Performance Evaluation of Transport Layer Security in the 5G Core Control Plane

Oliver Zeidler
oliver.zeidler@tum.de
Technical University of Munich
Munich, Germany

Julian Sturm
Daniel Fraunholz
julian.sturm@zitis.bund.de
daniel.fraunholz@zitis.bund.de
ZITiS
Munich, Germany

Wolfgang Kellerer
wolfgang.kellerer@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

As 5G is currently being rolled out, security considerations for this critical infrastructure are getting more into focus. Hereby, the security investigation of the 5G core as the central element plays a pivotal role. The structure of the core is based on a Service-Based Architecture (SBA) consisting of Network Functions (NFs). These NFs communicate via REST/HTTP2 interfaces, that can be secured using Transport Layer Security (TLS) for encryption. However, this enhanced security is not enforced by standardization, but up to the system operator to decide. Therefore, in this work we derive recommendations on when to use TLS. For that, we investigate the overhead of TLS in a simulation, based on the open-source frameworks Open5GS and UERANSIM. To measure a user-relevant overhead, we look into 5G's UE registration and Packet Data Unit (PDU) session establishment procedures. By testing 14 of the most relevant cipher suites, our results show, that TLS adds no more than 1 % of time overhead in a running system. Further, we show cipher suites using ECDSA keys to be faster than the ones using RSA keys. Surprisingly, TLS 1.3 shows a larger performance overhead than its predecessor TLS 1.2. We demonstrate CPU and memory overhead of TLS to be insignificant in the context of the 5G core.

CCS CONCEPTS

• **Networks → Network measurement**; *Mobile and wireless security*; • **Security and privacy → Security protocols**; **Distributed systems security**; *Domain-specific security and privacy architectures.*

KEYWORDS

5G, TLS, Network Security, Performance Measurements

ACM Reference Format:
Oliver Zeidler, Julian Sturm, Daniel Fraunholz, and Wolfgang Kellerer. 2024. Performance Evaluation of Transport Layer Security in the 5G Core Control Plane. In *Proceedings of the 17th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '24), May 27–30, 2024, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3643833.3656140

1 INTRODUCTION

Wireless communication enables a multitude of communication infrastructures and business models and has become essential for our modern society. The current version, 5G, marks the fifth generation of mobile communication networks and is considered critical infrastructure next to water, heat, and electricity. Therefore, protecting this critical infrastructure and all its parts is vital [6, 10].

Compared to its predecessor, 4G, 5G provides not only larger bandwidths and higher connection speeds, but also introduces new features and architectural changes. New features, like network slicing and multi-access edge computing, open up new possibilities, but at the same time also new potential security risks. The virtualization, which enables network slicing, increases security and isolation between functionalities but offers new attack vectors as well [3]. Especially control systems of critical infrastructure can be valuable targets, increasing the severity of small misconfigurations, e.g., an ineffective firewall rule [21].

The largest architectural changes in 5G targeted the core, which became a small network of core network functions itself. The 5G Core (5GC) network is often physically secluded. However, a successful attack on the 5GC would have far more severe consequences, than an attack on a base station (gNB). For example, if a single gNB is attacked and out of service, all users in a certain, small area suffer a service outage. Since the core network connects a large number of gNBs, a service outage in the 5GC leads to users in a large area, typically on a national level, experiencing disruptions in network connectivity. In addition to such Denial-of-Service (DoS) attacks, the core network contains a large amount of security critical and privacy-sensitive data in form of secret keys, billing data and user identities. These need to be protected against interception and eavesdropping.

Therefore, mechanisms affecting the 5GC security need to be carefully investigated. The 5GC network consists of multiple Network Functions (NFs), which communicate with each other via REST/HTTP2 interfaces. NFs are exchanging messages to control user access, user mobility, quality of service management, and slicing, to name a few of the most important tasks of the 5GC [5]. According to the governing security standard by the 3rd Generation Partnership Project (3GPP) TS 33.501, these communications should be secured by adding Transport Layer Security (TLS) as a security layer [2].

Surprisingly, while network operators can use TLS in their 5GC setups, they are not required to do so by standardisation. Therefore, in this work we investigate the application of TLS and, in particular, measure its overhead on the 5GC communication performance. The

WiSec '24, May 27–30, 2024, Seoul, Republic of Korea

ACM ISBN 979-8-4007-0582-3/24/05.
https://doi.org/10.1145/3643833.3656140

overhead of TLS can be use-case-specific, as previous studies have shown for MQTT [22] and Session Initiation Protocol (SIP) [24]. Thus, measuring this overhead in a 5G-specific scenario is vital for deciding when to use TLS and which of its options. From these measurements, we derive recommendations for the use of TLS in the 5GC, to provide the most security with the least performance overhead.

In detail, the contributions of this work are:

(C1) We quantify the user-relevant overhead introduced by TLS during User Equipment (UE) registration and a consecutive Packet Data Unit (PDU) session establishment procedure. This is the time an end-user has to wait before they can send or receive data.

(C2) We quantify the memory and average CPU utilisations of the 5GC during the same procedures.

(C3) We quantify performance differences between networks using plain-text connections and TLS versions 1.2 and 1.3.

(C4) We compare 14 cipher suites in total. Eight for TLS 1.2 and six for 1.3, allowing an in-depth comparison between TLS 1.2 and 1.3.

(C5) We measure and analyse the expected performance overhead of TLS between the worst case scenario of a freshly restarted system and in a running system.

The remainder of this work is structured as follows: Section 2 gives an overview of the 5GC while Section 3 introduces TLS. Section 4 details the experimental setup and Section 5 analyses the measurement results. Finally, Section 6 discusses related work and Section 7 concludes this work.

2 OVERVIEW OF 5G SYSTEM AND CORE

A 5G network consists of three domains, the User Equipment (UE), the Radio Access Network (RAN), and the 5G Core (5GC). The UE is the user endpoint, e.g. a smartphone, or any device relying on cellular connectivity. It is connected to the RAN via the gNB as a radio base station. The RAN is connected to the 5GC, which establishes a connection to a Data Network (DN), for example, the internet. The communication between UE, RAN and 5GC uses different sets of protocols to exchange information and can be divided into three groups based on their purpose: User plane (u-plane) communication for user data, control plane (c-plane) for signalling, and management plane (m-plane) for management of components.

In this work, we focus on the c-plane, since the majority of 5GC components communicate on the c-plane.

2.1 5G Core C-plane Architecture and Protocols

5G introduced the Service Based Architecture (SBA) for the c-plane in the core. The SBA is based on REST/HTTP2 and allows components to interact with each other based on standardized interfaces and protocols well-known from internet applications. This allows for modular and dynamic procedures, enabling better scalability and geographic redundancies. The NFs have dedicated areas of responsibility following the microservice architecture model. [5].

According to 3GPP TS 23.501, there are different modes of SBAs in the core [1]. Out of these, we use direct communications with the Network Repository Function (NRF) in this work. Therefore, each NF has to register at a central service registry called the NRF, which acts as a database of all running NFs. Additionally, each time a NF initially wants to interact with another NF, the NRF has to be queried to discover the connection details of the other NF. Afterwards, the NFs can directly communicate with each other. Consequently, we do not use the Service Communication Proxy (SCP), which would act as a central communication hub between the NFs. In that case, the NFs never directly communicate with each other, but with the SCP instead.

2.2 5G Core Security Protocols and Configuration

The 5GC leverages multiple protocols to ensure authentication and authorization across NFs. This is necessary as the SBA enables rapid provisioning and de-provisioning of individual NFs, requiring the security protocols to adapt. Authentication is the process of validating the identity of a NF, while authorization means validating if a given NF is allowed to execute a certain action.

To ensure authentication between two NFs, 5G relies on the TLS protocol.

That means, without activated TLS there is no inter-NF authentication. To verify a connected peer is authentic, the NFs rely on the proven possession of a certificate and the corresponding private key signed by the mutually trusted Certificate Authority (CA). In addition, 5G employs the OAuth2.0 protocols for authorization to enable the NRF to limit the actions each NF can trigger. Since these tokens need to be transmitted, an attacker is able to impersonate individual NFs if they are intercepted and not protected by TLS or other means.

TLS has seen widespread adoption for web-based services and is therefore well understood. In practice, there are two modes. For most web-based uses, only one party is authenticated, as the client wants to ensure it is connected to the right server and not a malicious party. The 5GC relies on a mode referred to as Mutual TLS (mTLS), where both server and client NFs mutually authenticate each other. The TLS identification is based on certificates, which are digital objects containing a public key and a set of parameters. The authentication of a NF can now be done by either checking the certificate's hash against known values or, more commonly, validating against a chain of trust. For this reason, one or more entities are accepted as trust anchors, in the context of TLS named CAs. These can then sign certificates to indicate that anyone trusting the CA can also trust this certificate. For security reasons, the CAs do rarely use their primary key pair for signatures, but use a second key, signed by their primary key pair. These certificates based on the secondary keys are called intermediate certificates. To verify if a communication partner is trustworthy, the chain of certificates needs to be traced back to one of the trusted CAs.

The authorization in the 5G core is enforced by the NRF using the OAuth 2.0 protocol. Before a NF can contact another NF, it has to request authorization from the NRF, which provides a signed JSON Web Token (JWT) enabling communication [2]. Since this mechanism is not implemented in most open source 5G networks, we are unable to quantify the impact of this on the connections, but expect it to be minimal, as any NF should always maintain a TLS tunnel to the NRF.

2.3 UE Registration Procedure

To investigate the communication overhead of TLS on the UE side, Figure 1 shows an exemplary UE registration procedure as a signal flow diagram. The UE registration request, marked with blue-solid lines, is the first sub-procedure that is necessary for a user to connect to the network. It involves the UE, the gNB, and multiple NFs in the 5GC. At the start, the UE establishes a radio connection with the gNB, and a logical connection directly with the Access and Mobility Management Function (AMF). The AMF handles user registrations, de-registrations, and user mobility, while including other NFs when their services are needed. In the beginning, the AMF requires the UE to send its identity. This identity sub-procedure is marked in black in Figure 1. Afterwards, the AMF forwards the identity to the Authentication Server Function (AUSF), which is used for authentication and authorisation of UEs. The AUSF queries the Unified Data Management (UDM), which manages the stored UE data in the 5GC. The database itself is another NF called Unified Data Repository (UDR), which only communicates with the UDM. To keep the figure simpler, the UDR is not shown. The AMF forwards the authentication challenge from the AUSF to the UE which replies using an answer based on the key stored on the SIM card. The AMF then informs the AUSF of the authentication result. The AUSF provides the keys required for security establishment to the relevant components. Afterwards, the Authentication and Key Agreement (AKA) procedure, shown in Figure 1 as red-dashed lines, is concluded. Using the key material, encryption and integrity protection is established between the UE and the AMF (logical link), as well as between the UE and the gNB. This part of the registration is called the Security-Mode Command (SMC) sub-procedure and is marked as green-dashed. Afterwards, the AMF interacts with the UDM, Policy Control Function (PCF), and Session Management Function (SMF) to enable data routing and charging functionalities. The PCF controls the policies of the network, such that each user gets the connection parameters they subscribed to. The SMF handles session management and configures the User Plane Function (UPF), which is responsible for handling u-plane traffic. Figure 1 displays this u-plane configuration as two orange-dashed lines. After the UPF is configured, the AMF concludes the registration procedure by confirming its success to UE, UDM, and PCF.

As mentioned before, each NF has to register at the NRF and query it for discovery information of the other NFs. Therefore, after a system reboot, a noticeable overhead is expected for the first UE registration procedure. This is especially true, if TLS is activated since each of these initial connections needs an additional TLS handshake.

2.4 PDU Session Establishment Procedure

Figure 2 shows an exemplary Packet Data Unit (PDU) session establishment procedure as a signal flow diagram. After the UE is registered at the core network, it can send a PDU session establishment request. This starts the session establishment procedure, which includes AMF, SMF, UPF, PCF, and UDM. The initial request is sent by the UE to the AMF, which extends it to the SMF. The SMF retrieves relevant UE information from the UDM. After obtaining the data, the SMF updates the AMF with a response. This PDU session authentication and authorization is marked with green-dotted lines in Figure 2. The orange-dashed lines mark the policy association establishment, which involves the PCF. Then, the SMF configures the UPF and thereby establishes a session for the UE. Figure 2 displays the session establishment as black-solid lines. After the session is established, the SMF informs the UE via AMF and gNB. As soon as the PDU session is successfully established, the UE can send and receive u-plane data from the DN, e.g. the internet. The u-plane traffic physically passes through the gNB and is forwarded to the DN by the UPF. Figure 2 marks this u-plane data transfer in blue.

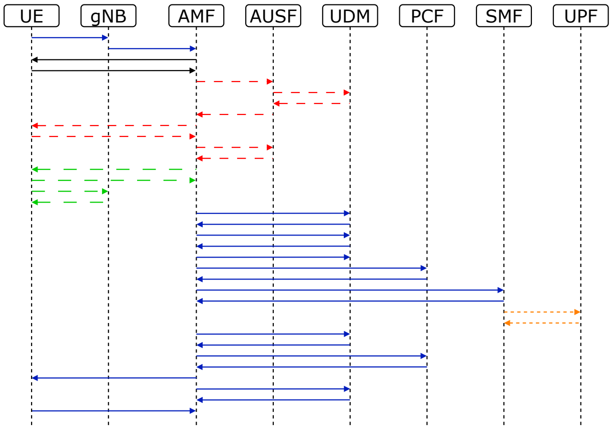


Figure 1: UE registration procedure in 5G. Blue solid: Registration request, black solid: identity subprocedure, red-dashed: AKA subprocedure, green-dashed: SMC subprocedure, orange-dashed: u-plane configuration.

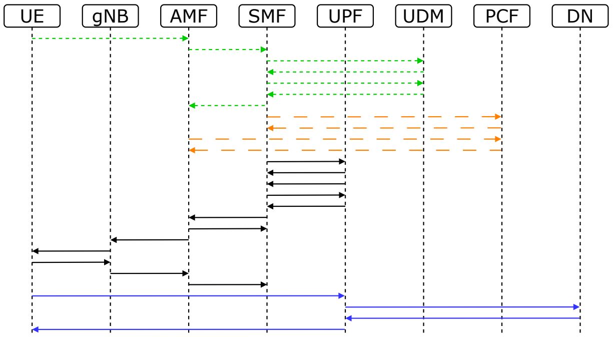
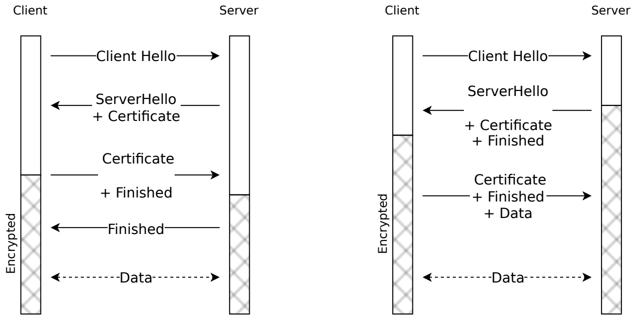


Figure 2: PDU session establishment procedure in 5G. Green-dotted: PDU session authentication, orange-dashed: policy association establishment, black-solid: PDU session establishment, blue-solid: data transfer.

3 TRANSPORT SECURITY IN THE 5G CORE

In TS 33.501 the 3GPP states, that "TLS shall be used for transport protection [between NFs] within a PLMN [Public Land Mobile Network] unless network security is provided by other means". This is detailed further: "In case interfaces are trusted (e.g. physically protected), there is no need to use cryptographic protection" [2]. For small networks, this requirement could be satisfied by using



(a) TLS 1.2

(b) TLS 1.3

Figure 3: Simplified handshake procedures for both TLS versions.

dedicated hardware switches and packet filters. In larger, more dynamic and roaming-enabled networks, this is not easily achievable. One solution might be the usage of a Virtual Private Network (VPN) based on IPsec. While IPSec could in theory provide secure links between a pair of NFs, the configuration overhead is significant. The performance of IPsec has been shown to be comparable to TLS [15] when using identical ciphers. In IPsec each pair of NFs (e.g. AMF and NRF) has to be configured separately, resulting in an exponentially increasing number of point-to-point links. This is not the case with TLS due to the usage of certificates. In larger, carrier-sized networks, frequently commissioned and decommissioned components are expected, making the point-to-point configuration impractical. In contrast, TLS is used throughout the web for more than 80% of websites, which required secure certificate management protocols such as the Automatic Certificate Management Environment (ACME). Let's Encrypt serves as CA for a large part of websites and managed to provide three million certificates per day fully automatic through ACME in 2023 [11]. We therefore excluded IPSec from consideration to focus on the more suitable TLS based solution, which therefore is expected to be more prevalent in real-world networks.

Two main factors of TLS potentially impact the overall network performance. At first, the used TLS version might impact performance, as TLS 1.3 changed the overall protocol for connection establishment. TLS 1.3 introduced many changes and remedied some shortcomings of TLS 1.2. This includes reducing the number of available cipher suites from over 90 in TLS 1.2 down to five in TLS 1.3 and removing all algorithms deemed unsafe. Partly, this reduction is based on a decoupling of key exchange and authentication algorithms from the rest of the cipher suite, removing duplicates. Furthermore, the handshake was streamlined, lowering the number of required messages and allowing data to be transmitted before the handshake is completed. At the same time, any data is encrypted as soon as the symmetric key is established between server and client , to prevent leaks from metadata such as certificates or extension values. Figures 3a and 3b show a simplified version of the TLS handshake procedures. Additionally, TLS 1.3 implements a downgrade protection, so that even if the handshake is intercepted and modified, two TLS 1.3 endpoints will not fall back to earlier TLS versions. This prevents an attacker from leveraging flaws or weaker cipher configurations present in previous TLS versions and ensures that such attempts can be detected. It is based on a bit-sequence in the handshake-nonce, which itself is protected though the key derivation. When this sequence is seen by a pair of TLS 1.3 endpoints, they will abort any attempt to establish a session with an older version.

Table 1: Structure of cipher suite names based on an example.

Noun	Meaning	1.2	1.3
TLS	Keyword	✓	✓
ECDHE	Key exchange algorithm	✓	✗
ECDSA	Authentication algorithm	✓	✗
WITH	Keyword	✓	✗
AES 128 GCM	Cipher algorithm	✓	✓
SHA256	Message authentication algorithm	✓	✓

As a second factor, we consider the cipher suite. Table 1 explains how the identifiers are structured between TLS 1.2 and TLS 1.3. Generally, a TLS 1.2 cipher suite defines a number of different algorithms for different use cases. They start with 'TLS' and contain (from left to right) the key exchange algorithm, the authentication algorithm, the keyword 'WITH', the symmetric cipher, and the message authentication algorithm. For TLS 1.3, the structure is similar, but the key exchange and authentication algorithms are omitted, as they are negotiated independently. Our selected cipher suites all use Elliptic-curve Diffie-Hellman with Ephemeral keys (ECDHE) as the key exchange algorithm and either Rivest-Shamir-Adleman (RSA) or Elliptic Curve Digital Signature Algorithm (ECDSA) for authentication. Additionally, we use Advanced Encryption Standard (AES) or ChaCha20 as the cipher algorithm and Secure Hash Algorithm (SHA) with different digest sizes for message authentication. Cipher algorithms are symmetric encryption algorithms that are able to effectively and efficiently encrypt large amounts of data. In contrast to asymmetric algorithms, each party needs the same key, which is used for both encryption and decryption. The cipher algorithms also specify the modes used for integrity protection. In the case of ChaCha20, this is Poly1305, in the case of AES, both the Galois/Counter Mode (GCM) and Counter with Cipher block chaining Message authentication code (CCM) mode are permissible. More information on these crypto primitives can be found in RFC 8446 and the referenced standards. Both RSA and ECDSA can be used for authentication through asymmetric cryptography. Each party stores their public key in their respective certificate. After the authenticity of the certificate has been proven through the chain of trust, the authenticity of the public keys has been proven. Everything signed with the corresponding private key can now also be accepted as authentic. While both algorithms work this way, one big difference is the required length of the public keys. Since ECDSA relies on elliptic curves, a 256-bit key is considered equally secure to a 3072-bit RSA key (German Federal Office for Information Security (BSI) TR-02102-1). This reduces the amount of data required to transmit the certificates by about 90%. In the following, we differentiate between ECDSA and RSA solely based on their encryption keys.

Other mechanisms, that could impact performance in real-world deployments are explicitly excluded from this study.

Especially with RSA based suites, TLS handshake messages can become big enough to require multiple frames to transmit. This packet fragmentation can lead to a higher latency, especially in unstable network conditions. Since we assume the 5G core components to run in a cloud environment or dedicated datacenter with carefully chosen Maximum Transmission Unit (MTU) values, we chose to exclude this mechanism, by ensuring no fragmentation occurs in our test.

In order for a client to verify a certificate has not been invalidated by issues, mechanisms for certificate revocation such as Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP) exist. These add additional overhead and are primarily designed for systems where the client and server are owned by different entities, where simpler mechanisms such as checking the certificates serial number cannot be implemented. We therefore decided not to implement any revocation check in our tests.

4 EXPERIMENTAL SETUP

This section provides an overview of the used hardware and software, as well as an explanation of the investigated set of procedures. We chose a completely softwarised setup with the complete system running on one Virtual Machine (VM). This setup is reasonable, as one of the goals of introducing Network Function Virtualisation (NFV) in the core was the ability to move away from hardware appliances towards more scalable cloud environments. While we measure the end-to-end delay from the UE's perspective, our focus lies on the 5G core network. Therefore, we avoid using a dedicated UE and a wireless channel, as this would introduce additional noise to our intended measurements.

4.1 Testbed

In our setup, we use Open 5G System (Open5GS) 2.6.4 [17] as a core network and UE RAN Simulator (UERANSIM) 3.2.6 [12], emulating UE and gNB, running on a single machine in parallel. These are both well-known open-source systems providing a base for our testbed. To not interfere with the measurement times, the log level in UERANSIM is set to "error" to reduce the time for disk writes. Additionally, the measured times are based on the high-resolution system clock and measured in microseconds (μs). This enables more precise measurements compared to the event-based approach of UERANSIMs timers. UERANSIM is modified to support our time measurements. The timer starts when the UE sends the registration request and stops once the UE receives the confirmation of the successful PDU session establishment.

The Open5GS system was adapted as well for measurements. Firstly, the configuration file of each NF is modified to support TLS. To evaluate different TLS options correctly, both client and server verify the certificates of each other. Like in UERANSIM, the log level is set to "error". Secondly, the files containing the client and server configurations were modified. For each test, the client only offers the exact TLS version and cipher suite in the ClientHello extensions, that is tested in the respective scenario. Additionally, we fixed the curve for Elliptic Curve (EC) based algorithms to "P-256".

Open5GS leverages the widely used OpenSSL as its cryptography library [23] and initially uses RSA keys with a length of 2048 bit. To test the cipher suites based on ECDSA, we created ECDSA keys with a length of 256 bit. To support certificate verification, we acted as a CA and signed the newly created certificates as explained in Section 2.2. We also created new RSA certificates with 2048 bit to avoid any unintended differences, that could interfere with our evaluation. We kept this value, since it is the most commonly used RSA configuration for HTTPS, accounting for over 90% of surveyed web hosts in 2021 [26].

For additional information, we also track the average CPU utilisation of the Open5GS NFs. The average CPU utilisation provides the percentage of time the observed process was occupying the CPU over the time the process was running. Thus, it is a time-averaged CPU value. Due to our sampling method, the linux utility ps, this data is tracked with 0.1 granularity [16].

The machine used for the measurements is a VM running Ubuntu 22.04.3 LTS, with 16 CPU cores of an Intel i9-10900k running at 3.70 GHz and 24 GB of RAM. Modern processors, as our chosen i9-10900k, are often equipped with hardware-based accelerators for AES computation. The corresponding CPU instruction set for x86-64 is called AES-NI and was first proposed in 2008. Since we assume the core network runs on commodity server hardware, all our tests were performed with AES-NI enabled.

4.2 Scenario

The UE registration and PDU session establishment procedures as described in Sections 2.3 and 2.4 were chosen as a scenario, to measure the overhead of TLS for users to evaluate the performance impact of the security protocol. We chose only these procedures, since they appear the most frequent in a 5G system. We measure the time between the registration request of the UE and the finished establishment of the PDU-session. After this point, the user is able to send and receive data traffic. Therefore, this is the time a user has to wait after switching on their UE until they are connected to the internet, or generally a DN.

We distinguish between a freshly started system as a worst-case scenario, called cold start, and a warm start scenario representing a running system. In each scenario, one UE is connecting to the core network. To have a clear separation between cold and warm start, we define two UEs, one exclusively for each scenario. In addition to the time, we measure the average CPU utilisation for cold and warm start. This allows us to investigate the influence of different TLS options on the core network itself. In detail, we collect the CPU utilisation for the cold start after the cold start UE is started and therefore include the complete process of the 5GC starting up. The warm start measurement is done after the warm start UE has been started. The CPU utilisation is an averaged value over the processes' lifetime, resulting in the warm start overhead always including the already measured cold start values. Therefore, we are restricted to a qualitative analysis, which nonetheless provides interesting trends.

4.3 Methodology

Both the 3GPP as well as national standardization bodies such as the United States National Institute of Standards and Technology (NIST) or the BSI publish guidelines on configurations for both TLS 1.2 and TLS 1.3. These guidelines can be seen as industry best practices and can even be legally required for larger telecommunication companies, that are considered critical infrastructure in their jurisdiction. They also mitigate possible security flaws in TLS 1.2 by restricting weak configurations. We therefore base our selection of cryptography algorithms on these guidelines.

The tested cipher suites and key exchange algorithms were selected as follows:

(1) All standardized ciphers were selected
(2) Algorithms not adhering to 3GPP TS 33.310 Annex E were excluded
(3) Algorithms not adhering to NIST Special Publication 800-52 (US national authority) were removed
(4) Algorithms not adhering to BSI TR-02102-2 (German national authority) were eliminated
(5) Recommendations of the three standardization entities filter the remaining algorithms.

Each cipher suite is tested in 1250 consecutive test runs with one cold and one warm start each. For the cold start, Open5GS and UERANSIM both are completely restarted. Only the MongoDB, which is the base for the UDR to run, is not reset and always contains 200 entries of UEs. Each iteration measures one cold start, followed by one measurement of a warm start in the already primed network. Afterwards, for the following measurement pair, the system is reset. After 1250 runs, when the cipher suite and/or the TLS version changes, Open5GS is also rebuilt to account for the change. Each of these test runs results in a time measurement in μs, as well as an average CPU utilisation in % of one core. These values provide sufficient information about the TLS performance, because the different test cases can be compared to baseline measurements without enabled TLS and with each other.

In detail, we measure the time overhead as a percentage by dividing a result by the baseline measurement result without TLS. So, for example the overhead of TLS 1.2 in a cold start scenario is calculated as shown in Equation (1).

$$Overhead = \frac{TLS1.2_{cold}}{noTLS_{cold}} - 1 = \frac{TLS1.2_{cold} - noTLS_{cold}}{noTLS_{cold}} \quad (1)$$

To measure the memory overhead of TLS in the 5GC, we investigate the number of TLS handshakes in our use-case. We only consider the handshakes needed for our two investigated procedures. Each of the following NFs has to register at the NRF and can use TLS: AMF, AUSF, PCF, UDM, UDR, SMF. The AMF needs to discover the AUSF, the UDM, the PCF, and the SMF and establish a connection after a TLS handshake. The AUSF also handshakes with the UDM, which has to handshake with the UDR once. For the PDU session establishment, one additional handshake between SMF and UDM is needed. In total, this equals 13 TLS handshakes which are needed for a freshly booted system.

5 EVALUATION

Table 2 shows the 15 different test cases (including the baseline), for which the 1250 cold and warm starts were performed each. The test with deactivated TLS acts as a baseline. Overall, there are 8 test cases with TLS 1.2, 6 with TLS 1.3., out of which 8 are with ECDSA, and 6 with RSA. All calculations are based on the mean values, which are marked as white circles in the respective violin plots. The black line marks the median, which acts as additional information about the distribution shown in the figures. All plots display data between the percentiles 2 and 98.

Table 2: Overview of performed test cases.

TLS	Cipher Suite	Key
None	None	None
1.2	TLS_ECDHE_ECDSA_WITH_AES_128_CCM	ECDSA
1.2	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDSA
1.2	TLS_ECDHE_ECDSA_WITH_AES_256_CCM	ECDSA
1.2	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDSA
1.2	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	ECDSA
1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	RSA
1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	RSA
1.2	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	RSA
1.3	TLS_AES_128_GCM_SHA256	RSA
1.3	TLS_AES_256_GCM_SHA384	RSA
1.3	TLS_CHACHA20_POLY1305_SHA256	RSA
1.3	TLS_AES_128_GCM_SHA256	ECDSA
1.3	TLS_AES_256_GCM_SHA384	ECDSA
1.3	TLS_CHACHA20_POLY1305_SHA256	ECDSA

5.1 TLS Versions

Figure 4 shows the performance difference between the different TLS versions (TLS 1.2 and TLS 1.3) and deactivated TLS (No TLS) for the UE registration procedure. Since AES-CCM was only tested within TLS 1.2 and ECDSA, it is not considered in this plot, as it would skew the results. As expected, TLS adds an overhead. This is especially noticeable for the cold start, while the performance overhead in a warm start scenario is significantly smaller. Comparing TLS in the cold start scenario, TLS 1.2 adds a mean overhead of 29.06 %, while the overhead of TLS 1.3 is slightly higher at 33.58 %. For the warm start, TLS 1.2 is 0.64 % slower than No TLS, while TLS 1.3 is again a bit worse with an overhead of 0.83 %.

Looking at the case without TLS in detail, the cold start adds 2.77 % of time over the warm start. This is because each NF has to register at the NRF and query the connection details of its peers. Therefore, starting the system and initializing the connections between the NFs adds an overhead, even without TLS. The larger overhead of TLS versions 1.2 and 1.3 for the cold start is caused by the TLS handshake in the communication initialization between the NFs in the 5GC. The added overhead by using TLS and its security features was expected. What is surprising, however, is the fact that TLS 1.2 adds less overhead than the newer version TLS 1.3, which is expected to be more efficient. This likely stems from the fact, that in TLS 1.3 the certificates are encrypted. Contrary to that, the fact, that TLS 1.3 allows application data to be sent earlier should improve the overhead. Since TLS 1.2 is still faster, we conclude, that this performance improvement has less impact than the additional performance overhead added by the certificate and header encryption in TLS 1.3.

5.2 Authentication algorithms and keys

Figure 5 displays the difference between using the encryption keys of the authentication algorithms ECDSA and RSA. The case without TLS does not have an encryption key and serves as a baseline comparison. Again, AES-CCM is not considered in this plot, as it would skew the results. The violins labeled ECDSA and RSA consist of both TLS versions 1.2 and 1.3. Compared to No TLS, TLS with RSA adds 34.63 % performance overhead on a cold start and 0.78 % on a warm start. ECDSA increases the time needed for registration

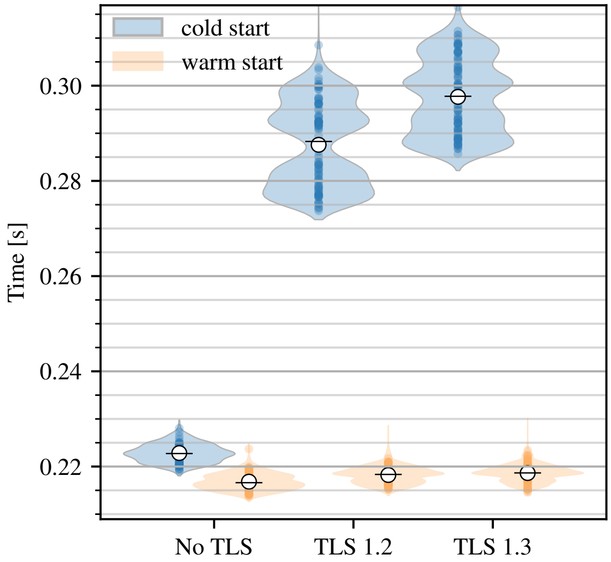


Figure 4: UE registration time for different TLS versions.

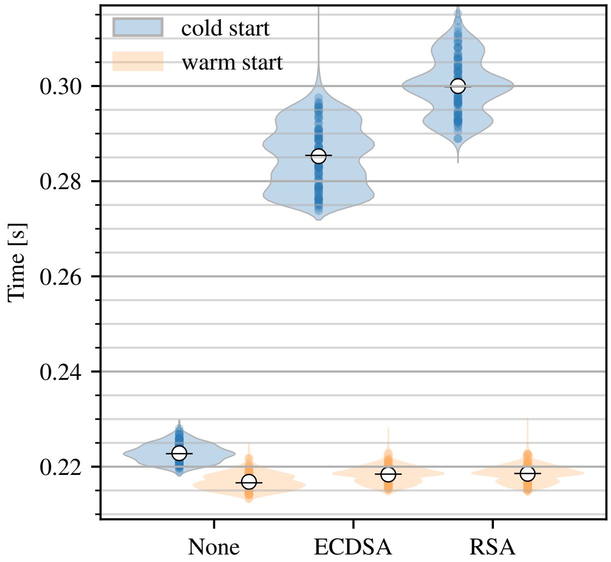


Figure 5: UE registration time for different authentication algorithms.

and session establishment on a cold start by 28.01 % and by 0.70 % on a warm start. In other words, ECDSA outperforms RSA in both start scenarios.

Since RSA uses a 2048 bit key, and the used ECDSA key is only 256 bit long, the increased performance overhead is expected. As mentioned in Section 3, the provided security by a 256 bit ECDSA key is roughly equal to a 3072 bit RSA key. Figure 5 shows, that ECDSA already outperforms RSA with a key-lenght of 2048 bit. Increasing the RSA key length, to achieve the same security, would further increase the performance overhead. In conclusion, ECDSA provides both a better performance and higher security.

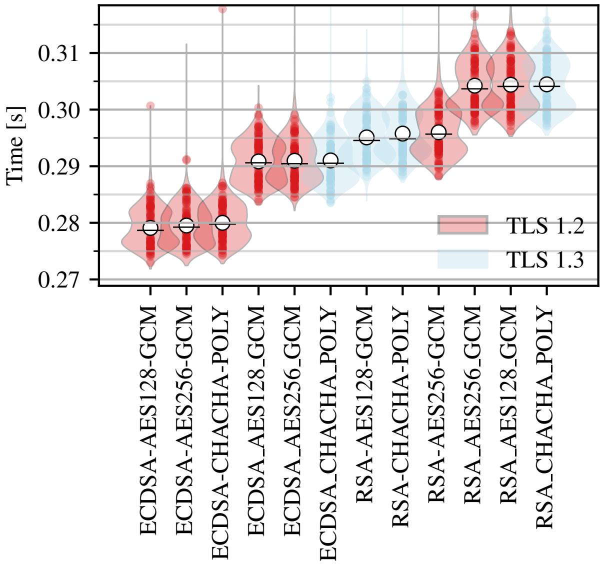


Figure 6: Comparison of the cipher suites for UE registration time in a cold start scenario.

5.3 Cold Start Scenario

The following discusses the cold start scenario in more detail. Figure 6 displays all cipher suites, that were tested for TLS 1.2 and TLS 1.3 for the cold start scenario, sorted by performance overhead. To provide a clearer overview, Figure 6 does not provide the complete name of the shown cipher suites, but only the information necessary to differentiate them, therefore omitting the terms TLS, ECDHE and SHA. These are either redundant or clearly linked to a specific cipher suite in terms of our selected cipher suites shown in Table 2.

Figure 6 shows one clear distinction between the cipher suites using ECDSA and RSA keys. This was anticipated due to the findings of Section 5.2. However, Figure 6 shows, that the used authentication algorithm with its key has the most performance impact in the cold start scenario. Within ECDSA, there is also a trend from cipher suites using TLS 1.2 to ones using TLS 1.3, performing noticeably worse. Again, this is an expected trend, as shown in Section 5.1. For RSA, also a clear difference between TLS 1.2 and 1.3 cipher suites can be seen.

Regarding the cipher algorithms ChaCha-Poly, AES-CCM, and AES-GCM, Figure 6 shows some trends. While the main differentiating factors between the cipher suites are the used TLS version and authentication key, within each of these groups, AES-GCM is the best-performing cipher algorithm. The most notable difference exists within TLS 1.2 and ECDSA, where AES-GCM is followed by ChaCha-Poly, while AES-CCM performs the worst. The larger performance difference between the two AES variants can be explained by AES-CCM relying on an additional AES execution for the integrity code, compared to AES-GCM using the GHASH procedure. [27] ChaCha-Poly performs worse than AES-GCM, because of the used AES-accelerator. Overall, the mean performance overhead of AES-GCM is, with one exception, always smaller than that of ChaCha-Poly.

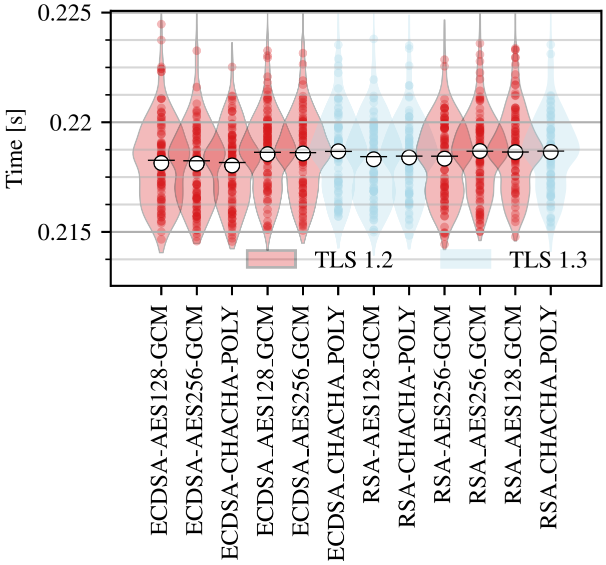


Figure 7: Comparison of the cipher suites for UE registration time in a warm start scenario.

To investigate statistical significance, we performed t-tests with 100.000 permutations and a significance level of $p = 0.0001$. These show both the difference between TLS 1.2 and 1.3, as well as the difference between ECDSA and RSA to be significant. Therefore, we strongly recommend ECDSA since it provides the best performance in the cold start scenario. Performance-wise, TLS 1.2 is preferred over version 1.3. The overall differences between the cipher algorithms however are not significant. Even so, when looking only at TLS 1.2 and ECDSA, AES-GCM performs significantly better than ChaCha-Poly and both are significant compared to AES-CCM. Although not being overall significantly better, we recommend AES-GCM as it is the fastest option. Additionally, the second best option, ChaCha-Poly, is not yet recognised by many standardisation organisations, making AES-GCM the more compatible option. In conclusion, our recommendations lead to the TLS 1.2 cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.

5.4 Warm Start Scenario

As discussed before, we assume a running system for a large majority of time. Therefore, Figure 7 displays all cipher suites again, but this time for the warm start scenario. As in Figure 6, the cipher names are reduced to the information needed to differentiate them. However, the ciphers are not sorted by performance, but in the same manner as in Figure 6. Compared to Figure 6, Figure 7 shows less variance between the violins. Interestingly, the t-tests still show the differences between the TLS versions and between ECDSA and RSA to be significant. Even if they only appear as marginal differences in the plot, considering many users registering to a running system, a small difference can lead to a larger performance impact. Before a final recommendation, Sections 5.5 and 5.6 investigate the options CPU and memory utilisations.

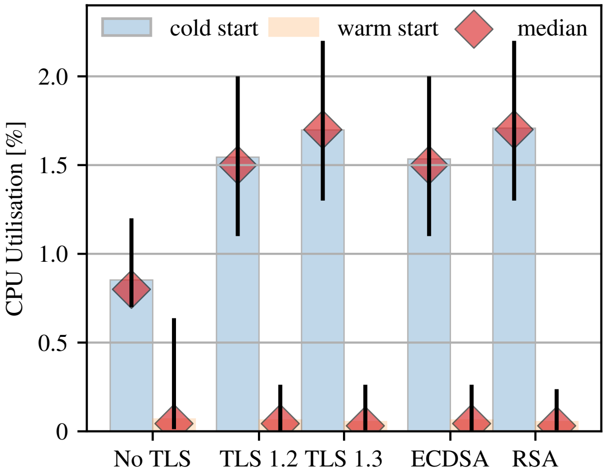


Figure 8: Average CPU utilisation for different TLS versions and keys. The bars for TLS 1.2 and 1.3 contain all ECDSA and RSA data and vice versa.

5.5 Average CPU Utilisation

To investigate the impact of the two TLS versions on the 5G Core, Figure 8 shows the average CPU utilisation as defined in Sections 4.1 and 4.2. In Figures 8 and 9, the bars display the mean values and the vertical black line represents the spread between the 2nd and 98th percentile of the data.

In the bar plot, we observe a similar trend as for the previously discussed time values: The highest difference lies between cold and warm start, while ECDSA demands less CPU power than RSA. The CPU load for a cold start is much higher, because of the connection initialisations and TLS handshakes. TLS 1.2 needs 0.69 % more CPU resources than no TLS, which is topped by TLS 1.3 needing 0.85 % more in a cold start scenario. Similarly, ECDSA outperforms RSA with an overhead of 0.68 % in a cold start scenario, while for RSA, the overhead is 0.86 %. The additional load on the system is highest in the cold start during the TLS handshakes, while the encryption only leads to a much smaller overhead in a running system. In the warm start, the CPU utilisation ranges around 0.1 %, which is our measured granularity. This limits our observations to the fact, that TLS adds less than 0.1 % of CPU overhead.

Regarding significance, both the difference between RSA and ECDSA and the two TLS versions are significant in the cold start case. The cipher algorithms are not shown in Figure 8, as their difference is insignificant. Since time and CPU utilisation show the same trends, we can conclude, that the increase in UE registration and PDU session establishment times depend directly on the load of the NFs in the 5GC.

To investigate, how the CPU utilisation overhead scales for a varying number of UEs in a running system, we performed an additional scaling test. Due to our 0.1 % measurement granularity, we used a system with a less powerful CPU for a more detailed investigation. However, it is still a qualitative analysis only differentiating between no TLS and one cipher suite, that being TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256. Figure 9 shows the warm start CPU utilisation for 0, 1, 5, 10 and 100 UEs

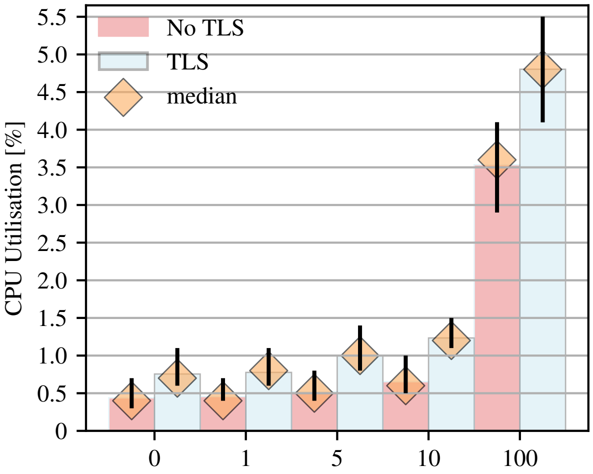


Figure 9: Average warm start CPU utilisation for an increasing number of UEs. The plot displays No TLS and TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.

that are started simultaneously. Looking at the case without any UE, we can derive the pure system overhead of TLS. In our test, this was 0.343 % of one CPU core, which is almost negligible, especially regarding much more powerful CPUs in real-world deployments. The average CPU utilisation related to one UE ranges between 0.018 and 0.032 % for a system without TLS and between 0.028 and 0.053 % for our chosen TLS cipher suite. This leads to a difference of 0.005 to 0.035 % per UE for TLS over no TLS. Our data even shows a trend, that this overhead shrinks for larger numbers of UEs. This is observed on a less powerful common-use CPU, and we consider a network operator to have much more powerful hardware in their 5G core. Additionally, in our setup, the whole core runs on one system. Considering one server per NF, the overhead per server reduces even more. Therefore, we can conclude, that the running 5GC system can handle a similar amount of users with activated TLS encryption compared to no encryption at all.

5.6 Network and Memory Analysis

To further analyse the overhead of TLS, we differentiate between network overhead and memory overhead. The network overhead consists both of additional packets in a TLS handshake and a per-packet overhead in the running system. The memory overhead relates to the certificates of every two connection parties to be stored in both NFs.

For TLS 1.2, the handshake adds four messages to a typical connection establishment, while TLS 1.3 adds only three. According to [18], a TLS handshake typically adds 4-7 kB of overhead to the system. Investigating the network traffic in our setup, we can support this result. In detail, we measured handshake overheads of 3666 to 5725 Byte. RSA-based cipher suites need 1955 Byte more for a handshake than ECDSA. This difference stems from the size of the used keys and the fact, that with certificate chains, multiple keys must be transmitted. TLS 1.3 needs one message less than version 1.2, but needs 104 Byte more nonetheless. However, due to the fact, that TLS 1.3 encrypts the data sooner, that can be explained, since a 16 Byte Message Authentication Code (MAC) is added to each message. Therefore, the handshake for TLS 1.3 is more memory-expensive than for version 1.2, which is caused by its added security.

In a running system, the overhead of TLS consists of the TLS record protocol header and the added MAC. Another potential cause for an overhead, the padding induced by the cipher algorithms block size, can be excluded here, since our chosen cipher suites are all operated in stream modes. The TLS record protocol layer header consists of 5 unencrypted and 4 encrypted Byte, totaling in 9 Byte. The MAC adds 16 Byte on top, resulting in an overall TLS overhead of 25 Byte per packet. [18] In our own measurements, we saw an overhead between 22 Byte for TLS 1.3 and 29 Byte for TLS 1.2. The remaining packet headers always add an overhead of 66 Byte, divided into the Ethernet header with 14 Byte, IP with 20 Byte, and TCP with 32 Byte. So, in comparison, TLS adds less overhead than the remaining protocols of the IP-stack.

Regarding memory consumption, according to [4, 18, 28], the amount of needed memory per connection ranges between 10 and 840 kB. Therefore, depending on the used TLS library and configuration, the overhead is always below 1 MB and for optimised libraries even less than 100 kB. For the 13 TLS handshakes in our scenario, we have 26 communication partners. Thus, the overall TLS memory consumption remains below 26 MB in all cases, which is almost negligible for modern hardware.

5.7 Summary of Performance Overhead

Table 3 shows the performance overhead of all measured factors for both cold and warm start, except for AES-CCM, due to its strong bias regarding TLS 1.2 and ECDSA. The system running without TLS acts as a baseline for all given percentages for cold and warm start respectively. For the cold start scenario, the time overheads range between 28.01 and 34.63 %. The largest impact on performance comes from the used authentication mechanism, where ECDSA outperforms RSA and even increases security, as discussed in Section 5.2. Choosing the cipher algorithm has the lowest impact and thus the least significance.

In the warm start scenario, the time overhead ranges between 0.64 and 0.83 %. Compared to the cold start, both the absolute values and the range of the overhead are much smaller. The largest difference of 0.19 % is between the TLS versions, followed by ECDSA and RSA.

Regarding CPU utilisation, the TLS versions, keys and cipher algorithms show the same trends as in the time values. Table 3 displays the absolute overhead of the different TLS options ranging between 0.68 and 0.86 % in a cold start. The warm start measurements did not show a noticeable overhead due to our measurement method. A scaling test with multiple warm start UEs revealed a low influence of TLS on a per-UE basis. Additionally, the non-UE-related system overhead of TLS was 0.343 % on our common-use CPU. Considering a more powerful real-world deployment, this overhead is negligible.

The necessary TLS handshakes need 3666 to 5725 Byte to be transmitted. As discussed before, ECDSA outperforms RSA clearly,

Table 3: Summary of performance overhead grouped by different factors. The top values are marked green and the bottom red.

Choice	Relative Time		Absolute CPU	
	Cold	Warm	Cold	Warm
None	-	-	-	-
TLS 1.2	**29.06 %**	**0.64 %**	**0.69 %**	< 0.1 %
TLS 1.3	**33.58 %**	**0.83 %**	**0.85 %**	< 0.1 %
ECDSA	**28.01 %**	**0.70 %**	**0.68 %**	< 0.1 %
RSA	**34.63 %**	**0.78 %**	**0.86 %**	< 0.1 %
AES-GCM	**31.27 %**	**0.74 %**	**0.76 %**	< 0.1 %
ChaCha-Poly	**31.41 %**	**0.75 %**	**0.77 %**	< 0.1 %

while TLS 1.2 needs fewer Byte than version 1.3. In a running system, that changes with TLS 1.3 adding 22 and TLS 1.2 adding 29 Byte per packet consisting of the header and MAC. Regarding storage, we derived a worst-case maximum of 26 MB for our complete 5GC setup. On commodity hardware this adds some overhead, but on a dedicated core network server architecture, these factors become insignificant.

As the TLS tunnels between the 5GC NFs are persistently retained, the impact of TLS on a running 5G system is less than on a system, that re-configures connections more often. While theoretically, each NF could close the tunnel after a period of inactivity, this functionality is not part of OpenSSL and would have to be implemented manually. In cases where this is used and cannot be disabled, the TLS heartbeat extensions could be used, although they are explicitly not recommended by some standardization bodies and have been vulnerable for exploits previously [9].

In summary, we recommend using TLS 1.3 over 1.2, since it adds more security and only performs slightly worse regarding time and CPU utilisation. ECDSA is recommended over RSA since it performs considerably better in all three categories and provides better security. We recommend the cipher algorithm AES-GCM, which performs better than ChaCha-Poly and AES-CCM. In a single cipher suite, this leads to a recommendation of TLS_AES_256_GCM_SHA384 using an ECDSA key. AES-256 provides doubled key size for symmetric encryption while adding only insignificant performance overhead compared to its AES-128 version.

6 RELATED WORK

Kotuliak et al. [15] compared the performance of IPsec and a TLS based VPN solution in the context of interconnected IP Multimedia Subsystems (IMSs). While they found a small performance advantage from using IPsec, this result might not hold up when using modern hardware or TLS directly without OpenVPN.

Like this, many other domain-specific performance evaluations for TLS have been done. This includes cloud environments [20], general purpose web servers [7], MQTT message brokers [22], SIP [24], and more. The outcome of these studies varies to a high degree by use case. It can show significant performance reduction in the case of SIP or almost no performance degradation in the case of MQTT, highlighting that there is no one-size-fits-all security solution.

In [14], Heijligenberg et al. investigate the impact of u-plane integrity protection in 5G. They conclude that, while there is a measurable impact on the latency, it should not impact the user experience outside of latency sensitive cases.

The study most similar to ours has been done by Vasoukolaei et al. [25], focusing specifically on the registration message between a NF and the NRF. They identified different metrics by capturing the traffic and extracting the required information from the dump. The procedure they chose is only used when a NF is first started or after a configuration change. The metric we are interested in is the user-visible end-to-end registration time from starting the UE until packets can be sent to the selected data network. Our cold start times therefore include the time required for NF registration, but also the times of other handshakes between the different components. The registration procedure is invoked quite regularly, e.g. after the airplane mode was used, the connection was lost, or the periodic registration timer has run out. The end-to-end measurements approach also allows us to test more ciphers with larger sample sizes.

7 CONCLUSION

In this work, we investigated the impact of TLS on the UE registration and PDU session establishment procedures in 5G. We differentiated between no TLS and TLS in versions 1.2 and 1.3. Additionally, we looked at the differences between a total of 14 cipher suites, that we selected carefully from recommendations of standardization entities like 3GPP, NIST, and BSI. To make statements about a worst-case overhead of TLS in case of a newly restarted system and the expected overhead of TLS in a running system, we performed cold and warm start tests. We found, that TLS adds a considerable overhead to the UE registration and session establishment time of 28.01-34.63 % in a cold start scenario. However, this worst-case overhead is expected to rarely happen in a real-world deployment. In a warm start scenario, the same time overhead of TLS ranges between 0.64 and 0.83 %. Therefore, the expected time overhead of TLS for a user's registration and PDU session establishment procedures is lower than 1 %. So, even if using TLS in the 5GC is not mandatory, we recommend using it for the additional security.

We found the CPU overhead, especially in a running system, to be small and thus insignificant in a large-scale 5G core deployment. With scaling tests, we showed the CPU overhead per UE to decline for larger packet counts. While the needed TLS handshakes add some overhead, they are just executed once at the start of the system. Similarly, we showed that the TLS memory overhead can be considered negligible, especially for larger deployments. Therefore, we consider the overhead of TLS on CPU, memory and network in a running system as insignificant.

In our work, we found cipher suites using an ECDSA key to be faster than ones using RSA with similar security properties. Although TLS 1.2 performs slightly better, we recommend TLS 1.3, since it adds downgrade protection and additionally encrypts more data.

In future work, we want to consider the impact of other security technologies, such as VPNs based on IPsec or WireGuard and conduct measurements in real-world mobile networks.

REFERENCES

[1] 3GPP. 2022. 3GPP TS 23.501 V17.5 5G: System architecture for the 5G System (5GS).

[2] 3GPP. 2022. 3GPP TS 33.501 V17.7 Security architecture and procedures for 5G system.

[3] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 2020. 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks* 167 (Feb. 2020), 106984. https://doi.org/10.1016/j.comnet.2019.106984

[4] Joseph Birr-Pixton. 2019. *jbp.io :: rustls versus OpenSSL: memory usage.* https://jbp.io/2019/07/02/rustls-vs-openssl-memory-usage.html

[5] Gabriel Brown. 2017. Service-Based Architecture for 5G Core Networks. (Nov. 2017).

[6] BSI. [n. d.]. *Modern telecommunications infrastructure (5G and 6G).* https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/5-G/5-g_node.html

[7] Cristian Coarfa, Peter Druschel, and Dan S. Wallach. 2006. Performance analysis of TLS Web servers. *ACM Transactions on Computer Systems* 24, 1 (feb 2006), 39–69. https://doi.org/10.1145/1124153.1124155

[8] Jean Paul Degabriele, Jérôme Govinden, Felix Günther, and Kenneth G. Paterson. 2021. The Security of ChaCha20-Poly1305 in the Multi-User Setting. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3460120.3484814

[9] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, and J. Alex Halderman. 2014. The Matter of Heartbleed. In *2014 Conference on Internet Measurement Conference* (Vancouver, BC, Canada) *(IMC '14)*. Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755

[10] ENISA. [n. d.]. *5G Security — ENISA.* https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services/telecoms/5g

[11] Internet Security Research Group. 2023. 2023 Annual Report. https://www.abetterinternet.org/documents/2023-ISRG-Annual-Report.pdf

[12] Ali Güngör. 2022. GitHub - UERANSIM Release 3.2.6. https://github.com/aligungr/UERANSIM

[13] Felix Günther, Martin Thomson, and Christopher A. Wood. 2023. *Usage Limits on AEAD Algorithms.* Internet-Draft draft-irtf-cfrg-aead-limits-07. https://datatracker.ietf.org/doc/draft-irtf-cfrg-aead-limits/07/ Work in Progress.

[14] Thijs Heijligenberg, Guido Knips, Christian Böhm, David Rupprecht, and Katharina Kohls. [n. d.]. BigMac: Performance Overhead of User Plane Integrity Protection in 5G Networks. In *16th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2023-05). ACM. https://doi.org/10.1145/3558482.3581777

[15] I. Kotuliak, P. Rybar, and P. Truchly. 2011. Performance comparison of IPsec and TLS based VPN technologies. In *2011 9th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. https://doi.org/10.1109/iceta.2011.6112567

[16] Branko Lankester, Michael K. Johnson, Michael Shields, and Charles Blake. 2023. *ps(1) — Linux manual page.* https://man7.org/linux/man-pages/man1/ps.1.html

[17] Sukchan Lee. 2023. GitHub - Open5GS Release 2.6.4. https://github.com/open5gs/open5gs

[18] Mattsson. 2014. *draft-mattsson-uta-tls-overhead-01.* https://datatracker.ietf.org/doc/html/draft-mattsson-uta-tls-overhead-01

[19] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. 2021. Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH (E). In *30th USENIX Security Symposium (USENIX Security 21)*. 213–230.

[20] Steffen Muller, David Bermbach, Stefan Tai, and Frank Pallas. 2014. Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems. In *2014 IEEE International Conference on Cloud Engineering*. IEEE. https://doi.org/10.1109/ic2e.2014.48

[21] Marcin Nawrocki, Thomas C. Schmidt, and Matthias Wahlisch. 2020. Uncovering Vulnerable Industrial Control Systems from the Internet Core. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE. https://doi.org/10.1109/noms47738.2020.9110256

[22] Thomas Prantl, Lukas Iffländer, Stefan Herrnleben, Simon Engel, Samuel Kounev, and Christian Krupitzer. 2021. Performance Impact Analysis of Securing MQTT Using TLS. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ACM. https://doi.org/10.1145/3427921.3450253

[23] OpenSSL Project. 2023. OpenSSL. https://www.openssl.org/

[24] Charles Shen, Erich Nahum, Henning Schulzrinne, and Charles Wright. 2010. The impact of TLS on SIP server performance. In *Principles, Systems and Applications of IP Telecommunications*. ACM. https://doi.org/10.1145/1941530.1941540

[25] Alireza Hosseini Vasoukolaei, Danish Sattar, and Ashraf Matrawy. 2021. TLS Performance Evaluation in the Control Plane of a 5G Core Network Slice. In *2021 IEEE Conference on Standards for Communications and Networking (CSCN)* (Thessaloniki, Greece, 2021-12-15). IEEE, 155–160. https://doi.org/10.1109/CSCN53733.2021.9686094

[26] David Warburton and Sander Vinberg. 2021. *The 2021 TLS Telemetry Report.* Technical Report. F5 Labs. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report

[27] Doug Whiting, Russ Housley, and Niels Ferguson. 2003. Counter with CBC-MAC (CCM). RFC 3610. https://doi.org/10.17487/RFC3610

[28] WolfSSL. 2021. *How Much Resource Does Your TLS Take? – wolfSSL.* https://www.wolfssl.com/much-resource-tls-take/

Table 5: Summary of exclusions (✗) and recommendations (✓) for TLS 1.2 cipher suites by 3GPP, NIST and BSI.

	Exclusion / Recommendation			
Cipher Suite	3GPP	NIST	BSI	Result
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_128_CBC_SHA	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_128_CBC_SHA256	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_128_GCM_SHA256	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_256_CBC_SHA	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_256_CBC_SHA256	✗	✗	✗	✗
TLS_DH_anon_WITH_AES_256_GCM_SHA384	✗	✗	✗	✗
TLS_DH_anon_WITH_RC4_128_MD5	✗	✗	✗	✗
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_DH_DSS_WITH_AES_128_CBC_SHA			✗	✗
TLS_DH_DSS_WITH_AES_128_CBC_SHA256				✗
TLS_DH_DSS_WITH_AES_128_GCM_SHA256				✗
TLS_DH_DSS_WITH_AES_256_CBC_SHA			✗	✗
TLS_DH_DSS_WITH_AES_256_CBC_SHA256				✗
TLS_DH_DSS_WITH_AES_256_GCM_SHA384				✗
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_DHE_DSS_WITH_AES_128_CBC_SHA			✗	✗
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256			✓[1]	✗
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256	✓	✓	✓	✗[2]
TLS_DHE_DSS_WITH_AES_256_CBC_SHA			✗	✗
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256			✓[1]	✗
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	✓	✓	✓	✗[2]
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_DHE_RSA_WITH_AES_128_CBC_SHA			✗	✗
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256			✓[1]	✗
TLS_DHE_RSA_WITH_AES_128_CCM	✓	✓	✓	✗[2]
TLS_DHE_RSA_WITH_AES_128_CCM_8	✓		✗	✗
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✗[2]
TLS_DHE_RSA_WITH_AES_256_CBC_SHA			✗	✗
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256			✓[1]	✗
TLS_DHE_RSA_WITH_AES_256_CCM	✓	✓	✓	✗[2]
TLS_DHE_RSA_WITH_AES_256_CCM_8	✓		✗	✗
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✗[2]
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	✓	✗	✗	✗
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_DH_RSA_WITH_AES_128_CBC_SHA			✗	✗
TLS_DH_RSA_WITH_AES_128_CBC_SHA256				✗
TLS_DH_RSA_WITH_AES_128_GCM_SHA256				✗
TLS_DH_RSA_WITH_AES_256_CBC_SHA			✗	✗
TLS_DH_RSA_WITH_AES_256_CBC_SHA256				✗
TLS_DH_RSA_WITH_AES_256_GCM_SHA384				✗
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA	✗	✗	✗	✗
TLS_ECDH_anon_WITH_AES_128_CBC_SHA	✗	✗	✗	✗
TLS_ECDH_anon_WITH_AES_256_CBC_SHA	✗	✗	✗	✗
TLS_ECDH_anon_WITH_NULL_SHA	✗	✗	✗	✗
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256				✗
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256				✗
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384				✗
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384				✗
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA		✗	✗	✗
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256			✓[1]	✗
TLS_ECDHE_ECDSA_WITH_AES_128_CCM	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8	✓		✗	✗
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA			✗	✗
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384			✓[1]	✗
TLS_ECDHE_ECDSA_WITH_AES_256_CCM	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8	✓		✗	✗
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✓
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	✓	✗	✗	✓[3]
TLS_ECDHE_ECDSA_WITH_NULL_SHA		✗	✗	✗
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA			✗	✗
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256			✓[1]	✗
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	✓	✓	✓	✓
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA			✗	✗
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384			✓[1]	✗
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	✓	✓	✓	✓
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	✓	✗	✗	✓[3]
TLS_ECDHE_RSA_WITH_NULL_SHA		✗	✗	✗
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256			✓[1]	✗
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256				✗
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384			✓[1]	✗
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384				✗
TLS_NULL_WITH_NULL_NULL	✗	✗	✗	✗
TLS_RSA_WITH_3DES_EDE_CBC_SHA		✗	✗	✗
TLS_RSA_WITH_AES_128_CBC_SHA			✗	✗
TLS_RSA_WITH_AES_128_CBC_SHA256			✗	✗
TLS_RSA_WITH_AES_128_CCM			✗	✗
TLS_RSA_WITH_AES_128_CCM_8			✗	✗
TLS_RSA_WITH_AES_128_GCM_SHA256			✗	✗
TLS_RSA_WITH_AES_256_CBC_SHA			✗	✗
TLS_RSA_WITH_AES_256_CBC_SHA256			✗	✗
TLS_RSA_WITH_AES_256_CCM			✗	✗
TLS_RSA_WITH_AES_256_CCM_8			✗	✗
TLS_RSA_WITH_AES_256_GCM_SHA384			✗	✗
TLS_RSA_WITH_NULL_MD5		✗	✗	✗
TLS_RSA_WITH_NULL_SHA		✗	✗	✗
TLS_RSA_WITH_NULL_SHA256		✗	✗	✗
TLS_RSA_WITH_RC4_128_MD5	✗	✗	✗	✗
TLS_RSA_WITH_RC4_128_SHA	✗	✗	✗	✗

Table 6: Summary of exclusions (✗) and recommendations (✓) for TLS 1.3 cipher suites by 3GPP, NIST and BSI.

	Exclusion / Recommendation			
Cipher Suite	3GPP	NIST	BSI	Result
TLS_AES_128_GCM_SHA256	✓	✓	✓	✓
TLS_AES_256_GCM_SHA384	✓	✓	✓	✓
TLS_CHACHA20_POLY1305_SHA256	✓	✗	✗	✓[3]
TLS_AES_128_CCM_SHA256	✓	✓	✓	✗[4]
TLS_AES_128_CCM_8_SHA256	✓		✗	✗

Table 4: List of preferences and exclusion criteria for each organization as per their relevant publications.

Organization	Preferences	Exclusion
3GPP	• AEAD cipher suites • PFS cipher suites	• Cipher suites with NULL integrity • Cipher suites with RC4 • Anonymous cipher suites
NIST	• GCM or CCM (not CCM_8) modes • PFS cipher suites	• Cipher suites not included in recommendation list
BSI	• Cipher suites marked as 2029+	• Cipher suites not included in recommendation list

A LIST OF SELECTED CIPHER SUITES

A cipher can either be excluded, neutral, or recommended. These classifications are based on TS 33.310 Annex E for the 3GPP, Special Publication 800-52 for the NIST, and TR-02102-2 for the BSI. Both preference and exclusion criteria were extracted as shown in Table 4. Only preferred ciphers were considered recommended and used for testing. TLS 1.0 and 1.1 were not considered as per RFC 8996. The associated data refers to information that is integrity protected but not encrypted. Perfect Forward Secrecy (PFS) refers to the property of a key exchange, that even if long-term key material (such as the TLS private keys) is recovered, past connections still cannot be decrypted. This is achieved by negotiating an ephemeral key for encryption, which is unique for each connection and dropped after usage. If a Diffie-Hellman based exchange is used, the actual symmetric key is never transmitted and can therefore not be recovered, even if the long-term keys used for the preliminary tunnel are known.

A.1 TLS 1.2

The list of available ciphers was extracted from RFCs 5246, 5288, 5289, 6655, 7251, 7905, and 8422. How the criteria map to the individual ciphers is shown in Table 5.

A.2 TLS 1.3

The list of available ciphers was extracted from RFC 8446. How the criteria map to the individual ciphers is shown in Table 6.

Unlike TLS 1.2, where the key exchange mechanism was encoded in the cipher suite, TLS 1.3 only allows ECDHE for non Pre-Shared Keys (PSK) suites. This means that all connections use PFS. The signature algorithm is negotiated similar to the cipher suite through the Client- and ServerHello messages.

[1] Only when Encrypt-then-MAC is enforced
[2] Excluded due to possible vulnerability to Raccoon Attack [19]
[3] Although neither NIST nor BSI officially recommends ChaCha20-based algorithms, they generally fulfill the stated requirements and are proven to be theoretically secure when used appropriately. [8] [13]
We opted to include them in this test since they are currently the only viable alternative to AES in TLS.
[4] Excluded due to runtime errors. Investigation on why is pending.