha20 was its integration into the TLS protocol, which provides HTTPS web traffic encryption. The initiative came from Google:

in 2013, engineers were searching a fast and secure cypher stream to use in Chrome on Android and other clients without AES hardware acceleration. The choice fell on the ChaCha20 + Poly1305 combination, and in 2014, Chrome began supporting an experimental set of TLS cyphers with ChaCha20-Poly1305 (Sullivan, 2014).

After a period of testing and coordination, the IETF standardised this connection: first as an Internet draft CFRG (Crypto Forum Research Group), and later in RFC 7905, several CipherSuite TLS 1.2 were defined using ChaCha20-Poly1305 with HMAC (Hash-based Message Authentication Code) on SHA-256 for handshaking. The new standard has been supported in libraries such as OpenSSL and BoringSSL. This became particularly relevant after the dangerous RC4 (Rivest Cypher 4) was removed from TLS, and an alternative to AES-GCM was needed for older devices. In TLS 1.3 (standardised in 2018), the ChaCha20-Poly1305 suite became one of the core suites (Rescorla, 2018). According to the protocol requirements, all TLS 1.3 clients and servers must support two cypher suites: TLS_AES_128_GCM_SHA256 and TLS_CHACHA20_POLY1305_SHA256. This confirms ChaCha20-Poly1305's status as an equal component of modern web cryptography.

At the same time, ChaCha20-Poly1305 in TLS 1.3 is positioned as a solution that improves performance and resistance to side-channel attacks in software execution (Fig. 2). According to research, the implementation of ChaCha20 in HTTPS has significantly increased page loading speeds on mobile devices and reduced battery consumption. Currently, virtually all popular browsers (Chrome, Firefox, Safari, Edge) and servers (e.g., nginx via OpenSSL) support ChaCha20-Poly1305. Statistics show an increase in the share of TLS connections using this cypher: in the first years after the standardisation of TLS 1.3, the use of ChaCha20-Poly1305 increased sharply, as it is automatically selected for at least some clients (mainly mobile ones). For instance, P. Crowley & E. Biggers (2019) reported that a significant percentage of their network traffic is already protected by ChaCha20-Poly1305 thanks to Chrome/Firefox browsers, which dynamically select this cypher for clients without AES-NI.

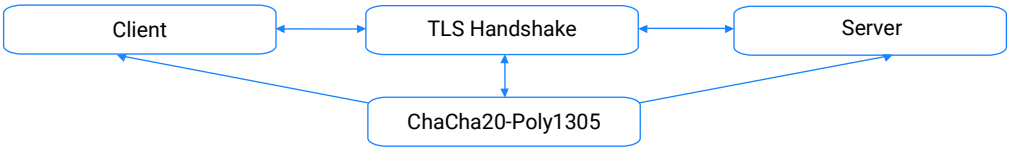

Figure 2. Use of ChaCha20-Poly1305 in TLS 1.3

Source: A. Langley *et al*. (2016)

Another area where ChaCha gained popularity was virtual private networks (VPNs) and other tunnelling protocols. The most famous example is the WireGuard protocol, introduced in 2016 as a modern alternative to IPsec and OpenVPN. WireGuard is designed with an emphasis on simplicity and speed and uses only ChaCha20 for symmetric encryption (with Poly1305 for authentication). The developers of WireGuard (Donenfeld, 2017) argued that ChaCha20 provides better or comparable encryption speed compared to AES, especially on typical router, server, and smartphone hardware, where AES instructions may be absent or where multithreading and simple code are more relevant. As a result, WireGuard is notable for the use of ChaCha20 for encryption, which is a faster alternative to the common AES-256 in other VPN protocols. Practical tests demonstrate that WireGuard outperforms OpenVPN in terms of throughput and latency, partly due to its choice of the lightweight ChaCha20 cypher and optimised implementation on kernels (running in OS kernel mode). The lack of algorithm choice (monoculture) in WireGuard is compensated for by thorough security analysis: ChaCha20-Poly1305 is used in the Noise_IK cryptographic protocol, and independent experts have confirmed the cryptographic strength of this solution. Thus, according to the conclusion of P. Crowley & E. Biggers (2019), ChaCha20 has become the basis for one of the most promising VPNs, which is already included in the Linux kernel and supported by many services.

Due to high performance on ARM processors, ChaCha20 has become popular not only for network traffic but also for data encryption in storage. In 2019, Google introduced the Adiantum algorithm (Crowley & Biggers, 2019), a new encryption mode for Android devices designed to protect disk storage on budget phones, smartwatches, and IoT devices that do not have AES hardware acceleration. Adiantum is based on XChaCha20-Poly1305, combined with a special permutation code (NH Poly1305 + enhancements) to achieve storage encryption length (i.e., without increasing data size), in contrast to standard GCM-type modes. The choice of ChaCha20 for Adiantum was due to the fact that on simple Cortex-A7 cores, the performance of AES-XTS (standard disk encryption mode) was insufficient, less than 50 MB/s, which slowed down the device. ChaCha20, on the other hand, can encrypt substantially faster by using only basic instructions. Adiantum can be used to encrypt all devices without a noticeable drop in performance: ChaCha20 is significantly faster than AES in the absence of hardware acceleration, while remaining secure. Adiantum is included in Android (starting with version 10) as an FBE (File-Based Encryption) option for devices that do not support AES-NI, thus protecting millions of budget smartphones around the world with ChaCha20. This is a case where a modern algorithm provides cryptographic protection to the "next billion" users, for whom encryption would otherwise be disabled due to the low performance of AES.

Another area of application is the generation of pseudorandom numbers for cryptography (e.g., /dev/urandom in operating systems). Traditionally, RC4 or other modern algorithms are used. Many implementations have switched to ChaCha20 as the core of the random stream generator. In particular, in the OpenBSD and FreeBSD operating systems, the arc4random() function is now implemented using ChaCha20 instead of the outdated RC4. The use of RC4 in arc4random has been criticised for its weak random number generation, as shown in a study by T. Ristenpart *et al.* (2009). The Linux kernel also uses ChaCha20 to initialise the entropy pool. Reasons: ChaCha20 is fast, has no known vulnerabilities, and is easy to implement without complex states. Thus, it provides a reliable stream of randomness for the cryptographic needs of the OS.

ChaCha20-Poly1305 is also used in other protocols: SSH (Secure Shell) (OpenSSH added this connection as an encryption option) – for interactive sessions, it provides less latency than AES. The Quick UDP Internet Connections (QUIC) protocol, developed by Google to replace TCP, also supports ChaCha20-Poly1305. Many high-level libraries (NaCl/libsodium, BoringSSL, etc.) offer ChaCha20 as one of their basic primitives. At the standards level, the algorithm is defined in RFC 7539 (Nir & Langley, 2015), recommended by the National Institute of Standards and Technology (NIST) for software implementations (Dworkin, 2016), and included in the list of algorithms approved for use in Internet standards. The international cryptography community is paying attention to both the analysis of ChaCha (to maintain trust) and the optimisation of its implementations for different platforms. For example, hardware implementations of ChaCha20 are being researched: there are IP cores for FPGAs that achieve throughputs of tens of Gbit/s, making ChaCha competitive even in high-end applications (Pfau *et al.*, 2019). Thus, ChaCha20 is widely integrated into critical security protocols (TLS, SSH, and IPsec via WireGuard) and data encryption implementations, especially where software performance is critical. It is one of the few new algorithms that has been able to occupy a niche alongside AES, complementing it.

Modifications and improvements to the ChaCha algorithm

Based on the success of ChaCha20, researchers and engineers further improved the algorithm, both in terms of increasing performance, enhancing security, and expanding areas of application (Procter, 2014). The main modifications and variants of ChaCha that appeared after its standardisation were considered. These improvements have made it possible to use this algorithm in more demanding environments, including resource-constrained systems, and to integrate it more effectively with other cryptographic protocols. Some variants, such as XChaCha20, have gained widespread support and are actively used in practice.

XChaCha20 is a variant with an extended nonce, designed primarily to eliminate the risk of repeating one-time values. The standard 96-bit nonce, although practically sufficient, can be exhausted with large amounts of encryption (the theoretical upper limit is 2^{32} blocks of 64 bytes, i.e. about 2^{38} bytes ~ 256 GB per key). XChaCha20 supports the use of a 192-bit nonce, which is practically inexhaustible. Technically, this is achieved using the HchaCha20 function – initialising a ChaCha20-like permutation on a 128-bit nonce to obtain an intermediate key. At the same time, the security of XchaCha20 is based on the same assumptions as ChaCha20, and the relevant IETF drafts (draft-irtf-cfrg-xchacha) recommend it for scenarios where very large data needs to be encrypted or long-term keys need to be maintained without the risk of nonce repetition (Arciszewski, 2019). XChaCha20 has already been implemented in popular libraries, such as libsodium, and is used in protocols (WireGuard uses 24-byte nonces for internal needs). This modification is aimed more at system reliability than at changing the core of the algorithm, but it is significant for the practical application of ChaCha in large-scale systems (Fig. 3).

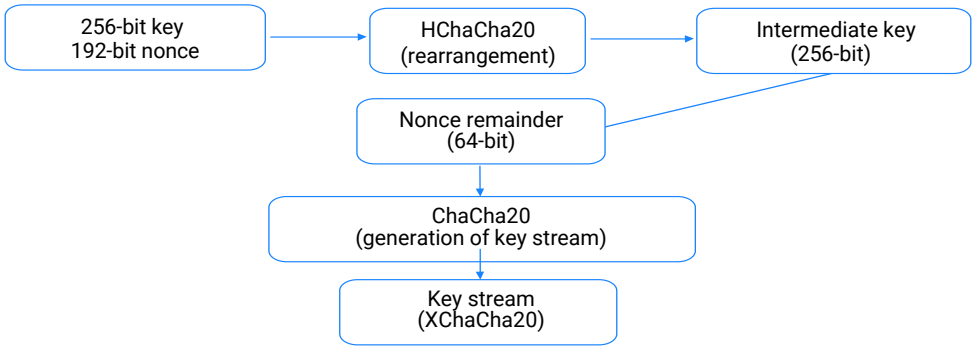

Figure 3. XChaCha20 algorithm diagram with extended nonce

Source: A. Langley *et al.* (2016)

ChaCha20-Poly1305 (AEAD) is not a modification of ChaCha itself, but a combination with MAC. The authenticated mode AEAD_CHACHA20-POLY1305 has become the de facto standard for using ChaCha in protocols. Before the advent of this AEAD, ChaCha20 was also proposed for use with traditional MACs (e.g. HMAC-SHA1), but during

the standardisation of TLS, it was decided to use Poly1305 as a faster and simpler option. Poly1305 is an integrity verification algorithm (MAC), also invented by D. Bernstein, optimised for 64-bit multiplication operations. The ChaCha20+Poly1305 combination provides both confidentiality and authenticity of messages; these algorithms are now mentioned together. Notably, Poly1305 obtains its key from ChaCha20 (by encrypting 32 bytes of zeros with a separate single-byte block with a special "nonce"), which eliminates the problems of MAC key reuse. This pair has undergone numerous security tests and has repeatedly proven its robustness in practice. In the context of the present study, ChaCha20-Poly1305 can be considered a composition that has significantly expanded the scope of ChaCha, making it a full-fledged AEAD component for modern protocols.

Although ChaCha20 dominates the standards, in some cases, options with fewer rounds are considered for higher speed. ChaCha12 provides a substantial level of security (all known attacks do not progress beyond 7-8 rounds); therefore, it is considered potentially secure. Google used ChaCha12 in Android Disk Encryption prototypes (before Adiantum) to accelerate encryption and noted the absence of known vulnerabilities at 12 rounds. Some implementations (e.g., OpenSSL) can be configured to reduce the round count of ChaCha for experimentation or specific needs. However, ChaCha12/8 has not gained widespread use because the speed advantage is not significant enough (ChaCha20 is already highly efficient) to sacrifice robustness. Therefore, ChaCha20 is usually used as the representative variant in performance reviews. But the ability to adjust rounds – the flexibility of the algorithm – is sometimes used by researchers to test security limits or speed gains (Maitra, 2016; Dey & Sarkar, 2023; Xu *et al.*, 2024).

On the other hand, in 2023, modifications to ChaCha were proposed with an increase in the number of rounds for even greater stability. For example, V.R. Kebande (2023) presents Extended-ChaCha20 (EchaCha20), a variant that uses 36 quarter-rounds (i.e., 36 rounds or 18 double rounds) instead of 20. The motivation is to increase the difficulty of differential attacks and improve the results of statistical randomness tests. The author tested EchaCha20 and showed that the algorithm successfully passes all NIST STS randomness tests, similar to ChaCha20, while demonstrating comparable performance to the original. In particular, the encryption/decryption time and memory usage remained unchanged when moving from 20 to 36 rounds, making EchaCha20 interesting for applications where maximising resilience is critical (perhaps in the context of post-quantum considerations, although a 256-bit key is already sufficient with a large margin). Currently, EchaCha20 is an academic proposal, but it demonstrates that the algorithm can be scaled towards security without significant performance losses. Such research expands the ChaCha's limits and may be reflected in future standards or specific protocols.

Although ChaCha20 is designed for software implementation, there are developments in hardware implementations. Some IP core manufacturers, including Silex Insight, have introduced scalable FPGA/ASIC cores ChaCha20-Poly1305, capable of providing throughput of up to 100 Gbps on FPGA and up to 800 Gbps on ASIC, which meets the modern requirements of high-performance encryption systems (Rashidi, 2024) This is achieved thanks to ChaCha's high level of parallelisation – for example, processing several blocks simultaneously on the pipeline. Such solutions can be used in network filters and VPN gateways, where AES is also implemented. Notably, even in a hardware environment, ChaCha can compete: the implementations cited achieve <1 cycle per byte on an FPGA, which is comparable to AES. Thus, ChaCha20 is gradually penetrating the hardware world. Modern implementations of ChaCha20-AEAD (e.g., the leancrypto library, version 1.5.1, 2025) actively use Intel AVX2, AVX-512, and ARM NEON CPU instructions to achieve high-performance SIMD acceleration on x86_64 and ARMv8 architectures. Each new generation of processors reduces the gap between AES and ChaCha or improves ChaCha's already substantial performance. For example, on x86 processors with AVX2, the ChaCha20 implementation can encrypt at a rate of ~12-15 GB/s per core, which is sufficient for most tasks, and with the release of AVX-512, this speed is expected to double.

In summary, the ChaCha ecosystem continues to evolve: variants have been developed for different needs (XChaCha20 – for nonce uniqueness, ChaCha20-Poly1305 – for AEAD, truncated/enhanced round versions – for speed/security balance), and methods of implementing the algorithm in both software and hardware are being improved. These modifications strengthen ChaCha's position as a long-term stream cypher standard. In the context of the active support of the community and industry, further expansion of the algorithm's applications can be expected. ChaCha20, thanks to its flexibility and high security, is already the basis for numerous modern cryptographic solutions.

Analysis of ChaCha's performance compared to other cyphers

One reason for ChaCha's popularity is its exceptional performance on different platforms. Fast performance on desktop CPUs. As already mentioned, the AES-GCM algorithm has an advantage on modern x86-64 processors with AES-NI support. Measurements conducted by Cloudflare showed that for message sizes > 1 KB, AES-128-GCM encrypts at a rate of ~1.5-1.8 cycles per byte, while ChaCha20-Poly1305 encrypts at ~2.0-2.3 cycles per byte. For small messages, ChaCha may even be slower due to the lack of specialised instructions (for example, at 64 bytes, AES significantly outperforms it). However, the difference is not critical: even 2 cycles/byte means that on a single 3 GHz core, ~1.5 GB of data per second can be encrypted, which is more than the throughput of most network interfaces. Therefore, for server applications where AES-NI is present, ChaCha20-Poly1305 is enabled primarily not for speed, but as a fallback in case AES becomes unsafe to use or if the client does not support AES-NI (Nir & Langley, 2015; Dworkin, 2016; Krasnov, 2016).

This is where ChaCha20 is the most utilised. For example, on popular 32-bit ARM Cortex-A7 processors (often used in inexpensive phones and IoT devices), AES encrypts ~10-20 MB/s (in CBC/CTR mode) due to the lack of acceleration, while ChaCha20 on the same core delivers 60+ MB/s. The situation is better on 64-bit ARMv8: Crypto Extensions instructions have appeared, but not all devices have them (for example, many mid-range smartphones from 2016–2018 had ARMv8 without AES instructions enabled) (Sullivan, 2014). In such cases, disabling AES-NI in OpenSSL benchmarks showed ChaCha20-Poly1305 accelerating up to 3-4 times over AES-256-GCM. This is consistent with the independent Google tests mentioned earlier. That is, for mobile platforms, ChaCha20 can process approximately 300-500 MB/s per core, while AES-GCM without hardware support barely reaches 100-150 MB/s. This gap was the decisive argument in favour of implementing ChaCha20 in Android, iOS and other mobile systems (Sullivan, 2014).

Another aspect of performance is the delay in processing small messages. In protocols such as SSH or VPN, where small packets are transmitted, minimal encryption latency is relevant. ChaCha20 has an advantage as it does not require complex dependent operations (AES has several rounds with non-linearities that are difficult to pipeline without hardware deployment). Thus, ChaCha can have less encryption delay per packet. For example, WireGuard has a better ping compared to AES-based VPNs precisely because of the lighter ChaCha encryption. ChaCha parallelisation is also possible: several 64-byte stream blocks can be generated simultaneously on different cores or SIMD chains, as the blocks are independent (different counter values). AES-CTR/GCM is similarly parallelised by blocks; therefore, there is parity. But ChaCha is also less resource-intensive: its implementation requires less memory and consumes less energy for the same amount of work (due to fewer common instructions, although this is a subtle point).

In microcontrollers, such a comparative analysis also favours ChaCha. On 8-bit/16-bit MCUs, AES is generally difficult to implement efficiently (due to byte-wise S-box processing), whereas ChaCha consists of simple operations that scale even on small bit depths (albeit slower) (Zinzindohoué *et al.*, 2017; Tsoupidi *et al.*, 2021). Some IoT protocols, such as the Noise Protocol Framework, support the use of ChaCha20 for secure connections between microcontrollers. Thus, ChaCha20 extends secure communication capabilities to devices where AES is not optimal. The comparison results are shown in Table 4.

Table 4. Comparative table of ChaCha20 and AES cyphers when processing small messages

Characteristic	ChaCha20	AES
Latency	Low, due to simple linear operations (addition, XOR, shifts) that are easily pipelined on the CPU without special instructions	Higher, due to complex non-linear operations (S-box), which are less easily pipelined without hardware acceleration (AES-NI)
Parallelisation	High encryption blocks are independent (different key stream blocks are generated in parallel)	High encryption blocks are independent (different key stream blocks are generated in parallel)
Resource consumption	Low, does not use large tables, smaller code size	Higher, especially without AES-NI (S-box tables required, larger code)
Energy consumption	Low, due to fewer simple operations (lower CPU load)	Higher, especially without AES-NI hardware instructions (higher complexity of operations, more CPU cycles)
Efficiency on 8/16-bit BSS	High, simple arithmetic operations are easily implemented on limited CPUs.	Low, complex and slow implementation via S-box on low-bit CPUs
Practical application	Wide, often used in IoT protocols, including Noise Protocol Framework and WireGuard for low-power devices	Limited, usually requires AES-NI hardware support or is very slow

Source: compiled by the authors

ChaCha20 is one of the fastest known symmetric cyphers on software platforms. It outperforms older stream cyphers (such as RC4) not only in security but also in speed, especially on modern CPUs capable of processing 32-bit operations in a single clock cycle. Compared to AES, ChaCha20 loses only when AES is supported by a specialised hardware module; in all other cases, ChaCha20 is at least as good, and often significantly better. This has made it the "default cypher" for many software implementations where cross-platform compatibility and constant execution time are substantial factors. J.P. Aumasson *et al.* (2008) emphasised in their technical review that ChaCha20-Poly1305 is automatically selected as the optimal alternative for all clients without AES-NI. Their results confirmed that ChaCha's performance is maintained even in the absence of specialised hardware acceleration, which is consistent with the findings of this study regarding the high efficiency of the algorithm on a wide range of platforms. ChaCha's performance scales up with increasing processor capabilities: larger register widths, more cores – all of which can be easily leveraged for acceleration, as the algorithm is linearly parallelisable. From a practical standpoint, ChaCha20-Poly1305 can be recommended for any system where hardware AES is not available or where the simplest and most reliable implementation is required. This approach is becoming

increasingly common in the industry, as confirmed by performance analysis data.

Conclusions

The ChaCha20 algorithm, created as an evolutionary development of Salsa20, is one of the key components of modern cryptography. After analysing its history, structure, properties, and applications, the following conclusions can be drawn. First, ChaCha has successfully achieved the goals set for improving Salsa20: by changing the round structure, it has increased diffusion per round without losing performance, as confirmed by both theoretical estimates and the absence of effective attacks on the full version. Second, ChaCha20 has demonstrated outstanding performance in software implementations, being a "mobile AES replacement": in environments without AES hardware acceleration, it provides a multiple speed advantage with equivalent cryptographic strength. This has had a direct impact on the industry: ChaCha20-Poly1305 has become the standard in TLS 1.3, VPN (WireGuard) and other protocols, where it has reduced the load on devices and improved energy efficiency. ChaCha20-Poly1305 was formally standardised in RFC 7539, which defines the AEAD construction format for modern security protocols. Thirdly, ChaCha20 is a flexible and adaptable algorithm: modifications (XChaCha20 for larger nonces, AEAD modes, versions with different numbers of rounds) and improvements have been created based on it, expanding its scope from web protocols to disk encryption and random number generation. It is well-suited for hardware implementations and scales to multi-core systems, which indicates the potential for using ChaCha in future solutions.

The value of ChaCha20 has been confirmed by its widespread adoption: in most modern cryptosystems, ChaCha20 is considered an option or standard. This algorithm has significantly improved the security and performance of many systems, making encryption more accessible to weaker devices. Overall, ChaCha20 was proved to be a reliable and effective stream cypher that complements classic algorithms and reduces the security ecosystem's dependence on a single solution (AES), thereby increasing the resilience of the entire cryptographic infrastructure. In the future, further research on ChaCha may focus on formal analysis of its robustness (e.g., proving resistance to certain attack models) and finding the optimal balance of rounds for different applications. In addition, the integration of ChaCha20 into new protocols (e.g., post-quantum hybrid VPN or QUICv2 schemes) and the study of its behaviour when interacting with quantum-resistant algorithms are also relevant topics.

Acknowledgements

None.

Funding

The study was not funded.

Conflict of Interest

None.

References

[1] Arciszewski, S. (2019). *XchaCha: eXtended-nonce ChaCha and AEAD_XchaCha20_Poly1305. Internet-Draft draft-irtf-cfrg-xchacha-01*. Retrieved from https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xchacha-01.

[2] Aumasson, J.P., Fischer, S., Khazaei, S., Meier, W., & Rechberger, C. (2008). New features of Latin dances: Analysis of Salsa, ChaCha, and Rumba. In K. Nyberg (Eds.), *Fast software encryption. FSE 2008. Lecture notes in computer science* (Vol. 5086, pp. 470-488). Berlin: Springer. doi: 10.1007/978-3-540-71039-4_30.

[3] Barbero, S., Bazzanella, D., & Bellini, E. (2022). Rotational cryptanalysis on ChaCha stream cipher. *Symmetry*, 14(6), article number 1087. doi: 10.3390/sym14061087.

[4] Bernstein, D.J. (2008). The Salsa20 family of stream ciphers. In M. Robshaw & O. Billet (Eds.), *New stream cipher designs. Lecture notes in computer science* (Vol. 4986, pp. 84-97). Berlin: Springer. doi: 10.1007/978-3-540-68351-3_8.

[5] Cai, W. (2022). Implementation and optimization of ChaCha20 stream cipher on Sunway taihuLight supercomputer. *The Journal of Supercomputing,* 78(3), 4199-4216. doi: 10.1007/s11227-021-04023-9.

[6] Crowley, P., & Biggers, E. (2019). *Introducing Adiantum: Encryption for the next billion users*. Retrieved from https://security.googleblog.com/2019/02/introducing-adiantum-encryption-for.html.

[7] Datadog Security. (n.d.). *RC4 encryption is now insecure*. Retrieved from https://docs.datadoghq.com/security/code_security/static_analysis/static_analysis_rules/go-security/import-rc4/.

[8] De Santis, F., Schauer, A., & Sigl, G. (2017). ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications. In *Design, automation & test in Europe conference & exhibition* (pp. 692-697). Lausanne: IEEE. doi: 10.23919/DATE.2017.7927078.

[9] Degabriele, J.P., Govinden, J., Günther, F., & Paterson, K.G. (2021). The security of ChaCha20-Poly1305 in the multi-user setting. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security (CCS'21)* (pp. 1981-2003). New York: ACM. doi: 10.1145/3460120.3484814

[10] Dey, C., & Sarkar, S. (2023). A new distinguishing attack on reduced round ChaCha permutation. *Scientific Reports*, 13, article number 13958. doi: 10.1038/s41598-023-39849-1.

[11] Donenfeld, J. (2017). WireGuard: Next generation kernel network tunnel. NDSS 2020. In *Network and distributed system security symposium* (article number 4846ada1492f5d92198df154f48c3d54205657b). San Diego: NDSS. doi: 10.14722/ndss.2017.23160.

[12] Dworkin, M. (2016). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC.* Gaithersburg: NIST. doi: 10.6028/NIST.SP.800-38D.
[13] Gülmezoglu, B., Irazoqui, G., Eisenbarth, T., & Sunar, B. (2019). Cross-VM cache attacks on AES. *IEEE Transactions on MultiScale Computing Systems*, 2(3), 211-222. doi: 10.1109/TMSCS.2016.2550438.
[14] Kebande, V.R. (2023). Extended-ChaCha20 stream cipher with enhanced quarter round function. *IEEE Access*, 11, 114220-114237. doi: 10.1109/ACCESS.2023.3324612.
[15] Krasnov, V. (2016). *It takes two to ChaCha (Poly)*. Retrieved from https://blog.cloudflare.com/it-takes-two-to-chacha-poly.
[16] Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., & Josefsson, S. (2016). *ChaCha20-Poly1305 Cipher Suites for TLS (RFC 7905)*. Retrieved from https://www.rfc-editor.org/rfc/rfc7905.html.
[17] Maitra, S. (2016). Chosen IV cryptanalysis on reduced-round ChaCha and Salsa. *Discrete Applied Mathematics,* 208, 88-97. doi: 10.1016/j.dam.2016.02.020.
[18] Najm, Z., Jap, D., Jungk, B., Picek, S., & Bhasin, S. (2018). On comparing side-channel properties of AES and ChaCha20 on microcontrollers. In *IEEE Asia Pacific conference on circuits and systems (APCCAS)* (pp. 552-555). Chengdu: IEEE. doi: 10.1109/APCCAS.2018.8605653.
[19] Nir, Y., & Langley, A. (2015). *ChaCha20 and Poly1305 for IETF protocols (RFC 7539)*. Retrieved from https://www.rfc-editor.org/rfc/rfc7539.html.
[20] Pfau, J., Reuter, M., Harbaum, T., Hofmann, K., & Becker, K. (2019). A hardware perspective on the ChaCha ciphers: Scalable ChaCha8/12/20 implementations ranging from 476 slices to bitrates of 175 Gbit/s. In *32nd IEEE international system-on-chip conference (SOCC)* (pp. 294-299). Singapore: IEEE. doi: 10.1109/SOCC46988.2019.1570548289.
[21] Polubelova, M., Bhargavan, K., Protzenko, J., Beurdouche, B., Fromherz, A., Kulatova, N., & Zanella-Béguelin, S. (2020). HACLxN: Verified generic SIMD crypto (for all your favourite platforms). In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (CCS'20)* (pp. 899-918). New York: ACM. doi: 10.1145/3372297.3423352.
[22] Procter, G. (2014). *A security analysis of the composition of ChaCha20 and Poly1305*. Retrieved from https://eprint.iacr.org/2014/613.pdf.
[23] Rashidi, B. (2024). High-performance hardware structure of ChaCha20 stream cipher based on sparse parallel prefix adder. *International Journal of Circuit Theory and Applications*, 53(5), 2947-2957. doi: 10.1002/cta.4264.
[24] Rescorla, E. (2018). *The transport layer security (TLS) protocol version 1.3. (RFC 8446)*. Retrieved from https://www.rfc-editor.org/rfc/rfc8446.html.
[25] Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009). Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on computer and communications security (CCS 2009)* (pp. 199-212). Chicago: ACM. doi: 10.1145/1653662.1653687.
[26] Serrano, R., Duran, C., Sarmiento, M., Pham, C., & Hoang, T. (2022). ChaCha20-Poly1305 AEAD for transport layer security 1.3. *Cryptography*, 6(2), article number 30. doi: 10.3390/cryptography6020030.
[27] Sullivan, N. (2014). *Do the ChaCha: Better mobile performance with cryptography*. Retrieved from https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography.
[28] Tsoupidi, R.-M., Balliu, M., & Baudry, B. (2021). Vivienne: Relational verification of cryptographic implementations in WebAssembly (verifies ChaCha20/Poly1305 in WHACL*). *ArXiv*. doi: 10.48550/arXiv.2109.01386.
[29] Xu, Z., Xu, H., Tan, L., & Qi, W. (2024). Improved differential-linear cryptanalysis of reduced-round ChaCha permutation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2024(2), 166-189. doi: 10.46586/tosc.v2024.i2.166-189.
[30] Zinzindohoué, J.-K., Bhargavan, K., Protzenko, J., & Beurdouche, B. (2017). HACL*: A verified modern cryptographic library. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security (CCS'17)* (pp. 1789-1806). New York: ACM. doi: 10.1145/3133956.3134043.

ChaCha: розвиток та модифікація Salsa20 у сучасних криптографічних системах

Олексій Палій
Аспірант
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0009-0006-8387-3609

Олександр Дудник
Кандидат технічних наук, доцент
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0009-0005-3684-965X

Анотація. У статті проведено огляд потокового шифру ChaCha20 як спадкоємця алгоритму Salsa20, зосереджено увагу на його розвитку, технічних особливостях та застосуванні в сучасних криптосистемах. Актуальність роботи обумовлена широким впровадженням ChaCha20 у протоколи безпеки (TLS 1.3, VPN тощо) завдяки високій швидкодії в програмних реалізаціях та стійкості до криптоаналізу. Метою роботи було проаналізувати еволюцію ChaCha від Salsa20, порівняти його з іншими шифрами та узагальнити останні досягнення щодо модифікацій і продуктивності. У межах дослідження використовувались методи аналізу літературних джерел і експериментальних даних про швидкодію та стійкість шифрів. Основні результати включають висвітлення історії створення ChaCha на основі Salsa20 та покращення дифузії за раунд, детальний опис структури алгоритму (матриця стану 4×4, операції додавання-обертання-XOR) і його криптостійкості (відсутність практичних атак на повну 20-раундову версію). Показано переваги ChaCha20 над Advanced Encryption Standard (AES) в програмному середовищі – зокрема, на платформах без апаратного прискорення AES ChaCha20 працює до 3 разів швидше при еквівалентному рівні безпеки. Розглянуто впровадження ChaCha20-Poly1305 в TLS і WireGuard, а також використання XChaCha для подовжених «nonce» і алгоритму Adiantum для шифрування дисків на мобільних пристроях. Проаналізовано сучасні модифікації ChaCha (наприклад, збільшення кількості раундів) та їх вплив на продуктивність і безпеку. Практична цінність огляду полягає в узагальненні сучасного досвіду використання ChaCha20, що може бути корисним для вибору криптоалгоритмів у ресурсно-обмежених системах і подальших досліджень у галузі потокових шифрів

Ключові слова: потоковий шифр; Advanced Encryption Standard; TLS 1.3; криптоаналіз; ресурсно-обмежені системи