

On the Security of TLS Session Tickets

Sven Niclas Hebrok

Doctoral Dissertation



On the Security of TLS Session Tickets

Sven Niclas Hebrok

Doctoral Dissertation

Submitted to the Faculty of Computer Science, Electrical Engineering and Mathematics, Department of Computer Science, in partial fulfillment of the requirements for the degree of

Doktor der Naturwissenschaften (Dr. rer. nat)

Examiners:	**Prof. Dr.-Ing. Juraj Somorovsky** Paderborn University, Germany
	Prof. Dr. Jörg Schwenk Ruhr University Bochum, Germany
	Prof. Dr.-Ing. Sebastian Schinzel FH Münster, Germany
Thesis submitted:	23rd July 2026
Thesis defense:	08th October 2026
Last revision:	12th October 2026

Abstract

TLS is one of the most widely used security protocols, protecting the majority of online traffic. Its handshake uses computationally expensive public-key cryptography to establish a shared secret and authenticate the parties. To improve performance, TLS allows resuming a previous session using session tickets. These tickets contain the shared secret and must therefore be protected.

Previous works criticized session tickets for potentially undermining the forward secrecy of TLS. Nonetheless, session ticket implementations have not been well analyzed. We systematically analyzed the security of session tickets in TLS and identified potential implementation issues. We analyzed the ticket formats of 14 open-source TLS libraries. For five libraries, we identified partitioning oracle vulnerabilities. We found the other libraries to be secure. Despite this, we found that major HTTPS server implementations that use these libraries were vulnerable to client authentication bypasses when using virtual hosting and session resumption. We further evaluated our proposed attacks in the wild, uncovering severe vulnerabilities in large cloud providers, including AWS, Cloudflare, and Fastly. Our vulnerabilities circumvent the confidentiality and authentication guarantees of TLS, allowing an attacker to completely compromise the security of the communication. We argue that misuse-resistant APIs and more guidance in the TLS standards could prevent such vulnerabilities.

Zusammenfassung

TLS schützt den Großteil des Online-Datenverkehrs. Um die Leistung zu verbessern, ermöglicht TLS die Wiederaufnahme einer früheren Sitzung mithilfe von Session-Tickets. Diese Tickets enthalten das Schlüsselmaterial und müssen daher geschützt werden.

Session-Tickets wurden bereits dafür kritisiert, dass sie die Forward Secrecy von TLS untergraben können. Dennoch wurden Implementierungen von Session-Tickets bislang nicht gründlich analysiert. Wir haben die Sicherheit von Session-Tickets in TLS systematisch analysiert und potenzielle Implementierungsprobleme identifiziert. Wir haben die Ticketformate von 14 Open-Source-TLS-Bibliotheken analysiert und bei fünf Bibliotheken Partitioning-Oracle-Schwachstellen gefunden. Ansonsten haben wir die Bibliotheken als sicher befunden. Dennoch haben wir festgestellt, dass bei großen HTTPS-Servern, die auf diesen Bibliotheken aufbauen, die Client-Authentifizierung durch Sitzungsaufnahme umgangen werden konnte, wenn Virtual Hosting verwendet wird. Wir haben unsere Angriffe in der Praxis evaluiert und dabei schwerwiegende Schwachstellen bei großen Cloud-Anbietern wie AWS, Cloudflare und Fastly aufgedeckt. Unsere Schwachstellen umgingen die Vertraulichkeits- und Authentifizierungsgarantien von TLS, sodass ein Angreifer die Sicherheit der Kommunikation vollständig brechen konnte. Wir vertreten die Ansicht, dass missbrauchsresistente APIs und umfassendere Leitlinien in den TLS-Standards solche Schwachstellen verhindern könnten.

Acknowledgements

I want to thank everyone who supported me on this journey. First and foremost, thanks to my parents for supporting my interest in computers from a very young age. Thanks to my sister for always lending an ear. A special thank you to Lena for supporting me since nearly the beginning of my doctoral studies and always having my back. Even though you don't understand everything, you always eagerly listened to me and helped me practice my talks.

I thank Tibor Jager for his great IT security lecture that sparked my interest in the field during my bachelor's studies. I also want to thank all of his (former) employees for their great work during teaching. In particular, Kai Gellert for supervising my bachelor's thesis, and David Niehues and Pascal Bemmann for allowing me to tutor the IT security exercises.

Thank you, Juraj, for supervising my master's thesis and trusting me to continue my research as your first PhD student. It is a pleasure to work with you, and I appreciate your insightful feedback and guidance. I want to thank all my coworkers for the great work climate. I enjoy working with all of you and am glad for all the work-related and non-work-related discussions we had. Special thanks to Jost for sharing an office with me and putting up with me all the time. Thanks, Niklas, for introducing me to snowboarding, enjoying the same music together, and generally being a good friend. Thanks, Max, for being one of the greatest travel companions I could imagine. I want to thank the students I had the pleasure of supervising and writing papers with. Tim Leonhard Storm and Felix Matthias Cramer, it was a great time supervising your theses and turning them into papers. I want to thank Simon Nachtigall for his great work on his thesis, which I had the opportunity to turn into my first "first-author" paper. Also, thanks to all of my other coauthors for your feedback and discussions.

Special thanks go out to all my friends who accompany me in my free time. Jannik, thanks for managing the KCP and organizing all the kayak trips. Niklas, Jana, Matthis, Esther, Steffen, Jana, Valentin, Timo, Claire, Tim, Jay, Estha, and Jendrik, I'm looking forward to our next bouldering sessions and other get-togethers.

Contents

1 Motivation 1
1.1 Dangers of Session Resumption 2
1.2 Related Work 4
1.3 Outline 6
1.4 Citation System 6
1.5 Contributions 7
1.5.1 Publications Underlying This Thesis 7
1.5.2 Further Publications 10
1.5.3 Use of Generative AI and Other Tools 12
2 Background 13
2.1 Basic Cryptographic Concepts 13
2.1.1 Authenticated Encryption 13
2.1.2 Key Exchange 14
2.1.3 Certificates 15
2.2 TLS 15
2.2.1 DTLS and QUIC 18
2.3 Session Resumption 19
2.3.1 Key Derivation in Session Resumption 22
2.4 Ticket Construction 23
2.4.1 Ticket Plaintext 24
2.4.2 Ticket Ciphertext 25
2.5 Scanning Tools and Resources 26
2.5.1 Ethical Considerations 27
3 Real-World Ticket Ciphertext Construction 29
3.1 Background on Encryption Ciphers 31
3.1.1 Block Ciphers and Padding Oracles 31
3.1.2 Stream Ciphers 32
3.2 Cryptographic Pitfalls 33
3.2.1 List of Pitfalls 33
3.2.2 Impact on TLS Sessions in Different TLS Versions 36
3.3 Open-Source Analysis 36
3.4 Scanning and Evaluation Methodology 40
3.4.1 Online Server Scans 40

3.4.2 Offline Analysis 42
3.5 Large-Scale Evaluation 45
3.5.1 Online Tests 47
3.5.2 Offline Tests 48
3.5.3 Disclosure 51
3.5.4 Session Ticket Structures 51
3.6 Discussion 52
3.6.1 Potential Causes for Weak Keys 52
3.6.2 Countermeasures 54
3.6.3 Secure Email servers 55
3.6.4 Evaluating the Quality of Key Material 55
3.7 Conclusions 56
4 Session Resumption and Authentication 59
4.1 Background on Virtual Hosting and CDNs 62
4.2 Vulnerabilities from STEK Sharing 63
4.2.1 Server Authentication Bypass 64
4.2.2 Client Authentication Bypass 65
4.3 Open-Source Analysis 66
4.3.1 Server Authentication 68
4.3.2 Client Authentication 70
4.3.3 Inconsistent Behavior 73
4.4 Real-World Evaluation 74
4.4.1 Server Authentication Scan 74
4.4.2 Client Authentication Bypass 81
4.4.3 Limitations of our Evaluation 82
4.5 Discussion of Root Causes 83
4.6 Countermeasures 84
4.6.1 Implementation Pitfalls of the Countermeasures 85
4.7 Further Related Work 86
4.8 Conclusions 87
5 Partitioning Oracle Attack Against TLS 89
5.1 Background on Partitioning Oracle Attacks 91
5.2 Partitioning Oracle Attacks on TLS 92
5.3 Library Evaluation 93
5.3.1 Methodology 93
5.3.2 Results 94

5.3.3 Attack Performance 98
5.3.4 Attack Limitations Against ChaCha20/Poly1305 98
5.4 Large-Scale Evaluation 99
5.4.1 Library Identification 99
5.4.2 Scan Setup 100
5.4.3 Methodology 101
5.4.4 Scan results 101
5.5 Related Work on Partitioning Oracle Attacks 102
5.6 Conclusions and Future Work 103
6 Discussion 105
7 Conclusions and Future Work 109
Bibliography 111
Own Publications 111
Academic Publications 112
Standards and References 126
Websites and Blogs 129
Code, Software, and Documentation 132
A Appendix 139
A.1 IANA TLS Services 139
A.2 Parameters for Ticket Decryption 139
A.3 Key Sets for Partitioning Oracles 141
A.4 Library Versions and Sources 141

Chapter 1
Motivation

Transport Layer Security (TLS) is arguably the most widely used security protocol on the internet. It secures the majority of internet communication, for example, when browsing websites (via HTTPS) [S117, W129, W132] or using email [S100, S107, S109, W131]. There are over 100 well-known services that use TLS.[1] In all of these protocols, the application protocol is tunneled through TLS, without needing to adjust the application protocol. TLS can be used in this variety of contexts because it was designed to be independent of the concrete application protocol.

Due to its widespread use, an industry has formed around providing TLS. This ranges from open-source software libraries [C151, C161, C162, C172, C175, C176, C178, C182, C185, C190, C193, C194, C195, C200] to proprietary cloud providers that provide TLS for applications [C157, C168, C169, C173, C180]. These services allow developers to easily integrate TLS into their applications without needing to understand the complexities of the protocol. At the same time, they concentrate security-critical decisions in a comparatively small set of implementations and services. As a result, design flaws or implementation mistakes in a widely deployed TLS stack can affect many applications and deployment scenarios simultaneously.

Security. TLS provides security guarantees for the applications that use it, such as confidentiality and integrity of the communication. It further promises Perfect Forward Secrecy (PFS),[2] which ensures that the compromise of long-term keys does not affect the confidentiality of past sessions. This should prevent an attacker from decrypting traffic, even if they compromise the long-term keys.

Motivated by the widespread use of TLS and its strong guarantees, its security has been extensively analyzed at both theoretical and practical levels. Several works have analyzed and proven the theoretical security of

1. In Appendix A.1 we explain how we determined the number of services based on data provided by IANA [W134].
2. TLS 1.2 only provides PFS if an ephemeral Diffie-Hellman key exchange is used. TLS 1.3 supports only such a key exchange.

TLS [19, 33, 51, 54]. For TLS 1.3, the design process included formal analysis of early drafts by independent researchers [18, 28]. Nonetheless, the assumptions made in these analyses may not always hold in practice, allowing attacks [61]. Some attacks exploit implementation mistakes, which are not covered by the theoretical analyses, such as Bleichenbacher's attack [20] and CBC padding oracles [80]. The applicability and impact of these attacks have been widely analyzed [23, 62, 63, 69]. Other attacks might only surface in combination with the application layer protocol, such as the recent ALPACA [22] and Opossum attacks [O5].

Session Resumption. A feature of TLS that has received comparatively little attention is session resumption. Session resumption improves the performance of subsequent TLS connections and is therefore commonly used in practice [W137, W144]. It works by allowing a client and server to reuse the established parameters and key material from a previous connection, without needing to perform a full handshake. To achieve this, either both parties store the session state from the previous connection (session IDs) [S115] or the server provides the client with a session ticket containing the necessary information for resumption [S96]. Session tickets are specified as an extension to TLS. The server can freely choose the format of these tickets. The standard recommends that the server encrypts and authenticates the session information using a symmetric key, the Session Ticket Encryption Key (STEK). The benefit of session tickets is that the server does not need to store state for each session; instead, it stores only one STEK for all sessions, improving scalability.

1.1 Dangers of Session Resumption

Perfect Forward Secrecy. While session resumption improves performance, it also introduces security risks. It can severely undermine PFS, as compromising the session cache or STEK compromises the confidentiality of all sessions contained in the cache or issued under this key. In TLS 1.2, this even includes past sessions and sessions that are not resumed. This issue has been highlighted previously [71, W145, W147]. Valsorda [W147] highlights how this enables dragnet surveillance by state-level adversaries. Springall et al. [71] measured the potential impact of this issue in practice, finding that 10 % of domains in the Alexa Top Million would continuously use the same STEK for at least 30 days. These prior works analyzed only

the potential impact of a STEK compromise and assumed a nation-state attacker. However, in 2020, Klute [W136] found that the STEK of GnuTLS was initialized improperly on startup and set to all-zero bytes [W138, W150].

Motivated by this finding, we systematically analyzed the security of session tickets in real TLS deployments (Chapter 3). We systematically analyzed the ticket implementations of 14 open-source TLS libraries. We discovered no immediate issues and found the libraries to be generally secure. However, when analyzing real-life deployments, we found several servers using all-zero STEKs or otherwise trivially guessable keys. Among the vulnerable servers were also large cloud providers, such as AWS and StackPath.

Authentication. Apart from PFS, session resumption also affects the authentication guarantees of TLS. In a full TLS handshake, the server is authenticated via its certificate, and the client can optionally authenticate itself using a certificate. In session resumption, neither party is authenticated again, and authentication is delegated to the ability to resume the session. While this seems to be a reasonable design choice, it has led to practical vulnerabilities when the session cache or STEK is shared across different hosts. Delignat-Lavaud and Bhargavan [30] found practical vulnerabilities in deployments of Akamai, Dropbox, and Mozilla. Their attack, which they called *virtual host confusion*, allowed an attacker to load attacker-controlled content within a different host's TLS session, leading to Cross-Site Scripting (XSS). They manually evaluated this vulnerability in a few cases but did not perform an automated large-scale scan to assess the prevalence of this issue in practice. Instead, they focused on a broad study of various attacks.

Since their study in 2015, TLS has become more widespread, the use of Content Delivery Networks (CDNs) has increased, and session tickets have become more common. We revisited the virtual host confusion attack 10 years later with an extended evaluation (Chapter 4). In our open-source analysis, we did not find vulnerable server implementations. Our automated large-scale evaluation uncovered vulnerabilities in multiple CDNs, including Cloudflare and Fastly. We additionally evaluated the security impact of session resumption on client authentication. Resumption might allow an attacker to bypass authentication. OpenSSL introduced API

changes to fix this vulnerability in 1999 [C177]. Nonetheless, our evaluation uncovered vulnerabilities in all analyzed web server implementations. We could not automatically evaluate the client authentication bypass on a large scale. Manual testing still revealed a vulnerability in Cloudflare.

Partitioning Oracle Attacks. In our analysis of session ticket formats, we found several libraries use AES-GCM or ChaCha20-Poly1305 for session ticket authentication. In 2021, Len et al. [57] presented a new attack against these schemes. Their attack allows recovering the encryption key by forging ciphertexts and observing whether the server accepts them. For this attack to work, the actual key must be of low entropy. No explicit claims have been made about the applicability of this attack to TLS. It was implied, however, that this requirement prevents the attack from working, since the keys in TLS are generated with sufficient entropy.

In our analysis of session ticket deployments, we discovered that the STEK is not always of high entropy. We evaluated the applicability of partitioning oracle attacks to TLS session tickets (Chapter 5). Of the seven analyzed libraries that use the affected algorithms, we found that five libraries provide an oracle. For two libraries, the forged ciphertext is precomputable, which could make the attack viable in practice. To assess real-world impact, we scanned live deployments. We found no server using both a vulnerable implementation and a low-entropy STEK.

1.2 Related Work

Session Tickets. In the previous section, we already discussed several works criticizing session tickets for their implications on PFS [O2, O7, 71, W145, W147] and authenticity [O3, 30]. Apart from these, tickets have also been criticized for their privacy implications. In 2018, Sy et al. [74] analyzed how users can be tracked across the web via TLS session resumption mechanisms. They found that 65 % of users could be permanently tracked by at least one website using TLS session tickets with a lifetime of at least seven days due to continuous ticket renewal. In 2019, Arfaoui et al. [10] analyzed the theoretical properties of session tickets in TLS 1.3. They conclude that session tickets allow linking sessions. However, they note that this is inherent to the design of TLS session resumption and not a specific issue of session tickets.

In 2020, Sy et al. [75] proposed using session resumption across hostnames. They proposed a new TLS extension that informs the client which hostnames within the same certificate share the same STEK, enabling the client to use session resumption across these hostnames. Gellert and Handirk [40] formally analyzed the security of this proposed extension, showing that it can be securely implemented and does not introduce new security issues.

We are not the first to find practically exploitable vulnerabilities in session tickets. Valsorda [W146] discovered an implementation flaw similar to Heartbleed [36] in the F5 TLS session ticket implementation that allowed an attacker to extract up to 31 bytes of memory at a time.

TLS Large-Scale Scans. Large-scale measurements have become a standard approach for studying the ecosystem of deployed TLS infrastructure. Kotzias et al. [53] combined passive observations collected over six years with internet-wide measurements to trace how protocol features spread and how deployments reacted to newly disclosed attacks. Durumeric et al. [35] used repeated active scans to characterize the HTTPS certificate ecosystem and to uncover systemic risks. Subsequent work extended this perspective beyond HTTPS by examining TLS deployment in other protocols, including SMTP, IMAP, XMPP, and IRC [49, 60].

In our studies [O2, O3, O5, O7], we performed large-scale scans to evaluate the prevalence of the vulnerabilities we proposed and to estimate their impact. Internet-wide scans were previously used to quantify Heartbleed exposure, patch deployment, and evidence of real-world exploitation [36]. Related studies measured the prevalence of several weaknesses in Diffie-Hellman-based key exchange, including insufficient validation of key exchange parameters and the use of composite moduli [32, 78, 79]. The same methodology was also applied to estimate the attack surface of the DROWN attack [12], Bleichenbacher-style vulnerabilities [23], CBC padding oracles [62], the Raccoon attack [61], and the Opossum attack [O5]. Further large-scale studies analyzed cross-protocol confusion caused by certificate reuse across services [22]. Others identified servers that emit faulty RSA signatures from collected TLS traffic, enabling secret-key recovery [73]. Heninger et al. [45] analyzed the prevalence of shared RSA moduli, which is only feasible with large-scale measurements.

1.3 Outline

This thesis is divided into seven chapters. It builds on three of my publications focusing on the security of TLS session resumption and the mechanisms used to implement it. I outline my personal contributions and those made in this thesis in Section 1.5.

We begin with a common background chapter in Chapter 2 that explains TLS and session resumption using session tickets. To start the analysis of tickets, we first analyze how session tickets are constructed cryptographically in Chapter 3, which is based on our paper from 2023 [O2]. We propose potential pitfalls that can occur when implementing session tickets, analyze how open-source libraries implement them, and evaluate the security of deployed servers through a large-scale scan. In Chapter 4, we analyze how session tickets affect the security in virtual hosting scenarios, which is based on our paper from 2025 [O3]. We again start by proposing possible attacks, analyzing open-source libraries, and then evaluating the security of deployed servers through a large-scale scan. Based on our analysis in Chapter 3, we saw multiple libraries using AES-GCM or ChaCha20-Poly1305. In Chapter 5, we analyze whether these libraries are susceptible to partitioning oracle attacks [57], which is based on our paper from 2024 [O7]. We discuss the implications of our findings in Chapter 6 and conclude this thesis in Chapter 7.

1.4 Citation System

We divide the bibliography of this thesis into five parts by publication type. References are numbered continuously throughout the entire bibliography. Nonetheless, each part uses a distinct prefix, making the type of cited publication easy to identify. The parts and prefixes are as follows:

Own Publications (e.g., [O3]) Academic publications I co-authored.

Academic Publications (e.g., [30]) Peer-reviewed papers and completed theses.

Standards and References (e.g., [S96]) Standards that define algorithms, protocols, or other technical specifications.

Websites and Blogs (e.g., [W147]) Non-peer-reviewed web sources.

Code, Software, and Documentation (e.g., [C185]) References to software artifacts, their code, and documentation.

1.5 Contributions

During my studies, I contributed to several publications. In the following, I outline my contributions to all of these publications. For the publications underlying this thesis, I also outline any contributions included in this thesis that go beyond the published versions of the papers. For the publications that are not included in this thesis, I also provide a summary of the publication.

1.5.1 Publications Underlying This Thesis

Chapter 3
[O2] **Sven Hebrok**, Simon Nachtigall, Marcel Maehren, Nurullah Erinola, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. 2023. **We Really Need to Talk About Session Tickets: A Large-Scale Analysis of Cryptographic Dangers with TLS Session Tickets**. In *32nd USENIX Security Symposium (USENIX Security '23)*, 2023. USENIX Association, Anaheim, CA, 4877–4894. Retrieved from https://www.usenix.org/conference/usenixsecurity23/presentation/hebrok

Contributions to the Publication. The foundation of this paper is based on Simon Nachtigall's master's thesis, in which he conducted the first large-scale scans (pre-T1M, T1M) and analysis of open-source implementations. I co-supervised the thesis and provided feedback throughout. I was in charge of extending the study into a full paper. To this end, I substantially extended the work by designing and performing further scans (T100k, IP100k, IPF). I extended the scan implementation and methodology to deepen the analysis and support internet-wide scanning with ZMap and ZGrab. I analyzed the data and presented the results in the paper. I presented the work at the conference.

Contributions in this Thesis. Since the original paper was published in 2023, the analyzed libraries have released new versions. I re-evaluated the current versions of the libraries and included additional open-source implementations in the analysis to reflect the current state of the ecosystem. I redesigned several figures to improve their clarity and readability. In this thesis, I also include data from a later scan evaluating email servers (IPmail) and discuss them (Section 3.6.3).

Chapter 4

[O3] **Sven Hebrok**, Tim Leonhard Storm, Felix Matthias Cramer, Maximilian Radoy, and Juraj Somorovsky. 2025. **STEK Sharing is Not Caring: Bypassing TLS Authentication in Web Servers using Session Tickets**. In *34th USENIX Security Symposium (USENIX Security '25)*, 2025. USENIX Association, Seattle, WA, 8017–8034. Retrieved from https://www.usenix.org/conference/usenixsecurity25/presentation/hebrok

Contributions to the Publication. This paper started as two bachelor's theses: one by Tim Leonhard Storm [72], the other by Felix Matthias Cramer [27]. I proposed both topics and supervised them. During the supervision, I regularly discussed the current approach and guided them in the research direction. This included high-level feedback on the methodology, as well as low-level reviews and debugging of the implementation and findings. To extend the topics into a full paper, I cooperated with the students in the research direction and provided feedback on their work. I wrote a substantial portion of the tests and scan code, wrote proof-of-concepts for the attacks, and led the responsible disclosure process with the vendors and developers. I was in charge of the writing process, coordinating all authors. I presented the work at the conference.

Contributions in this Thesis. To evaluate whether the server authentication bypass applies to different ecosystems, I performed a large-scale scan of email servers (IPmail in Section 4.4.1.2). I could not find definitive proof that the attack is possible in this scenario. I also extended the analysis of the client authentication bypass to include Apache Tomcat and Microsoft IIS and found issues in both. Willi Hensch first analyzed these two servers in his Bachelor's Thesis [46], which I supervised. I reevaluated his findings and analyzed the configuration to find the root causes in Microsoft IIS (Section 4.3.2.2).

Chapter 5
[O7] Maximilian Radoy, **Sven Hebrok**, and Juraj Somorovsky. 2024. **In Search of Partitioning Oracle Attacks Against TLS Session Tickets**. In *29th European Symposium on Research in Computer Security*, 2024. Bydgoszcz, Poland, 320–340. https://doi.org/10.1007/978-3-031-70896-1_16

Contributions to the Publication. This paper started as the master's thesis of Maximilian Radoy. I proposed the thesis topic and supervised it. My main contribution is the supervision during the thesis and guidance on the research direction. In the final paper, I guided Maximilian Radoy in the writing process and wrote a large portion of the final text.

Contributions in this Thesis. I extended the analysis to include JSSE (OpenJDK), which I found to be vulnerable.

1.5.1.1 Permissions

For the works published at USENIX Security [O2, O3], I retain the copyright of the work and can therefore include it in this thesis without needing explicit permission: "USENIX permits authors to retain their ownership of the copyrights in their works."[3]

For the work presented at ESORICS [O7], I also retain the copyright and am allowed to use the published version in a thesis with an appropriate acknowledgement: The work was first published in *29th European Symposium on Research in Computer Security (ESORICS 2024), Bydgoszcz, Poland, September 16–20, 2024,* pages 320–340, by Springer Nature. Reproduced with permission from Springer Nature.

3. https://www.usenix.org/conferences/author-resources/submissions-policy (accessed: 2026-07-19)

1.5.2 Further Publications

[O4] Marcel Maehren, Philipp Nieting, **Sven Hebrok**, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. 2022. **TLS-Anvil: Adapting combinatorial testing for TLS libraries**. In *31st USENIX Security Symposium (USENIX Security '22)*, August 2022. USENIX Association, Boston, MA, 215–232. Retrieved from https://www.usenix.org/conference/usenixsecurity22/presentation/maehren

Summary. TLS-Anvil[4] is a test framework for TLS implementations. During a TLS handshake, many parameters are negotiated, and some bugs occur only with specific combinations of these parameters. TLS-Anvil uses combinatorial testing to systematically evaluate combinations of parameters. It includes numerous tests based on 13 TLS RFCs. In the paper, the framework was presented, and 13 TLS implementations were evaluated. The framework's architecture allows developers to easily add new tests and support new implementations. The paper also presents a library of Docker images for testing TLS implementations, which is also available as open-source.[5]

Contributions to the Publication. I implemented the RFC 7919 test case, validated its behavior, and integrated the required parameter derivation into the test framework. During the writing process, I reviewed the paper and proposed changes that improved both technical presentation and overall clarity.

[O6] Niklas Niere, **Sven Hebrok**, Juraj Somorovsky, and Robert Merget. 2023. **Poster: Circumventing the GFW with TLS Record Fragmentation**. *CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (November 2023), 3528–3530. https://doi.org/10.1145/3576915.3624372

Summary. The poster presents a work in progress (at that time) on TLS censorship. With this poster, a novel approach to circumvent the Great Firewall of China (GFW) was presented. Usually, the GFW censors based

4. https://github.com/tls-attacker/TLS-Anvil (accessed: 2026-07-19)
5. https://github.com/tls-attacker/TLS-Docker-Library (accessed: 2026-07-19)

on the hostname specified in the ClientHello's SNI extension. The new approach splits the TLS message such that the hostname is split into two TLS records. In contrast to other censorship circumvention methods, this approach is standard-compliant and well supported by TLS servers (>90 % support). Therefore, this approach can immediately be used with most servers. Additionally, the approach can be implemented in a proxy and does not require changes on the client side, nor does it affect the security of the connection. Together with this publication, DPYProxy[6] was published, which implements this approach.

Contributions to the Publication. I discussed the technical details of the work with Niklas Niere and provided feedback throughout. Based on my previous research on TLS and my experience with TLS-Attacker, I could provide helpful insights into TLS. I assisted him in designing the final poster he presented at CCS 2023.

[O1] Fabian Bäumer, Marcus Brinkmann, Nurullah Erinola, **Sven Hebrok**, Nico Heitmann, Felix Lange, Marcel Maehren, Robert Merget, Niklas Niere, Maximilian Radoy, Conrad Schmidt, Jörg Schwenk, and Juraj Somorovsky. 2024. **TLS-Attacker: a dynamic framework for analyzing TLS implementations**. In *Proceedings of cybersecurity artifacts competition and impact award (ACSAC '24)*, 2024. Hawaii. Retrieved from https://www.acsac.org/2024/program/artifacts_competition/comp-acsac24-final11.pdf

TLS-Attacker was awarded second place at the competition.

Summary. TLS-Attacker is a widely used framework for analyzing TLS implementations. It was originally published by Somorovsky in 2016 [70] and has since been extended by many contributors. In this publication, we briefly introduce the framework and present its impact on the research community and industry.

Contributions to the Publication. For the publication, I conducted literature research to identify publications that use tools from the TLS-Attacker framework. Together with Niklas Niere, we presented the work at ACSAC 2024.

6. https://github.com/UPB-SysSec/DPYProxy (accessed: 2026-07-19)

[05] Robert Merget, Nurullah Erinola, Marcel Maehren, Lukas Knittel, **Sven Hebrok**, Marcus Brinkmann, Juraj Somorovsky, and Jörg Schwenk. 2026. **Opossum Attack: Application Layer Desynchronization using Opportunistic TLS**. In *35th USENIX Security Symposium (USENIX Security '26)*, 2026. USENIX Association, Baltimore, MD. Retrieved from https://www.usenix.org/conference/usenixsecurity26/presentation/merget

Summary. Many protocols were initially designed as plaintext protocols. To provide security, these protocols were later extended to support TLS. Many protocols supported two approaches for this: starting with a plaintext protocol and upgrading to TLS (explicit TLS/STARTTLS) or starting with TLS from the beginning (implicit TLS). In this paper, we present the Opossum attack: When a server supports both approaches, an attacker can desynchronize the server and client. In HTTP, this allows an attacker to override the complete request, enabling multiple attacks, including XSS.

Contributions to the Publication. I contributed to the analysis of HTTP-based protocols. Together with Lukas Knittel, I analyzed the behavior of Apache httpd and created PoCs for the attack. I further analyzed CUPS's IPP implementation and checked for implementation-specific vulnerabilities. I mainly contributed to the written paper through discussions and visualizing the attack.

1.5.3 Use of Generative AI and Other Tools

While writing this thesis, GitHub Copilot[7] was used. It provided suggestions to complete sentences (autocomplete) and possible fixes while writing the text and Typst[8] code. I used it to improve the clarity of this thesis by prompting it to reword sentences or paragraphs. I further let Copilot review the thesis content and grammar, where it provided comments for me to work on. I used Grammarly[9] to check the grammar and spelling of the text. I used DeepL[10] to initially translate the abstract from English to German, which I then adjusted manually.

7. https://github.com/features/copilot/ (accessed: 2026-07-19)
8. https://typst.app/ (accessed: 2026-07-19)
9. https://www.grammarly.com/ (accessed: 2026-07-19)
10. https://www.deepl.com/ (accessed: 2026-07-19)

Chapter 2
Background

Transport Layer Security (TLS) allows for secure communication over an otherwise insecure network. It is designed to transparently tunnel an application protocol while providing confidentiality, integrity, and authenticity. To improve the performance of subsequent connections, TLS supports session resumption. This allows a faster connection establishment if both parties have previously communicated and still have the necessary information to establish a secure connection. A common implementation is to use session tickets, where the client stores a ticket provided by the server and can later use this ticket to resume a session.

In the following sections, we provide a more detailed background on TLS, session resumption, and session tickets.

2.1 Basic Cryptographic Concepts

TLS builds on three families of cryptographic primitives to provide its security guarantees. Symmetric encryption protects the confidentiality of transmitted data, and Message Authentication Codes (MACs) ensure its integrity. Public-key cryptography is used to establish shared key material and authenticate the communicating parties.

2.1.1 Authenticated Encryption

Authenticated Encryption (AE) provides confidentiality and integrity in a single primitive. Commonly, this is implemented by combining symmetric encryption with a MAC over the ciphertext. When using AE, tampering with the ciphertext is detected during decryption, preventing an attacker from silently modifying the communication. An extension of this is Authenticated Encryption with Associated Data (AEAD), which additionally protects Associated Authentication Data (AAD). This AAD is not encrypted, but is still protected by the integrity guarantees of the AEAD scheme. This is useful for any data that needs to be sent in plaintext but should still be protected against tampering, for example, the header of a network packet.

MACs may be combined differently with symmetric encryption. Computing the MAC over the ciphertext, as above, is called Encrypt-then-MAC. Alternatively, the MAC can be computed over the plaintext. If the tag is appended to the plaintext and then encrypted, this is called MAC-then-Encrypt. If the tag is appended to the ciphertext, this is called MAC-and-Encrypt. It is generally recommended to use Encrypt-then-MAC [14, S98], as the validity of the ciphertext is verified before decryption. This prevents an attacker from exploiting potential vulnerabilities in the decryption algorithm, such as a padding oracle vulnerability [80].

Carter-Wegman MACs. A common approach to construct AE(AD) schemes is to internally use an encryption algorithm and a MAC and append the MAC tag to the resulting ciphertext. AES-GCM and ChaCha20-Poly1305 are examples of this approach. They use Carter-Wegman-style MACs internally [16, S94]. Carter-Wegman MACs typically compute the tag by constructing a polynomial from the input message and evaluating it with the key as input [82]. These MACs can be implemented efficiently and have therefore become a common choice for AE(AD) schemes [53].

AES-GCM [S94] combines AES-CTR[11] with a Carter-Wegman MAC over the Galois Field GF(2^{128}). ChaCha20-Poly1305 [16, 17, S111] combines the ChaCha stream cipher with a Carter-Wegman MAC based on a polynomial modulo $2^{130} - 5$.

2.1.2 Key Exchange

To use symmetric cryptography, both parties need to agree on a shared secret key. This is achieved using a key exchange protocol, which allows two parties to agree on a shared secret over an insecure channel. Commonly, a Diffie-Hellman key exchange is used, which provides Perfect Forward Secrecy (PFS). This means that the long-term private key is not involved in deriving the session secret. Therefore, a subsequent compromise of the long-term key does not expose any communication that occurred before the compromise. After the key exchange, the shared secret is used to derive the symmetric keys used for encryption and integrity protection.

11. The nonce is initialized differently. Otherwise, the computations are the same.

2.1.3 Certificates

A key exchange on its own does not provide any authentication of the communicating parties and thus does not prevent an attacker from impersonating one of them. To achieve authentication, TLS uses X.509 certificates [S92, S102]. These certificates contain at least a public key and a name (e.g., a domain) to which this key belongs. As anyone can generate a certificate and claim that a name belongs to their public key, certificates are signed by a trusted third party, known as a Certificate Authority (CA). Receivers can verify the signature(s) on the certificate using the public key of the CA, which is known to the receiver. This way, the client can validate that the certificate is valid and trusted. On its own, the certificate does not prove that the entity presenting the certificate is the owner of the public key. Therefore, TLS uses signatures over (parts) of the handshake to prove possession of the private key corresponding to the public key in the certificate.

2.2 TLS

TLS [S115, S119] is designed to secure the communication between two peers, a client and a server. To achieve this, TLS works in two steps. First, a handshake is performed to establish a secure channel between the client and server. The handshake negotiates parameters of the secure channel, establishes shared key material, and authenticates the server (and optionally the client). Then, the application protocol is tunneled through this secure channel, providing confidentiality, integrity, and authenticity.

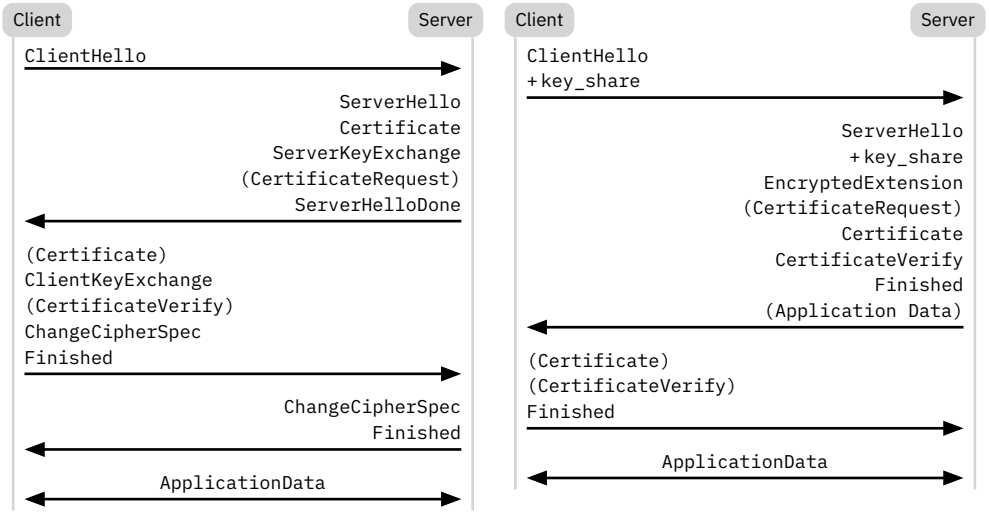
Attacker Model. TLS aims to protect against passive and active attackers. Against passive attackers who can only observe the communication, TLS encrypts it. This should prevent the attacker from learning the content of the communication. Against active attackers, also called Malicious-in-the-Middle (MitM), that can manipulate the communication, TLS additionally authenticates the server and optionally the client. This ensures that an attacker cannot impersonate another party. Server authentication is important because it prevents the attacker from tricking the client into sending sensitive information. Similarly, client authentication is relevant to prevent the attacker from impersonating the client to the server.

Versions. TLS has undergone several iterations. The newest version is TLS 1.3. It was first standardized in 2018 [S118] and was recently updated in 2026 [S119]. This version simplified the protocol by removing parameters that could lead to insecure implementations and configurations. It also improved the performance of the handshake, allows forward-secure session resumption, and supports 0-RTT resumption. Its predecessor, TLS 1.2 [S115], remains the most widely supported version of TLS [W142] and is still considered secure when properly configured. All previous versions, including SSL (Secure Sockets Layer), are considered deprecated [S90, S108, S114]. In this thesis, we focus on TLS 1.2 and TLS 1.3. Some analyses in this thesis also consider TLS 1.0 [S87] and TLS 1.1 [S93]. In the context of this thesis, these versions can be considered to work like TLS 1.2.[12]

Handshake. TLS first performs a handshake to establish a secure channel between the client and server. While the concrete messages differ between TLS 1.2 and TLS 1.3, as shown in Figure 1, the general concepts of the handshake are similar in both versions. During the handshake, three goals are achieved:

(1) The parties agree on the parameters of the secure channel. Central to this is the choice of the *cipher suite* to use, which determines the key exchange algorithm (TLS 1.2 only), the encryption algorithm, and the hash function used for authentication.[13]
(2) The parties establish a shared secret, which will be used to protect the communication. Typically, an ephemeral Diffie-Hellman key exchange is used, providing PFS. In TLS 1.2, RSA can also be used to establish the shared secret.
(3) The server and optionally the client authenticate themselves to each other, proving that no attacker is impersonating one of the parties. To this end, the server sends a certificate and a signature, which the client validates. For client authentication, this is done analogously, with the client sending a certificate and signature to the server.

12. The changes in these versions mainly focus on cryptographic agility (e.g., the PRF is determined by the cipher suite, support for AEAD). The overall structure of the protocol and the handshake remained the same.
13. The hash function is also used to derive keys. Therefore, cipher suites using AEAD algorithms also specify the hash function, even though it is not used for authentication.



(a) TLS 1.2 (b) TLS 1.3

Figure 1: Full TLS Handshake. Optional messages are shown in parentheses. Relevant extensions are shown with a + prefix.

Handshake in TLS 1.2. The client initiates the handshake with the `ClientHello` message. This message contains a list of cipher suites and other parameters (e.g., supported Diffie-Hellman groups [S97]) the client is willing to use. The server responds with a `ServerHello` message, selecting the parameters for the secure channel from the client's proposed options (1). The server then sends its certificate in the `Certificate` message. Next, it sends a `ServerKeyExchange` message, which contains the server's Diffie-Hellman key share. The server additionally signs the `ServerKeyExchange` message and parts of the `ClientHello` and `ServerHello` messages, proving its authenticity (3).[14] After these messages, the server sends a `ServerHelloDone` message, indicating that it has finished its part of the handshake.

The client then sends a `ClientKeyExchange` message, which contains the client's Diffie-Hellman key share. At this point, both parties can compute the shared secret from the exchanged key shares (2). Then, the client sends a `ChangeCipherSpec` message, indicating that it will start using the negotiated parameters for the secure channel (i.e., encrypt all subsequent messages). Finally, the client sends a `Finished` message, which the server

14. We assume a Diffie-Hellman key exchange in this description. In an RSA key exchange, no `ServerKeyExchange` message is sent. The server's authenticity is proven by the ability to finish the handshake, which requires knowledge of the private key.

validates. This message proves that both parties have derived the same shared secret and that the handshake messages were not tampered with.

The server responds with a `ChangeCipherSpec` message, indicating that it will also start using the negotiated parameters for the secure channel. It also sends a `Finished` message to ensure the handshake messages were not tampered with.

If the server requires client authentication, it requests a certificate from the client by sending a `CertificateRequest` message before the `ServerHelloDone` message. The client then sends a `Certificate` and `CertificateVerify` message, proving its authenticity.

Handshake in TLS 1.3. The handshake in TLS 1.3 is conceptually similar to TLS 1.2 but has concrete differences: The handshake requires one less round trip than TLS 1.2. The client already includes its Diffie-Hellman key share in the `ClientHello` message,[15] and the server includes its key share in the `ServerHello` message. This allows every message after the `ServerHello` to be encrypted, including the server's certificate and signature, improving privacy. Instead of signing the key exchange message, the server also uses a `CertificateVerify` message to prove its authenticity. The handshake still achieves the same goals as in TLS 1.2 but is more efficient and encrypts more handshake messages (e.g., server certificate).

2.2.1 DTLS and QUIC

TLS is based on TCP and can be used in place of TCP to provide a reliable[16] and secure transport. However, this means that TLS cannot be used in scenarios where TCP is not suitable, for example, in real-time applications that often use UDP. To address this, the Datagram Transport Layer Security (DTLS) protocol was developed [S116], which can be used in place of UDP to provide a secure transport layer that behaves like UDP.

15. If the client sends a key share that the server does not support, the server will send a `HelloRetryRequest` message, asking the client to send a key share for a different group. The handshake will then start again with the client sending a new `ClientHello` message containing a supported key share.
16. N.B. "Reliable" refers to the network concept of reliability. A reliable protocol ensures that sent data is either received by the other party or the sender is notified that the data was not received. In contrast, UDP is an unreliable protocol in which messages may be dropped without notice.

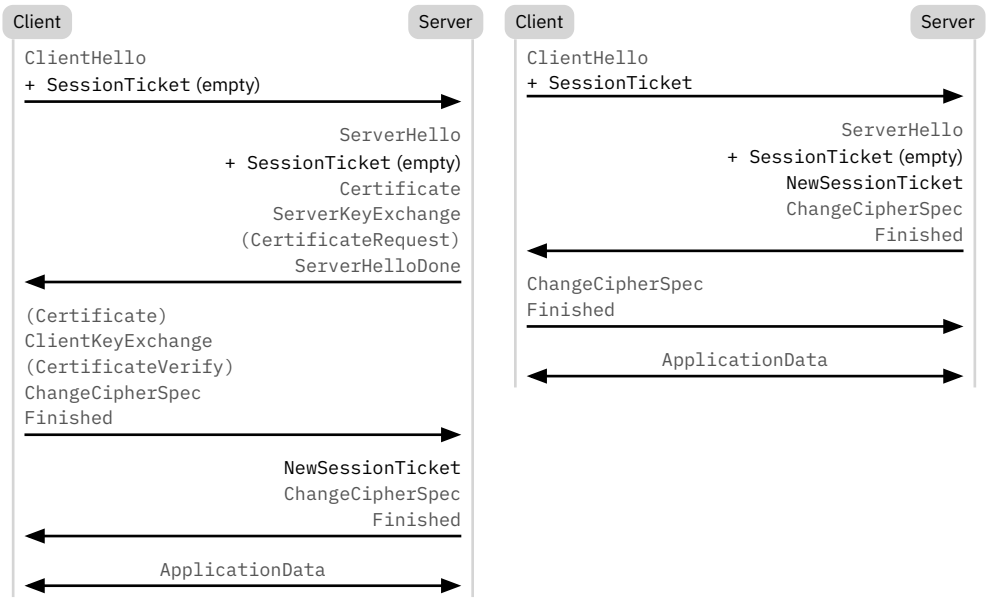
More recently, the QUIC protocol was developed [S104], which is a transport protocol that provides security and reliability similar to TLS. It incorporates TLS 1.3 for security. It is designed to be more efficient than TLS over TCP by using UDP as the underlying transport protocol, while still being reliable. QUIC supports further features like multiplexing multiple streams over a single connection and improved congestion control.

2.3 Session Resumption

During the TLS handshake, both parties use public-key cryptography to establish a shared secret and authenticate each other. This process is computationally expensive and can lead to long connection establishment times. To address this, TLS supports session resumption, which allows a faster connection establishment if both parties have previously communicated and still have the necessary information to establish a secure connection. In TLS 1.2, session resumption skips all public-key operations (key exchange and authentication) and instead derives the shared secret from the stored session information. As a result, the handshake can be completed in one less round trip than a full handshake. In TLS 1.3, session resumption skips the authentication. The key exchange can still be performed to ensure PFS, but both parties can agree to skip this. Further, the client can immediately include application data in the first message of the handshake, allowing for a zero-round-trip-time (0-RTT) resumption.[17] In all versions, the server can reject a resumption attempt and fall back to a full handshake.

Session resumption was initially defined with session IDs [S87, S93, S115]. The idea is that the server can issue a session identifier to the client, which the client can use in future connections to resume a session. This requires the client and server to store session information for each session. Upon resumption, the client sends the session ID in the `ClientHello` message, and if the server recognizes it, both parties can skip the expensive public-key operations and directly derive the shared secret from the stored session information.

17. While TLS calls this 0-RTT, in reality, the parties still need to perform the TCP handshake, which requires one round trip. However, from the view of TLS, the first application data can be sent in the first message. To overcome this limitation, QUIC does not rely on TCP and can truly achieve 0-RTT resumption.



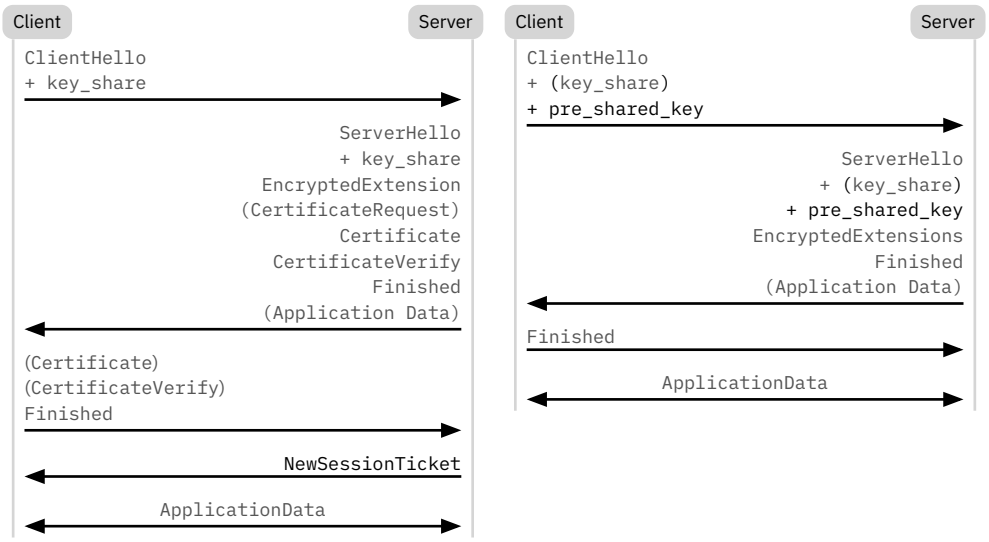
(a) Ticket issuance (b) Ticket resumption

Figure 2: TLS 1.2 ticket issuance and resumption. We gray out the handshake messages that are the same as in the full handshake and highlight changes in black.

However, this approach has scalability issues because the server must store session information for each client. This does not scale well for servers with many clients or for distributed servers where session information needs to be shared between servers.

Session Tickets. To solve the scalability issues of session IDs, session tickets were introduced [S96]. These skip the need for the server to keep state for each client. This is achieved by storing the session information in a *session ticket*, which the server sends to the client in the `NewSessionTicket` message. The client stores this session ticket and the session information. When the client wants to resume a session, it sends the session ticket in the `ClientHello` message. The server can then extract the session information from the session ticket and use it to resume the session without needing to store any state for the client.

By design, a session ticket contains sensitive information and needs to be protected against eavesdropping and tampering. Especially during resumption, the ticket cannot be encrypted by the client because no key exchange has been performed yet. Therefore, the server encrypts the ticket using a key known only to it, the Session Ticket Encryption Key (STEK).



(a) Ticket issuance (b) Ticket resumption

Figure 3: TLS 1.3 ticket issuance and resumption. We gray out the handshake messages that are the same as in the full handshake and highlight changes in black. Note that tickets can be issued at any time after the handshake's completion; that is, tickets can be sent in between or after application data messages.

The client does not need to know this key because only the server needs to extract information from the ticket.

In TLS 1.2, the server issues a ticket in the `NewSessionTicket` message before the `ChangeCipherSpec` message, as shown in Figure 2. That is, the key exchange is completed, but encryption is not yet used. Therefore, the issued tickets are not protected by the shared secret established during the handshake, but only by the STEK. Within the message, the server specifies the ticket and a lifetime hint, indicating how long the ticket is valid. To resume a ticket, the client includes the session ticket in the `SessionTicket` extension of the `ClientHello` message. This extension only includes the ticket and does not provide a mechanism to prevent replay attacks.

In TLS 1.3, tickets are also issued in the `NewSessionTicket` message. The server can send a new ticket at any time after the handshake's completion, meaning the ticket issuance is protected by the shared secret established during the handshake, as shown in Figure 3. The server can send the message multiple times to issue multiple tickets. Each message includes a unique nonce, which results in different keys being derived for this ticket. The client stores this key material as a Pre-Shared Key (PSK)

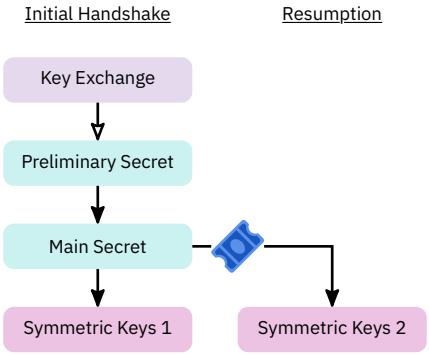


Figure 4: TLS key derivation in TLS 1.2. The solid arrows denote a hash-based one-way function. The output of the key exchange is immediately used as the preliminary secret, which is then used to derive the main secret and the symmetric keys. For simplicity, we omit other inputs for the key derivation, such as the handshake transcript and the client and server random values. Note that we use the updated terminology specified in RFC 9846 [S119: Appendix D].

alongside the ticket. To resume a ticket, the client includes the ticket they want to resume in the `pre_shared_key` extension of the `ClientHello` message. They further prove that they know the PSK associated with the ticket by computing a "binder" value, which they include in the extension.

TLS 1.3 further allows the client to include application data in the first message of the handshake, allowing for a zero round-trip time (0-RTT) resumption. To prevent replay attacks on this application data, the client includes the age of the ticket in the `pre_shared_key` extension. Together with the binder, this allows the server to check whether the client knows the key material and whether the ticket was replayed. To reduce the risk of correlating sessions, the age is obfuscated with a value specified in the `NewSessionTicket` message.

2.3.1 Key Derivation in Session Resumption

TLS does not directly use the key material established in the handshake. Instead, it derives its keys using a hash-based key derivation function, determined by the negotiated cipher suite. The derivation function ensures that the keys are uniformly random and of the correct length for the negotiated cipher suite.

In TLS 1.2, the output of the key exchange is used as the preliminary secret, which is then used to derive the main secret, from which the symmetric keys are derived. Upon resumption, the same main secret is reused

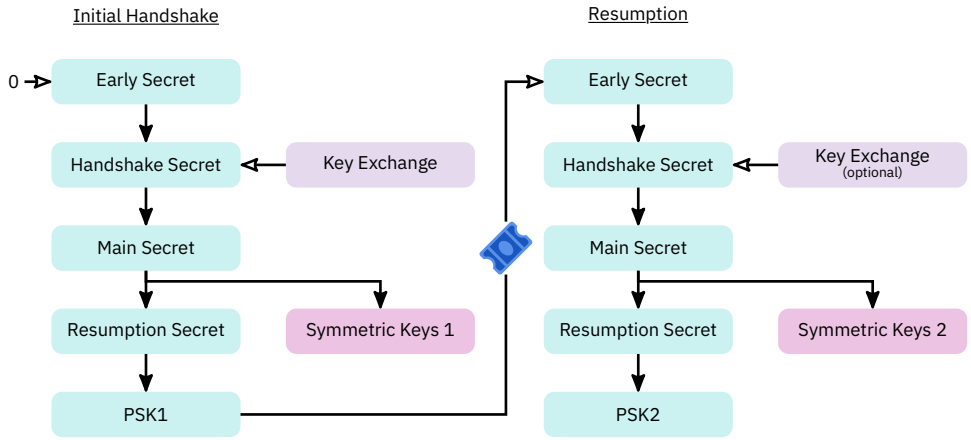


Figure 5: TLS key derivation in TLS 1.3. The solid arrows denote a hash-based one-way function. The hollow arrows denote that this value is used as input to the key derivation function. For simplicity, we omit other inputs and secrets for the key derivation.

to derive the symmetric keys. Note that while the symmetric keys differ between the initial handshake and the resumption, they are derived from the same main secret. We show the key derivation in TLS 1.2 in Figure 4.

In TLS 1.3, the key derivation is more complex. TLS 1.3 starts with the early secret. In a full handshake, the early secret is derived from an empty string. In a resumption, the early secret is derived from the Pre-Shared Key (PSK) that was established in the previous session. Afterward, the handshake is used together with the early secret to derive the handshake secret, after which the main secret is derived. As in TLS 1.2, the symmetric keys are derived from the main secret. However, for resumption, a separate resumption secret is derived. This ensures that even if the resumption secret is compromised, the symmetric keys derived from the main secret are not affected, preserving PFS. For each ticket, a unique PSK is derived from this resumption secret. This ensures that the keys used in each resumption are independent and that compromising one resumption does not compromise any other resumption. We show the key derivation in TLS 1.3 in Figure 5.

2.4 Ticket Construction

The ticket contains security-critical session information, including key material and the client's authentication status. It therefore needs to be protected against attackers. The standard [S96] provides recommendations on how to construct the session ticket but does not mandate a specific con-

```
struct {
    ProtocolVersion protocol_version;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    opaque master_secret[48];
    ClientIdentity client_identity;
    uint32 timestamp;
} StatePlaintext;
```

Listing 1: Recommended ticket plaintext construction [S96].

struction. It outlines how to construct the plaintext within the ticket and how to encrypt it into a ciphertext. However, these are just recommendations. Implementations are free to deviate from these recommendations. This is possible because the client treats the session ticket as an opaque block of data, and only the server needs to be able to understand the ticket.

Session tickets were originally specified as an extension for TLS 1.0 and 1.1 [S96]. Since the extension mechanism was not changed in TLS 1.2, the same extension can be used in TLS 1.2. TLS 1.3 obsoletes the session ticket extension and replaces it with PSKs. Neither TLS 1.2 nor TLS 1.3 specifies how to construct the session ticket, implying that the recommendations from [S96] still apply. In the following, we describe these recommendations for the construction of the ticket. We cover which information is recommended to include in the ticket and how the ticket is encrypted and authenticated to protect it from attackers.

2.4.1 Ticket Plaintext

Within the ticket, the server needs to store information to resume the session with the client. This includes key material and algorithm choices, as well as the client's authentication status. We show the recommended construction of ticket plaintext in Listing 1.

The recommended ticket structure contains the algorithm choices (`protocol_version`, `cipher_suite`, `compression_method`), as TLS 1.2 specifies that a session must be resumed using the same cipher suite and compression method as the original session [S115: Section 7.4.1.4]. While no statement is made about the protocol version, it is reasonable to assume that the same protocol version should be used for resumption.

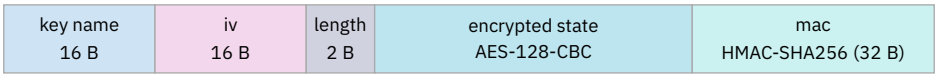


Figure 6: Recommended ticket ciphertext construction.

The standard also recommends storing the key material (`master_secret`) in the ticket. Note that in TLS 1.2 the `master_secret` stays the same for resumed sessions (see Section 2.3.1 for more details). TLS 1.3 derives new key material for each resumed session. There is no statement about which secret to include in TLS 1.3 tickets. It is reasonable to assume that the PSK (`resumption_master_secret`) should be included, as this is the secret used to derive the keys for the resumed session.

The standard also recommends including the authentication information of the client (`client_identity`). If the client was authenticated with a certificate during the handshake, the certificate chain should be included here. The last recommended field is a timestamp, which the server can use to limit the lifetime of the ticket.

2.4.2 Ticket Ciphertext

To protect the ticket against attackers, it needs to be encrypted and authenticated. We show the recommended structure of the ticket ciphertext in Figure 6. The content of the ticket is encrypted using 128-bit AES in CBC mode with a random IV. The key used for this is called Session Ticket Encryption Key (STEK) and should only be known to the server. This key should be rotated regularly to limit the impact of a potential compromise on PFS. When the server receives a ticket, it must first identify the key used to encrypt the ticket before it can decrypt the ticket and retrieve the plaintext. For this purpose, the ticket includes a key name that identifies the corresponding STEK. To protect the ticket against tampering, the whole ticket is authenticated using HMAC-SHA256. The resulting tag is appended to the ticket, such that the server can verify its authenticity before decrypting it. The standard recommends including a length field for the ciphertext. Note that this field is not necessary, as the length of the ciphertext can be derived from the total length of the ticket and the remaining fields, as those have a fixed length.

STEK Terminology. While the name "Session Ticket Encryption Key" implies that it is a single key used only for encryption, it is an umbrella term for all keys used to encrypt and authenticate tickets. This means that, in the recommended construction, the STEK consists of a key for AES-CBC and a key for HMAC-SHA256. Furthermore, the key name can be considered part of the STEK, as it uniquely identifies the key. Implementations may choose to include further metadata in the STEK.

2.5 Scanning Tools and Resources

TLS-Attacker. In our works, we use tools from the TLS-Attacker framework to analyze the security of TLS session resumption and session tickets. TLS-Attacker [O1, C196] was first published by Somorovsky in 2016 [70] as a framework to analyze TLS implementations. It implements the TLS protocol in various versions. It allows defining an arbitrary protocol flow with granular modifications that need not adhere to the standard. This enables researchers to easily implement their attacks and analyze the behavior of TLS implementations. We use it to implement our attacks, retrieve tickets, and send modified tickets to the server.

TLS-Scanner [C198] builds on TLS-Attacker. In this project, many tests are defined that can be run against a server to analyze its security. We implemented our attacks as tests in TLS-Scanner, which allows us to run them against a server and analyze the results. To perform large-scale scans, we use TLS-Crawler [62, C197]. It is a framework that enables parallel scanning of multiple servers using TLS-Scanner.

ZMap Project. We also use tools from the ZMap project [C155] to perform IPv4-wide scans. These tools can handle larger scans than TLS-Attacker but perform shallower tests per host. ZMap is a tool to perform port scans [37, C204]. This way, we can identify servers with open ports, for example, port 443 for TLS. ZGrab2 [34, C203] is a tool to perform a handshake on the application layer. We used it for large-scale scans to retrieve TLS session tickets from servers. ZDNS [50, C202] is a tool to perform multiple DNS queries in parallel.

Tranco List. When scanning the internet, we need to select a set of servers to scan. One approach is to scan all or parts of the IPv4 address space.

Another approach is to select servers based on their popularity. We use the Tranco list [68, W122] to select the servers when evaluating popular servers.

2.5.1 Ethical Considerations

Throughout our scans, we adhered to the ethical guidelines proposed by Durumeric et al. [37]. We performed all scans from a dedicated virtual machine within our university. We cooperated with our ISP and registered our scans with them. They forwarded any abuse reports to us, which we responded to promptly. We explained the purpose of our scans and provided the option to be excluded from further scans by adding their IPs to our blocklist. We set up a reverse DNS entry for our scanning VM, which pointed to a website with contact information and information about the purpose of our scans. For all scans, we ensured that the traffic does not interfere with the normal operation of the network. We further validated our tests locally to ensure that they do not cause any harm to the scanned servers. We disclosed any findings to the affected parties, which we outline in the respective sections of this thesis.

Chapter 3
Real-World Ticket Ciphertext Construction

TLS session tickets must contain all information a server needs to resume the session. This notably includes key material and other parameters chosen during the initial handshake, such as the cipher suite. During the resumption, the ticket is sent before any key exchange has been performed. Therefore, it cannot be encrypted by the client. It is important that an attacker observing the ticket cannot learn any information about the key material. Additionally, since the client holds the ticket, it is important that the server can verify that the ticket was not tampered with and that the server itself issued it. Therefore, RFC 5077 [S96] recommends that the ticket should be encrypted and authenticated using a key known only to the server, the Session Ticket Encryption Key (STEK).

The STEK is a symmetric key that is used to encrypt and authenticate the ticket. The server should rotate the STEK regularly to minimize the impact of a potential compromise. This has been criticized before because it affects the PFS guarantees of TLS. Previous works have found that over a third of servers rotate their STEKs less than daily [71].

If the STEK is compromised, the security guarantees of TLS are completely undermined. A passive attacker can decrypt the ticket and learn the session key. In TLS 1.2, this allows the attacker to decrypt the session in which the ticket was issued and all resumed sessions. The same risk exists even if the ticket was never resumed, as the ticket is sent before the exchanged keys are used to encrypt the connection. Valsorda [W147] previously criticized this limitation. In TLS 1.3, this has been improved. TLS 1.3 optionally allows performing a key exchange during a resumption. If this key exchange is performed, a passive attacker can decrypt only data preceding it. In all versions, an active attacker can impersonate the server because the server is not re-authenticated with a certificate during the resumption. Instead, authentication is delegated to the ability to resume the session. If an attacker knows the STEK or, more generally, can extract the key material from the ticket, they can use it to resume a session with the client, impersonating the server.

As the session ticket mechanism is critical for the security of TLS, the ticket must be constructed and handled correctly. This leads to the following research question:

RQ1: Which cryptographic vulnerabilities may be introduced through faulty TLS session ticket implementations?

To answer this question, we perform the first systematic analysis of the session ticket mechanism. One motivation for this analysis is the security vulnerability in GnuTLS discovered by Klute in 2020 [W136]. The bug caused the STEK to be all zeroes when starting the server [W138, W150]. The server would eventually rotate the STEK, after which it would behave correctly. However, all sessions issued during the first all-zero STEK can trivially be decrypted by an attacker. We describe this and other potential pitfalls in Section 3.2.

Based on these theoretical issues, we can evaluate whether they occur in practice. However, because the ticket format is not strict, we need to analyze how tickets are constructed in practice before we can evaluate deployed servers:

RQ2: How do state-of-the-art open-source libraries implement session tickets?

We analyzed 14 open-source TLS libraries in Section 3.3 and found that they generally behave securely. However, we also found that none of the libraries exactly follows the recommendations of RFC 5077 [S96]. These divergences range from omitting the length field or changing the algorithms to including completely new fields in the ticket. Using the insights from the analysis of open-source libraries, we can construct tests to analyze real-world TLS servers:

RQ3: Are real-world TLS servers vulnerable to the proposed attacks?

We performed a large-scale scan of deployed TLS servers in Section 3.5. We evaluated whether servers are vulnerable to the proposed attacks and found that over 1.9 % of servers on the Tranco top 100k occasionally used an all-zero STEK. In further scans, we discovered partially initialized

STEKs and a reused keystream. These issues completely undermined all security guarantees of TLS for the affected sessions and would allow an attacker to decrypt them.

Contributions.

- *Attacks*: We systematically analyze cryptographic implementation pitfalls in session ticket handling of TLS servers and their impact on TLS connections (Section 3.2).
- *Open-Source Analysis*: We conduct a source code analysis of 14 open-source TLS libraries and 5 open-source HTTPS servers. We study their usage of cryptographic algorithms, session ticket formats, and vulnerability mitigations (Section 3.3).
- *Large-Scale Analysis*: Based on our analyses, we implement online and offline tests for the potential vulnerabilities (Section 3.4) and perform a large-scale scan of the TLS ecosystem (Section 3.5). Our scans detected critical security issues in the collected session tickets, allowing attackers to decrypt the TLS communication. Most notably, we discovered an issue in a large portion of AWS hosts, affecting around 1.9 % of the Tranco top 100k hosts.

Disclaimer on Originality

This chapter is mostly based on [O2]. For more details, see Section 1.5.1.

3.1 Background on Encryption Ciphers

Some of our proposed pitfalls are specific to certain types of encryption ciphers. In the following, we provide a brief background on the different types of encryption ciphers and known vulnerabilities.

3.1.1 Block Ciphers and Padding Oracles

Block ciphers are encryption schemes that encrypt fixed-size blocks of plaintext into ciphertext. If the plaintext is not a multiple of the block size, the cipher cannot be applied directly. To solve this, the plaintext is padded to a multiple of the block size.

In this chapter, we consider two padding schemes: PKCS#7 [S101] and the padding defined in TLS 1.2 [S115]. Both work similarly: They both append the byte n times until the plaintext is a multiple of the block size. In PKCS#7, n is equal to the number of bytes appended. In TLS 1.2, n is

equal to the number of bytes appended minus one. That is, a three-byte padding in PKCS#7 would be `030303`, while in TLS 1.2 it would be `020202`.

CBC Padding Oracle Attacks. Vaudenay [80] presented an attack on CBC mode encryption schemes that use padding. The attack allows an attacker to decrypt a ciphertext without knowing the key. It abuses the fact that the ciphertext is malleable and the server may respond differently to a ciphertext with valid padding than to one with invalid padding. In CBC, a block C_n is decrypted by computing $P_n = \text{Dec}_k(C_n) \oplus C_{n-1}$, where Dec_k is the decryption function, and C_{n-1} is the previous ciphertext block. If an attacker induces a bit flip in C_{n-1}, the plaintext P_n will contain the same bit flip.

In a padding oracle attack, the attacker uses the different responses from the server as an oracle to determine whether the padding is valid, as this gives the attacker information about the plaintext. The attacker will first try to find a ciphertext with a one-byte padding. For this, they induce bit flips in the last block of C_{n-1}. When the server accepts the ciphertext, the attacker knows that the last block contains a valid padding. To validate whether this is a one-byte padding, the attacker will induce a bit flip in the second-to-last byte of C_{n-1}, as this would invalidate any longer padding than a one-byte padding. If the server still accepts the ciphertext, the attacker knows that the last block contains a one-byte padding. The attacker can then continue this procedure to find a ciphertext with a two-byte padding until they have recovered the entire block.

3.1.2 Stream Ciphers

Stream ciphers are encryption schemes that allow the encryption of plaintext of arbitrary length. They are usually constructed by generating a keystream from a secret key and XORing it with the plaintext to produce the ciphertext. Common examples of this are ChaCha20 or cipher modes like GCM, CCM, or CTR. These use the key to generate a keystream K_s and compute the ciphertext as $C = K_s \oplus P$.

It is important to ensure that the same keystream is never used to encrypt different plaintexts. An attacker knowing a plaintext P_a and a ciphertext C_a can recover the keystream $K_s = C_a \oplus P_a$. If another plaintext P_b is encrypted with the same keystream K_s into ciphertext C_b, the attacker can recover the plaintext from the ciphertext $K_s \oplus C_b = P_b$.

3.2 Cryptographic Pitfalls

The format of session tickets is not strict [S96], and servers may deviate from the recommended format. Since the STEK is only known by the server, the client cannot check whether the server is using the recommended format. A server deviating from the recommended ticket design or making an implementation flaw would hence not be apparent to a client. However, it could have severe consequences for the security of the connection. In the following, we discuss potential deviations from the recommended format, their consequences, and potential cryptographic implementation pitfalls. All of these flaws allow an attacker to recover the secrets from the ticket, for example, the main secret.

3.2.1 List of Pitfalls

Unencrypted Session Tickets. The first possible vulnerability can appear if a server issues session tickets with an unencrypted session state. Such configuration mistakes would allow attackers to access the session secrets directly. While the vulnerability seems contrived, the session ticket mechanism would work perfectly fine with unencrypted session states. The session ticket structure is opaque to the client, and it would not notice if connection secrets were transmitted within it.[18] Similar issues have been found in the context of S/MIME and OpenPGP; they led to sending plaintext emails even when encryption was turned on [W128, W143].

Weak Encryption Keys. Inspired by the bug in GnuTLS [W136, W138, W150], another vulnerability can appear if a vulnerable server uses default keys or keys with low entropy to protect its session tickets. This vulnerability might be caused by uninitialized memory and could lead to session tickets encrypted with all-zero keys. This would allow an attacker to decrypt tickets and extract the session secrets.

Reused Keystream. If implementations deviate from the proposed encryption algorithms, algorithm-specific issues may appear. An example of this is keystream reuse in ciphers like ChaCha20, or in cipher modes like GCM, CCM, or CTR. This mistake was previously observed in some TLS servers using AES-GCM [24]. As outlined in Section 3.1.2, an attacker

18. Independent of our publication in 2023 [O2], this vulnerability has been implemented satirically in “httpv”: https://tom7.org/httpv/httpv.pdf (accessed: 2026-07-19).

knowing a plaintext and a ciphertext can recover the keystream and use it to decrypt other ciphertexts encrypted with the same keystream. The attacker can abuse a server that occasionally encrypts session tickets with repeating keystreams by repeatedly requesting session tickets until a session ticket with a keystream used in a victim's session ticket is received. The decryption itself does not tell the attacker directly whether the attack was successful. However, the attacker can verify the success by checking whether the recovered secret is suitable to decrypt the victim's session.

Cryptographic Wear-out. Depending on the selected algorithms and the number of session tickets expected to be issued with a given STEK, a problem called cryptographic wear-out can appear. Typically, ciphers have a threshold of ciphertext that can be generated under a given key until they become insecure. For example, for AES-GCM, the nonce should never be repeated, as using the same nonce under the same key with different plaintexts leaks the XOR of the plaintexts and the authentication key [24]. The nonces in AES-GCM are typically 12 bytes long, meaning that randomly chosen IVs will collide with 50 % probability after 2^{48} session tickets. Since 50 % is typically not conservative enough to be considered secure, real-world systems additionally require a security margin. With an acceptable collision risk of $\frac{1}{2^{32}}$, AES-GCM allows for the encryption of 2^{32} ($\approx$ 4.2 billion) session tickets. This limitation has also been set by the National Institute of Standards and Technology (NIST) [S94]. Note that NIST also recommends this for any other nonce length. Depending on the longevity of the STEK and the server infrastructure, it is possible that this limit gets exhausted, which can result in a nonce collision, allowing the users of affected sessions to decrypt the other colliding session. For AES-CBC with HMAC, cryptographic wear-out is not of practical concern as the block and the initialization vector sizes are both 16 bytes, allowing for the encryption of 2^{48} session tickets before the threshold is passed, which is unlikely to ever be hit by a real server, even if the STEK is never rotated.

Nonce collisions are not the only threat one needs to consider when using authenticated encryption schemes. Other security properties, like integrity or indistinguishability, also degrade with increasing data sizes and rotated encryption keys [15, 29, 48, 59]. However, for session tickets, these effects are of limited practical relevance: tickets are short, and nonce-collision limits are reached earlier in realistic deployments. In practice,

nonce-reuse risk therefore becomes relevant before these broader degradations do.

Weak Algorithms. If the server uses a weak algorithm, different attacks to recover the plaintext may be possible. For example, the algorithm could use short keys (e.g., 56 bits in the case of DES), allowing an attacker to bruteforce the STEK.

Other Algorithm-Specific Weaknesses. Other attacks might be feasible depending on the algorithm. For example, partitioning oracle attacks against AES-GCM and ChaCha20-Poly1305 may allow an attacker to recover the STEK. We explore these in Chapter 5.

Broken Authenticated Encryption. An implementation does not only need to encrypt the session ticket, but it must also ensure its integrity and authenticity. It may use different algorithms than suggested or may even decide not to authenticate tickets at all, for example, by omitting the HMAC or its validation. Even if authentication is enforced, similarly to weak encryption keys, servers might use weak HMAC keys. This would allow an attacker to recompute correct HMACs over modified session tickets, rendering authenticated encryption useless.

Broken authenticated encryption does not directly lead to session ticket decryption. However, it can open the door to new potential attacks, notably decryption side channels.

Decryption Side Channels. Depending on the structure of the session ticket, an implementation may be vulnerable to side-channel attacks. For example, RFC 5077 [S96] recommends the usage of AES-CBC for session ticket encryption along with HMACs in the Encrypt-then-MAC scheme. If the ticket is not properly authenticated, an attacker can modify the ciphertexts and make the implementation vulnerable to a padding oracle attack [38, 70, 80]. The padding oracle attack exploits the malleability of the CBC mode of operation and strict padding validation by the receiving application. It allows the attacker to query the server with carefully modified ciphertexts. If the server leaks information about the padding validity, the attacker can decrypt the plaintext and thus the session ticket after a few hundred queries.

Note that even if authentication with HMACs is enforced, a server might still reveal a padding oracle vulnerability when using MACs in

an insecure way. For example, servers can use the MAC-then-Encrypt or MAC-and-Encrypt scheme, whose application makes secure padding oracle prevention very challenging [38, 62, 70]. MAC-then-Encrypt was used in TLS-CBC in TLS 1.2 by default [S98], making it tempting for implementations to reuse existing code.

The side channel attacks can take various forms. The server may respond with different messages, or it may take different amounts of time to respond, depending on the validity of the ticket. Such timing side channels are hard to exploit for a remote attacker due to the noise introduced by network nodes [25, 38, 76].

3.2.2 Impact on TLS Sessions in Different TLS Versions

Most presented pitfalls allow an attacker to decrypt the session ticket and, therefore, retrieve the main secret. However, due to the design differences of session resumption between TLS 1.2 and TLS 1.3, the impact differs between the two versions. If the attacker obtains the session secrets stored in a TLS 1.2 session ticket, they can decrypt all sessions in which the compromised session ticket was either issued or redeemed. In TLS 1.3, an attacker can decrypt the 0-RTT data of all sessions where the compromised session ticket was redeemed. If no additional key exchange is performed, the following application data can also be decrypted. The impact also depends on the secret contained in the ticket. If the ticket contains the main secret instead of the PSK, the session in which the ticket was issued can be decrypted.

For some of the presented attacks, a passive attacker suffices to break the security of the TLS session, allowing them to eavesdrop on the exchanged application data. An active attacker can impersonate the server when the client resumes the session in all TLS versions.

If the attacker can also forge new tickets (for example, due to a STEK compromise), they can also set the client identity stored in the ticket. This may allow them to circumvent client authentication or assume another identity.

3.3 Open-Source Analysis

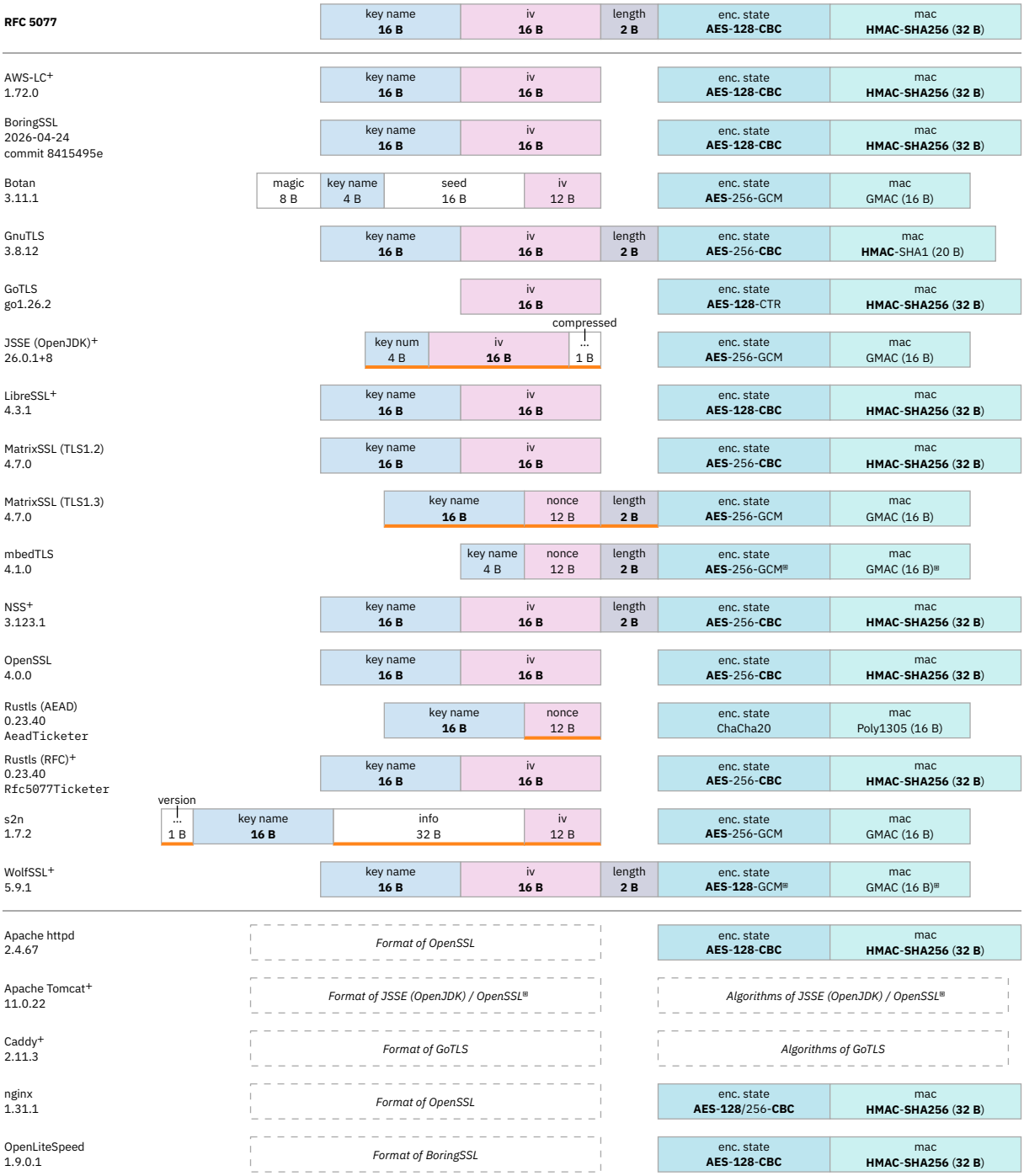
To start our evaluation, we looked at the session ticket implementation in common open-source TLS libraries. We manually analyzed the source

code of TLS libraries to determine their session ticket format. We further evaluated whether the potential implementation flaws proposed in Section 3.2 apply to the libraries. In total, we evaluated 14 different TLS libraries and five web servers. While these web servers also use libraries covered in our open-source analysis, they might behave differently due to how they use them. Our goal was to better understand how different servers handle session tickets and whether they comply with the recommendations of RFC 5077 [S96]. We summarize the results in Figure 7.

Encryption & Authentication Algorithms. We first examine the encryption and authentication algorithms used by the different implementations. RFC 5077 recommends using *AES-128-CBC* to encrypt the tickets and *HMAC-SHA256* to authenticate the tickets (including the encrypted contents). All tested libraries use authenticated encryption with strong algorithms. Note that GnuTLS uses *HMAC-SHA1*. While still secure, this is weaker than the recommended *HMAC-SHA256*. Most libraries diverge from the recommended algorithms and use different modes. Rustls is the only library that does not use AES and instead uses ChaCha20. Furthermore, nearly half of the analyzed libraries use an AEAD algorithm rather than an HMAC algorithm for authentication. For the web servers, Apache Tomcat and Caddy do not specify algorithms and use the default configuration of their underlying libraries. Apache httpd chose to use AES-128-CBC instead of AES-256-CBC (OpenSSL's default), while nginx supports both key lengths. OpenLiteSpeed uses BoringSSL by default and uses the same algorithms as BoringSSL, but sets them explicitly in their code.

We found that nearly all libraries that use an HMAC for authentication use Encrypt-then-MAC to authenticate the ticket. The only exception is WolfSSL, which uses Encrypt-and-MAC. As it also uses AES-CBC, it might seem vulnerable to padding oracle attacks. However, it does not use any padding and therefore is not vulnerable to such attacks.

The majority of libraries authenticate all fields, as recommended by RFC 5077 [S96]. However, we found some exceptions. JSSE (OpenJDK), due to a programming mistake, and MatrixSSL (TLS 1.3) do not add any additional authenticated data. They only authenticate the encrypted state. s2n does not authenticate the ticket version, info, or nonce. It only



+: Added library in this thesis (compared to [O2]).

⊞: The library can be configured to use different algorithms:

- mbedTLS can use any AEAD algorithm from the PSA API [S88].
- WolfSSL supports a variety of algorithms depending on the compile-time configuration: AES-128/256-CBC + HMAC-SHA256/384/512, ChaCha20 + Poly1305, AES-128/256-GCM, SM4. On our system, the automatic configuration uses AES-128-GCM (`./configure --enable-session-ticket`).
- Apache Tomcat can be configured to use JSSE or OpenSSL for TLS. It uses the format and algorithms chosen by the respective library.

Figure 7: Ticket format of evaluated TLS libraries and web servers. Similarities with the session ticket format according to RFC 5077 are highlighted in bold. All libraries use AEAD or HMAC to authenticate the ticket. Any fields that are not authenticated are underlined in orange. In this thesis, we updated all libraries and servers to the latest versions at the time of writing.

authenticates the key name and a variable called `implicit_aad`, which is stored with the STEK. This data consists of twelve randomly generated bytes. These bytes are generated together with the STEK and are never transmitted. This way, even if an attacker knows the AES key, they cannot forge a valid ticket without knowing the `implicit_aad` value. Rustls (AEAD) does not authenticate the nonce explicitly. However, in AES-GCM, the nonce is already authenticated due to the internal construction.

Session Ticket Format. Next, we evaluate the session ticket format and compare it to the recommended session ticket format specified in RFC 5077 (see Figure 7). The web servers use the format of the underlying libraries. Most libraries follow the ticket format recommendation. However, only six libraries include the size of the encrypted state within the ticket. The encrypted state is the only field with a variable size in the ticket; hence, the format is unambiguous, as the size of the complete ticket is specified in the TLS ticket extension. Most deviations are simple changes in the length of the IV or MAC because they use different algorithms.

Four libraries that notably diverge from the recommendation: Botan, JSSE (OpenJDK), GoTLS, and s2n. Botan and s2n include a static prefix to identify the ticket format version. Both also include a field between the key name and IV, which they use to derive a fresh key from the long-term STEK. This fresh key is used to authenticate and encrypt the ticket. GoTLS omits the key name completely to improve privacy [C199]. During decryption, it tries to decrypt the ticket with all available STEKs until one succeeds. JSSE (OpenJDK) includes a flag indicating whether the ticket plaintext is compressed with GZIP.

Cryptographic Wear-out. We discovered that for some libraries, cryptographic wear-out is a potential danger. JSSE (OpenJDK), MatrixSSL (TLS 1.3), mbedTLS, and WolfSSL use AES-GCM with random IVs to encrypt their session ticket, meaning that these implementations should only encrypt up to 2^{32} session tickets under a given STEK. The situation is similar for Rustls (AEAD), which uses ChaCha20 with a random IV to encrypt its session tickets. For the same reasons as with GCM, ChaCha20 should only be used to encrypt up to 2^{32} session tickets until the threshold is reached. Whether it is realistic to reach this limit depends on the key rotation policy and the load the server can handle.

We deem the other implementations not to suffer from wear-out. At first glance, Botan and s2n also seem to suffer from the issue, but a random seed is used to construct a short-term STEK that drastically reduces wear-out. The GoTLS implementation uses AES-CTR with a random IV. This allows GoTLS to encrypt up to 2^{48} blocks with the same key until the threshold is reached. Since session tickets are typically small, it is unlikely that the block limit is exceeded before key rotation happens.

3.4 Scanning and Evaluation Methodology

While in a lab setting only a limited number of servers can be examined, a much bigger range of servers can be evaluated in a real-world setting. Previous research has shown that real-world evaluations of TLS servers yield interesting new insights [12, 22, 23, 35, 36, 45, 53, 61, 62, 78, 79]. Since the session ticket format can be chosen arbitrarily by the server, the major challenge was to design black-box tests for a limited evaluation of session tickets suitable for large-scale studies on the TLS ecosystem.

Based on our evaluation of open-source libraries, we implemented tests for a subset of the implementation pitfalls and vulnerabilities mentioned in Section 3.2 using TLS-Scanner. We divided the tests into two categories: online and offline tests. Online tests demand active modifications of session tickets during the scan, while offline tests only require access to collected session tickets and metadata. Cryptographic wear-out cannot reasonably be evaluated with black-box tests and is therefore outside the scope of this analysis. Timing side-channel attacks have also been excluded, as they require either very precise or many measurements per host to obtain accurate results.

3.4.1 Online Server Scans

Session Ticket Support. For each tested TLS version, we first check whether a server issues and resumes tickets. For this, we perform an initial handshake and check whether it contained a ticket. If it contained a ticket, we perform ten additional handshakes to collect more tickets. Should the server not issue tickets in all connections, we consider it to only support tickets partially. For servers that partially support tickets, we do not perform further online tests, as we presume the results will be inconsistent. We still store the tickets for these servers for the offline

tests. With the session tickets, we also store the corresponding secrets used to derive keys for each connection (cf. Section 2.3.1). For TLS 1.2, these are the preliminary secret and main secret. For TLS 1.3, the secrets are the handshake secret, main secret, resumption secret, and PSKs. To test whether the ticket mechanism is functional, we also check whether the server accepts one issued ticket.

Missing Authenticity Protection. To test the authentication and integrity algorithms, we perform tests where we manipulate the session ticket. As shown in our open-source analysis (cf. Figure 7), most libraries protect every session ticket byte using a MAC. Therefore, any session ticket modification should lead to ticket invalidation. To test whether this is correctly implemented, we induce bit flips in the ticket and send it to the server. For each byte in the ticket, we induce a single bit flip and observe the server's behavior. If the server accepts any ticket, we know it does not properly check the ticket's authenticity.

Padding Oracle Attacks. While TLS-Scanner provides tests for TLS padding oracles [80], these cannot be applied to our use case. These tests assume that TLS-Scanner possesses symmetric encryption keys and thus can precisely encrypt the ciphertext, allowing it to include malformed padding on purpose. In our case, we do not know which padding is used, which block size is used, or where the ciphertext in the ticket ends. Based on our analyses in Section 3.3, we assume servers use a format where the ticket contains the ciphertext and MAC as the last fields (see Figure 8).

To detect a CBC padding oracle vulnerability, we need to modify the second-to-last ciphertext block $n-1$, as outlined in Section 3.1.1. Based on the assumed format, we combine possible MAC lengths (0, 16, 20, 28, 32, 48, 64) with possible block lengths (8, 16) to get possible positions for block $n-1$. To get the last byte, we need to modify this block such that the last block n contains a valid one-byte padding. To this end, we XOR the last byte of block $n-1$ with each guessed plaintext p and a one-byte padding. If we guessed correctly, the last block will contain a valid one-byte padding, and a vulnerable server will behave differently.

We consider two possible padding schemes: PKCS#7 [S101] and padding as defined for block ciphers in TLS 1.2 [S115]. To speed up our

attack, we assume the last block already contains padding. Therefore, we have fewer possible plaintexts to test.

We send each modified ticket to the server and observe its behavior. We observe differences in the message types, how messages are split into TLS records, and the state of the TCP socket. If a difference is observed, we confirm the vulnerability by guessing the second-to-last byte. For this byte, we do not assume a specific value but test all possible byte values. If the behavior is again different, we consider the server vulnerable.

We send each ticket multiple times to counteract independent behavior changes being detected as behavior differences to our padding changes. We initially send each ticket twice. If a behavior difference is detected, we send each ticket again to check for statistical significance. For the last byte, we send the ticket eight additional times, and for the second byte, two additional times. We use the statistical tests from Merget et al. [61] to test whether the behavior difference was significant ($p < 0.05$).

3.4.2 Offline Analysis

The last group of tests needs no further interaction with the server. First, we perform basic analyses of the retrieved tickets. We store the length of every session ticket. Next, we attempt to identify prefixes, such as the key name, and continue analyzing cryptographic issues.

Detecting Prefixes. To detect the prefixes, we build a prefix tree of the received tickets. The root node represents an empty string at depth zero. For each ticket, we add each byte to the tree. After the tree is built, we sum up the degrees for each depth. That is, we count how many divergences there are at each position in the tickets. We consider the depth with the highest degree to be the prefix length. This approach should detect the prefix length, even if we receive tickets with different key names, for example, due to a load balancer.

Unencrypted Tickets. The first issue we check is whether the tickets are encrypted at all. For this, we iterate over each ticket and check whether any of the secrets we stored with the ticket are directly present in the ticket bytes. If we find that a secret is present, we consider the server vulnerable.

Reused Keystream. Another issue we analyze is the reuse of keystreams. To check whether a keystream was reused, we take two tickets and their

corresponding secrets. Let $T_1 = K_s \oplus p_1$ and $T_2 = K_s \oplus p_2$ be two tickets, where K_s is the reused keystream used to encrypt the plaintext p_i for each ticket. To check whether a keystream was reused, we XOR their bytes: $T_x \coloneqq T_1 \oplus T_2 = K_s \oplus p_1 \oplus K_s \oplus p_2 = p_1 \oplus p_2$. We then XOR the secrets corresponding to these tickets, for example, the main secrets ms_1 and ms_2: $\mathsf{ms}_x \coloneqq \mathsf{ms}_1 \oplus \mathsf{ms}_2$. If the XOR value of the tickets $T_x = p_1 \oplus p_2$ contains the XOR value of the secrets ms_x, we know the tickets used the same keystream. We repeat this for each pair of tickets received from the server and their corresponding secrets.

3.4.2.1 Weak Keys

To test the issued tickets for the usage of weak keys in encryption and authentication algorithms, we created a list of possible key candidates. This list contains keys with repeating byte values (e.g., `0x000000`...) and keys with consecutively increasing bytes (e.g., `0x000102`...). We chose these specific keys to account for manually set or hard-coded keys that were chosen by humans or might have been copied from code examples. In addition, to target keys possibly constructed from uninitialized memory, we included known debug values from different applications [W149], for example, uninitialized heap memory for applications compiled with Microsoft's C/C++ in debug mode (`0xCDCD`...) [W125]. Inspired by prior findings that NIST specification example keys have appeared in production deployments [W133], we also included such keys from the AES NIST specification [S113]. In total, our list includes 144 different weak keys. We list our selection of keys in Appendix A.2.

We use our list of weak keys for extensive tests of the retrieved tickets. For the default encryption key, we attempt to decrypt the given ticket and test whether the decrypted plaintext contains one of the secrets corresponding to the ticket. For the default HMAC key, we compute the HMAC tag and test whether the ticket contains the resulting tag. To perform these tests, we consider two additional parameters: the algorithm and ticket format.

Algorithm. In our source code analysis (see Figure 7), we observed eight different encryption algorithms and five different authentication algorithms. We extended the list of encryption and authentication algorithms with different key sizes. This resulted in a total of 15 encryption

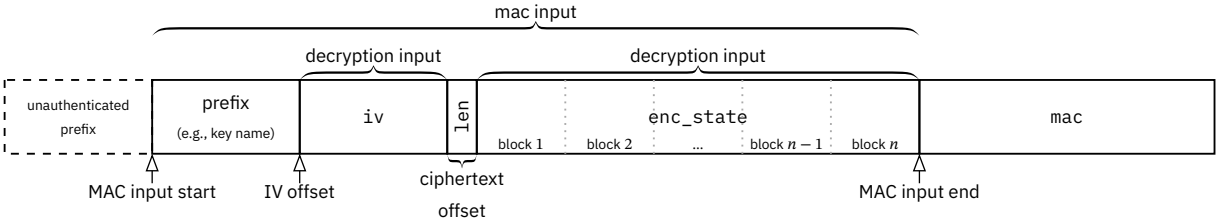


Figure 8: Assumed ticket format. The `enc_state` is split into blocks in case a block cipher is used. The correct inputs for the MAC and decryption are annotated.

algorithms and 5 HMAC algorithms. We provide a complete list of the used algorithms in Appendix A.2.

Ticket format. As we perform a black-box evaluation, we do not know the structure of a received ticket. In our open-source analysis, we have seen that libraries generally follow a similar format. We do not know which format closed-source libraries use. Nonetheless, we make the following assumptions about the format of received tickets: We assume that the ticket might start with some prefix, like the key name. Additionally, we allow for a prefix that is not authenticated using the MAC. Next, we have an IV, followed by an optional length field (`len`), the ciphertext, and the MAC. The assumed format is shown in Figure 8.

In our implementation, this translates into the following constraints: When checking for a default HMAC key, we consider two offsets describing the HMAC input start and the HMAC input end. We assume that the prefix and suffix lengths are multiples of 8 bytes. Further, we assume that the prefix length is between 0 and 128 bytes and the suffix length is between 16 and 64 bytes. This results in up to 126 combinations.

When checking for a default encryption key, we consider two offsets pointing to the IV/nonce and to the ciphertext, respectively. We assume the IV starts at any position in the range of 0 to 128 bytes. For the offset of the ciphertext, we assume that it starts immediately after the IV or two bytes after it.[19] We also allow for an IV set to all zeroes that is not included in the ticket. This results in up to 434 considered formats.

19. This does not cover JSSE (OpenJDK) in its current version, which uses a one-byte offset. However, before 2025 (commit `99048c3`), the offset was 0 bytes, which was covered.

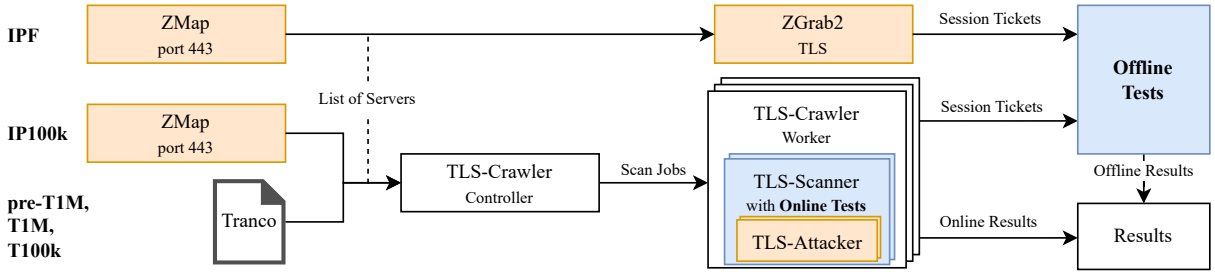


Figure 9: Setup for our scans. Blue parts contain our contribution. Orange parts are responsible for communicating with other servers on the internet.

3.5 Large-Scale Evaluation

Scan Setup. In Figure 9, we give an overview of the tools we used in our scans. We performed multiple scans against hosts on the Tranco list and publicly available IPv4 hosts. This mix represents the most popular websites as well as random, smaller TLS servers, covering a variety of implementations. We used two approaches for our scans, which we describe in the following. We then describe the scans we performed.

Tooling Approaches. Our first approach uses TLS-Crawler to perform the scans. We first determined which hosts to scan using the Tranco list [68] or ZMap [37, C204] to find random IPv4 hosts that responded to a TCP SYN on port 443. These were fed into TLS-Crawler, which is split into a controller and workers. The controller creates a list of scan jobs that are handled by worker instances. The workers run TLS-Scanner, where our online tests are implemented, against multiple servers simultaneously. To connect to the server, our tests utilize TLS-Attacker. After the online tests are run, the session tickets, including the session key material, are passed to the offline tests to analyze. All results are collected in a database.

Our second approach, for larger scans, did not use TLS-Crawler. Instead, we used ZGrab2 [34, C203], as it is better suited to scans of the whole IPv4 space. ZGrab2 stores the session tickets and corresponding key material, which we fed into our offline tests to generate results for these hosts.

T1M (and pre-T1M). We performed the first scan of the Tranco top 1M in May 2021 to gain first insights into the session ticket ecosystem. This scan only covered a subset of our proposed vulnerabilities. We only scanned for keys consisting exclusively of zeroes in TLS 1.2 and 1.3. The other

vulnerabilities and keys were envisioned after this scan and hence were not scanned here.

Before performing the final T1M scan, we performed test scans, subsequently summarized as the pre-T1M scan. The pre-T1M scan is an artifact of our research workflow meant to verify the first version of our scanner. Unexpectedly, our scanner quickly detected a large portion of Amazon servers vulnerable to zero-key flaws (cf. Section 3.5.2). Due to the severity of the discovered issue, we had to notify Amazon immediately before we could extend the original study with further ideas. As the issue fix would alter the results of subsequent scans, the pre-T1M entry presents the findings of this initial scan.

T100k and IP100k. We performed two smaller but more detailed scans of 100k hosts each in April 2022. These were chosen as the top 100k from the Tranco list and 100k random IPv4 hosts that responded on port 443. For these, we performed extensive online and offline tests to assess their relevance. We tested TLS versions 1.0 through 1.3 for each host.

IPF. We scanned the entire IPv4 address range in August 2022. Since earlier scans did not reveal any issues in the online tests, we performed only offline tests. Evaluating only the offline tests further saves resources not only on our end but also on the scanned servers. To this end, we only collected session tickets and performed the offline tests afterward. This also means that we do not try to resume tickets. We performed three TLS connections per host using ZGrab2, each time attempting to obtain a session ticket. This is a reduction from the previous scans, where we performed ten connections to collect tickets. We chose this number to reduce the number of connections while hopefully still connecting to different servers if there is a load balancer. Note that ZGrab2 did not support TLS 1.3 at this time. Therefore, this scan only contains data up to TLS 1.2.

IPmail. The last scan we performed was not part of the original study. It was performed after the publication of our work. We scanned the whole IPv4 address space for email servers. We scanned IMAP on ports 143 and 993, POP3 on ports 110 and 995, and SMTP on ports 25, 465, and 587. Similar to the IPF scan, we only collected session tickets and performed the offline tests afterward. However, compared to the IPF scan, we performed only one connection to further reduce the load on the scanned servers.

Table 1: Results from our scans. We performed the pre-T1M and T1M scans to retrieve a first insight into the session ticket ecosystem. Subsequently, we added tests for keystream reuse and introduced the online analysis. For IPF and IPmail, we performed an offline analysis on tickets obtained with ZGrab2 and tested only the highest protocol version, up to TLS 1.2, supported by the server.

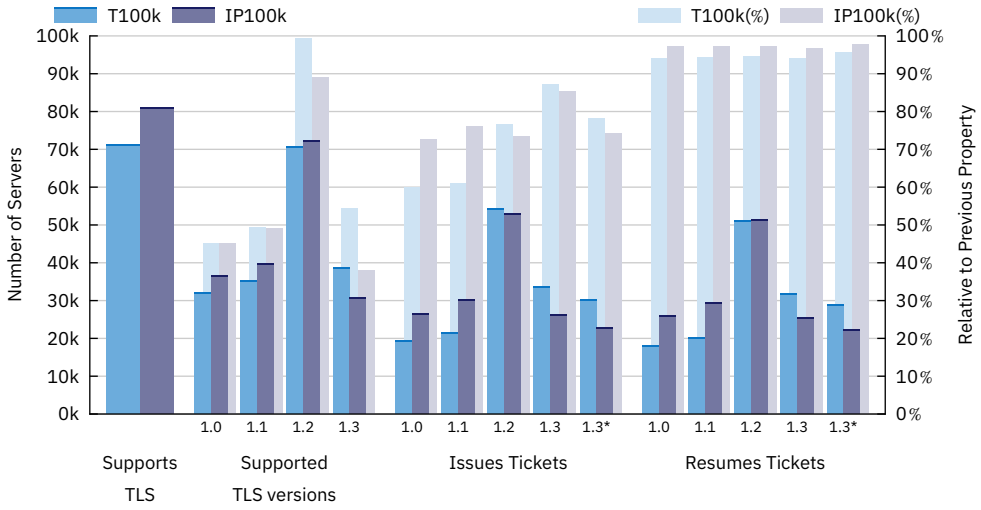
			Statistics			Offline Analysis			Online Analysis	
Scan	Date	Tested Versions	Supports TLS	Issues Ticket	Resumes Ticket	Unencrypted Ticket	Weak STEK	Reused Keystream	Missing Auth. Protection	Padding Oracle
pre-T1M	2021-04	1.2	66,992	53,059	–	0	1,923	–	–	–
T1M	2021-05	1.2–1.3	760,293	594,238	547,159	0	3	–	–	–
T100k	2022-04	1.0–1.3	71,200	58,069	55,003	0	1	0	0	0
IP100k	2022-04	1.0–1.3	80,972	57,493	55,969	0	0	0	0	0
IPF	2022-08	≤ 1.2	39,390,365	29,621,531	–	0	189	1	–	–
IPmail	2025-09	≤ 1.2	14,856,928	10,549,464	–	0	0	–	–	–
… SMTP	2025-09	≤ 1.2	4,931,383	1,227,871	–	0	0	–	–	–
… IMAP	2025-09	≤ 1.2	5,117,536	4,778,534	–	0	0	–	–	–
… POP3	2025-09	≤ 1.2	4,808,009	4,543,059	–	0	0	–	–	–

This also means we could not scan for reused keystreams, as we would need at least two tickets to do so. However, since we only found one reused keystream in the previous scans, we do not expect to find any in this scan either.

3.5.1 Online Tests

In the T100k scan, we observed that 82 % of the TLS servers issue tickets in at least one TLS version, as shown in Table 1. For the IP100k scan, 71 % of TLS servers issue tickets in at least one version. Of these, around 95 % (T100k) and 97 % (IP100k) supported resumption with the issued tickets. In both scans, we observed that less than 1 % of servers issued tickets only on some connections. Most servers either always sent tickets or never did. Support for session tickets in the T100k seemed higher in newer versions, while in the IP100k, it was similar for all TLS versions. We provide an overview of the ticket support per version in Figure 10. In TLS 1.2 and 1.3, when filtering out PSKs that seem to be session IDs due to the length, around 75 % of servers issue tickets. Around 95 % of servers that issue tickets also resume these tickets.

Authenticated Tickets and Padding Oracles. In our scans, we could not detect missing authenticity protection or a server vulnerable to padding oracle attacks. Our open-source analysis identified that the implementations use Encrypt-then-MAC, AEAD ciphers, or no padding, which all prevent such attacks. This may also be the case for servers deployed on the internet. However, our tests did not search for more elaborate side channels (e.g., timing-based side channels). We thus cannot rule out all vulnerabilities enabling these attacks.



*TLS 1.3 servers that sent tickets with a length other than 32. Tickets with length 32 might be session IDs.

Figure 10: Observed support for TLS and session tickets in the T100k and IP100k scans. We show the absolute numbers and relative numbers of servers that support TLS, issue tickets, and resume tickets. The relative numbers are shown on the right y-axis and are relative to the previous property (e.g., the number of servers that issue tickets is relative to the number of servers that support the respective TLS version).

3.5.2 Offline Tests

In the scans focusing on offline tests, we determined similar support for session tickets. We did not find any unencrypted tickets. However, we found several servers that still allowed us to recover the plaintext.

3.5.2.1 Weak STEKs: AWS and StackPath

We identified several weak STEKs in the offline tests for our scans. The results are shown in Table 2. Below, we further analyze our findings.

AWS. In the initial scans, we found 1,903 unique hosts, all within the Tranco top 100k, belonging to AWS ALBs that used an all-zero STEK. These tickets were encrypted using AES-256-CBC and the format of RFC 5077 without the length field. We did not find the HMAC key and assume it was set correctly. Due to the high impact, we decided to report our findings before continuing with the full scan. We could see from the already obtained data that the hosts were not vulnerable at all times but only in some time intervals. This suggested an error in the key rotation, which was later confirmed to us by AWS developers. They stated that the issue mostly affected so-called redirector hosts. These are responsible for redirecting clients from an initial domain (e.g., `example.com`) to a

Table 2: Weak keys discovered in our scans.

Scan	Servers Found	Encryption		Authentication	
		Algorithm	Key	Algorithm	Key
pre-T1M	1903	AES-256-CBC	00 00 … 00 00	–	–
	20	AES-128-CBC	00 00 … 00 00	HMAC-SHA256	00 00 … 00 00
T1M	3	AES-128-CBC	00 00 … 00 00	HMAC-SHA256	00 00 … 00 00
T100k	1	AES-128-CBC	00 00 … 00 00	HMAC-SHA256	00 00 … 00 00
IPF	5	AES-256-CBC	00 00 … 00 00	–	–
	94	AES-128-CBC	00 00 … 00 00	HMAC-SHA256	00 00 … 00 00
	12	AES-256-CBC	00 00 … 00 00	HMAC-SHA384	00 00 … 00 00
	3	AES-128-CBC	10 11 … 1e 1f	HMAC-SHA256	20 … 2f 00 … 00[a]
	75	AES-256-CBC	31 … 31 00 … 00[b]	HMAC-SHA256	31 … 31 00 … 00[b]

a: This key consists of 16 consecutively increasing bytes followed by 16 `0x00` bytes.

b: This key consists of 16 `0x31` bytes followed by 16 `0x00` bytes.

final domain (e.g., `www.example.com`). However, we want to stress that connections to the initial domain often also contain HTTP cookies, which should be confidential. Further, a MitM attack is also still possible.

StackPath. During the initial pre-scans, we also found 20 hosts belonging to StackPath that used an all-zero STEK and HMAC key. These tickets were encrypted using AES-128-CBC, authenticated using HMAC-SHA256, and used the format of RFC 5077 without the length field. Again, we immediately reported the issue. The affected servers behaved differently from AWS servers. Only a small number of tickets were vulnerable, but this was not limited to a specific timeframe, unlike with AWS. Interestingly, we found all servers were hosted on the same IP address. Using an online reverse DNS lookup tool, we found a total of 171 domains on the same IP.[20] By scanning these and collecting 1,000 tickets per hostname, we identified 90 hostnames for which we could observe vulnerable tickets. We identified that on vulnerable hosts, on average, 1.4 % of the issued tickets per host were affected. StackPath did not give us any insight into how this came to be, but resolved the issue.

3.5.2.2 Further Weak STEKs

During the final T1M scan, we found three further hosts using an all-zero key for both encryption and HMAC. These hosts used AES-128-CBC with HMAC-SHA256 and supported TLS 1.2. One of these hosts also supported TLS 1.3, also using zero-keys for session tickets. We did not further inves-

20. We used https://www.yougetsignal.com/tools/web-sites-on-web-server/ (accessed: 2026-07-19).

tigate these hosts. For the T100k, we still found the host supporting TLS 1.2 and 1.3 to still be using a zero-key.

During the IPF scan, we found 189 servers using a weak key. Out of these, 111 servers used an all-zero key. For 5 hosts, we did not detect a weak key for the HMAC algorithm. 12 hosts used SHA384, which none of the analyzed open-source libraries use. This implies that they used a different implementation or reconfigured the used library. The remaining 78 servers did not use keys that solely consisted of zero bytes. We explore the structure of these keys below.

Non-Constant Keys. Three servers encrypted their tickets using AES-128-CBC using the following key consisting of increasing bytes `0x101112...1F`. Looking at the first bytes of the ticket, we identified a key name that also uses bytes that count up, starting at zero (i.e., `0x000102...0F`). We further discovered that the authentication tag was computed using HMAC-SHA256 with the key `0x2021...2f`. By connecting to the server and taking a look at the HTTP response, we determined that it is an nginx server. We briefly explore a possible cause for this in Section 3.6.1.

Partial Keys. We found another set of 75 servers that used an HMAC key that only consisted of the byte `0x31` (ASCII '1'). These servers used HMAC-SHA256 but with a 16-byte key, which internally gets padded with zero bytes [S105]. Investigating these servers, we could find that they also use AES-256-CBC with a key consisting of $16 \times$ `0x31` bytes followed by $16 \times$ `0x00` bytes. We assume the key was only partially initialized, and the part that was initialized was set to a weak value.

3.5.2.3 Reused Keystream

In our IPF scan, we tested for reused keystreams and found one affected server. Upon connecting to the server on a later date, the server no longer reused keystreams. The web interface showed that the server ran software called 'GateManager'. Further, the format of the observed tickets resembled mbedTLS. Hence, we assume that the server had low entropy when scanned. When looking at the format (assuming mbedTLS), we could see that the tickets during the scan reused the same nonce. We contacted the software vendor but could not identify the root cause of this issue.

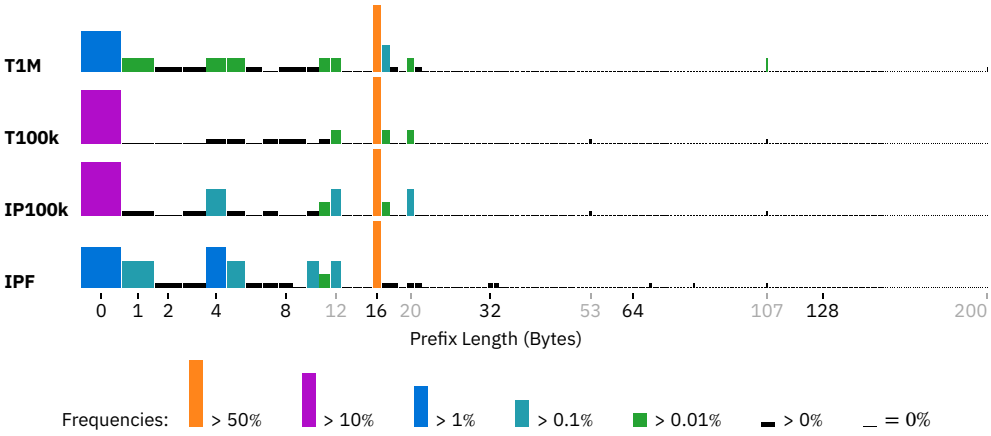


Figure 11: Frequency of detected ticket prefix sizes in our scans. We grouped the frequencies into bins, with 100 % representing the total number of hosts, and used a pseudo-logarithmic scale to visualize them.

3.5.3 Disclosure

We responsibly disclosed all of our findings to the respective developers and our national CERT. AWS developers acknowledged our findings and promptly fixed the issue on their servers [W123]. While we did not get feedback from StackPath, the issue was also resolved on their servers. We further contacted a hosting provider and an infrastructure company. The hosting provider told us they would forward our report to their customers. The infrastructure company did not respond. We sent the remaining list of servers to our local CERT.

3.5.4 Session Ticket Structures

While performing our scans, we also collected statistics about the session ticket structure. As shown in Figure 7, most analyzed open-source libraries use a key name of 16 bytes. This was also the dominant prefix length we observed throughout our scans, followed by a prefix length of zero bytes. Many hosts used a prefix length of four bytes, which is also consistent with our open-source analysis. We provide an overview of the observed prefix lengths in Figure 11.

We further found that servers issue session tickets of vastly different lengths. Most ticket lengths were multiples of 16 in the range of 160 to 240 bytes [O2: Figure A.8]. However, not all servers responded with tickets of reasonable size. Many servers also responded with tickets longer than 1,000 bytes [O2: Figure A.9]. The longest tickets we received were over 9,000

bytes. We could not observe such deviations in the tickets from the Tranco top 100k hosts; all the collected tickets were smaller than 700 bytes.

We also received surprisingly short tickets. While the RFC defines no minimum ticket length, we expected that a ticket would contain enough bytes to encrypt the 48 bytes of the main secret in order to achieve a stateless session resumption or at least 32 bytes to mimic a session ID. The shortest "tickets" we received were 14 bytes and consisted of the ASCII string `TICKET FAILURE`. We observed this behavior for 18 servers during our IPF scan.

IPmail. During our scans of email servers, we found that 92 %–97 % of TLS-enabled POP3 and IMAP servers issue tickets, which is higher than the support we observed for HTTPS servers in our previous scans. For SMTP, this support was significantly lower, with only 14.3 %, 36.9 %, and 12.8 % on ports 25, 456, and 587, respectively. We attribute this low support to the fact that SMTP is used for sending mail, which happens less frequently than fetching the inbox via POP3 or IMAP. Hence, the support for session resumption is less relevant for SMTP. During our scans, we found no email server that issued tickets with a weak STEK.

3.6 Discussion

We discovered two critical issues in our evaluation: using weak keys and reusing a nonce. In the following, we take a closer look at possible causes for the use of weak keys. Afterward, we discuss possible countermeasures for both issues.

3.6.1 Potential Causes for Weak Keys

Most web servers we analyzed in Section 3.3 allow the STEK to be provided as a file. Apache Tomcat does not allow this but assumes the underlying library manages the STEK. For Apache httpd, nginx, and OpenLiteSpeed, this file contains the key name, the AES key, and the HMAC key. Since these servers simply read the file contents, a static config file containing only zeroes or predictable structures, such as `00 01 02 … 2f`, could result in weak keys. Caddy's file format differs from the others: The file contains a serialized internal structure that contains 32 bytes of random data and metadata about key expiration. Caddy passes the random data to GoTLS, which then uses SHA512 to derive the AES and HMAC keys.

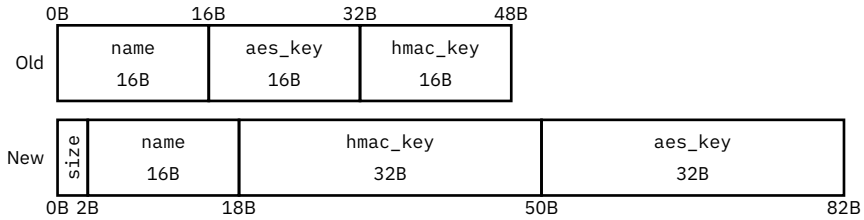


Figure 12: Internal structure of nginx to store session key material. The format changed in December 2016. We show the format before (old) and after (new) this change.

While this derivation means that an encryption or HMAC key consisting of only zeroes is unlikely, a misconfiguration of the file could still result in predictable keys.

The internal structures of the analyzed web servers are similar to the file formats. In the following, we take a closer look at nginx because it changed its internal structure in December 2016. The old and the new format are shown in Figure 12. The new format is longer and introduces an additional size field. If the size is set to 48, nginx will use AES-128-CBC. Otherwise, it will use AES-256-CBC. We argue that this change could be the cause of some servers with weak keys we discovered with uninitialized or partially initialized keys. If an external program modifying this struct is unaware of the change, it might still assume the old format, while nginx uses the new format. This can cause two behaviors, both of which we observed in our scan. Either the `hmac_key` is initialized, but the `aes_key` is left empty. Or, only 16 bytes of each `hmac_key` and `aes_key` are set.

The first case occurs if the external program assumes all fields are aligned in memory and writes 48 bytes into the struct. In this case, the `name` and `hmac_key` are set, and `aes_key` is left uninitialized. Since `size` is not explicitly set to 48, nginx will expect 32-byte keys. However, even if `size` is set to 48, the `aes_key` is still uninitialized. We have seen such behavior in our AWS finding and five times in the IPF scan (see Table 2).

The second case occurs if the external program assumes that the keys are still expected to be 16 bytes and writes each key independently. In this case, only 16 bytes are written to the `hmac_key` and `aes_key`. We have seen this behavior for the 75 hosts that used a key consisting of 16 `0x31` bytes followed by 16 `0x00` bytes (see Table 2).

3.6.2 Countermeasures

While the root causes for the uncovered issues can be hard to fix generically, we propose that TLS libraries deploy defense-in-depth countermeasures to protect themselves from catastrophic implementation defects.

Weak Keys. To prevent the usage of weak keys, we propose that libraries validate the keys when they are set. That is, whenever the STEK is rotated, the libraries validate the key material. This validation could be as trivial as checking that not every byte is set to zero. In most cases, this check is rather cheap and would have already prevented most of our discovered issues. However, since most STEKs live for at least an hour [71] and the STEK only has to be validated on a change, the check could also be more complex. Libraries could, for example, check the entropy using the Hamming weight or the average Hamming distance between each byte. The Hamming weight counts the number of set bits in the key material, while Hamming distance measures the bit differences across key bytes. Both could serve as lightweight metrics that remain low for predictable or weakly initialized data. While these tests are not computationally free, they are still lightweight. We argue that even more sophisticated tests might be justified, given the impact of a weak STEK.

Reused Nonces & Cryptographic Wear-out. The impact of reused nonces and cryptographic wear-out can be mitigated by sticking to AES-CBC with HMAC, as proposed in RFC 5077 [S96]. If these algorithms are not available, the library should respect the safety limits of the used cipher and mode of operation. This could, for example, be achieved through an internal counter that automatically switches the STEK once the threshold is reached. Since AES-GCM fails catastrophically in the case of nonce reuse, misuse-resistant algorithms should be preferred. Examples of this include AES-SIV [44], AES-GCM-SIV [S99], AEZ [47], and MRO from the MEM-AEAD [42] cipher family. If these algorithms are not implemented, validating the reuse of the last nonce as a minimal check would already mitigate many vulnerabilities related to repeated randomness. This is, for example, already implemented in the Bouncy Castle cryptography library for AES-GCM in general.[21]

21. Added in release 1.56. https://www.bouncycastle.org/download/bouncy-castle-java/?filter=java%3Drelease-1-56#release-notes (accessed: 2026-07-19)

3.6.3 Secure Email servers

We were surprised to find that no email server we scanned used a weak STEK. In the HTTPS ecosystem, we observed most weak keys in centralized hosting providers. The business model of these providers is to accelerate content delivery to clients. Session resumption is a key component in achieving this, as it accelerates the TLS handshake and, at the same time, reduces the load on the server. In contrast, email servers are often operated by smaller providers or even individuals. The handshake times are less critical here, as emails are not as time-sensitive as web content. Hence, we believe that email server operators are less likely to modify the default behavior of the TLS library, which we have found secure in our analysis of open-source libraries (Section 3.3).

3.6.4 Evaluating the Quality of Key Material

In this chapter, we discovered several servers using low-entropy keys, undermining the security of the TLS connection. Prior work has repeatedly shown that weak or insufficiently random key material is a practical problem in deployed TLS systems. For RSA, Heninger et al. [45] found shared primes across certificates in a large-scale scan, which enabled recovery of the affected private keys. Nemec et al. [66] studied whether collected RSA public keys are factorable with Coppersmith's attack and identified a small number of affected TLS keys. For signature schemes, Breitner and Heninger [21] showed that biased or reused ECDSA nonces in real-world deployments can expose the signing key. Weak public-key parameters have likewise been observed in key exchange: Valenta et al. [78] and Dorey et al. [32] reported TLS servers with weak Diffie-Hellman groups and small subgroups. Valenta et al. [79] found servers reusing elliptic-curve public keys that should have been ephemeral. Complementing these studies, Böck [W127] provides a service for checking public keys for known weaknesses.

At a high level, weak STEKs resemble weak handshake keys in that both originate from poor key generation or handling. The crucial difference is observability: prior work could inspect public material transmitted during the TLS handshake, whereas the STEK itself is never revealed to the client. Although the ticket is sent to the client, its format and underlying algorithms are not standardized. As a result, we could identify low-

entropy STEKs only by bruteforcing a range of ticket formats, algorithms, and candidate keys. This limited auditability makes the STEK a particularly high-risk component of TLS.

3.7 Conclusions

In this chapter, we proposed possible pitfalls in the construction of session tickets, answering RQ1. We analyzed the source code of 14 open-source TLS libraries and 5 open-source HTTPS servers, answering RQ2. Our large-scale analysis of session tickets confirmed critiques raised by many cryptographic experts [71, 74, W147], answering RQ3. We showed that improper usage of session tickets can have devastating consequences for TLS connections and break forward secrecy guarantees delivered by TLS. The concrete issues discovered seem unlikely but are an important reminder that looking for seemingly trivial or contrived issues can be worthwhile for auditors.

While our study could already detect a significant number of vulnerable servers, it was limited by the number of tests we could perform in a large-scale black-box evaluation. We only performed a white-box analysis of some servers in the lab. This did not cover all deployed TLS stacks, especially closed-source ones. A white-box analysis of the used implementations might find more issues, making session tickets a generally risky feature.

Since even servers administrated by well-established cloud providers were affected by these issues, the community must acknowledge that catastrophic software defects happen in the real world and that we must build resilient security systems and avoid dangerous shortcuts.

At its core, we attribute the observed issues to the unauditability of session tickets. Since the STEK and the layout are unknown and never revealed to the client, clients cannot simply validate the strength of the key, the presence of a MAC, or even the algorithms used. This lack of transparency creates a space for bad implementations, silently failing crypto, and hidden backdoors. While key generation always seems to have this potential, it is especially severe for one-sided symmetric keys, as with asymmetric keys, at least the public key can be audited externally [45, 86]. Having hard-to-analyze keys in a protocol at a place where a weak or

leaked key causes the protocol to fail catastrophically is a huge risk that requires careful consideration. Since these keys cannot be audited externally, we argue that libraries should start auditing them themselves before they use them. In public-key cryptosystems, it is already common practice to ensure that the generated key material is of a specific form, for example, to ensure that the key material will result in a strong key or as a safety net for failing random number generators [S89, S112]. Adding additional checks to randomly drawn symmetric keys could, at least to some extent, ensure that accidentally weak key material does not break the protocol.

On the positive side, we could not observe any padding oracle in our scans. This vulnerability resurfaced in many recent scientific papers analyzing TLS record processing [38, 70] and affected nearly 2 % of the Alexa Top 1 Million servers in 2019 [62]. Compared to TLS record encryption, which in TLS 1.2 uses the MAC-then-Encrypt construction by default [S98], RFC 5077 recommends using Encrypt-then-MAC for AES-CBC with HMAC [S96]. Encrypt-then-MAC cryptographically prevents CBC padding oracle attacks, and we believe this is the main reason for not finding this vulnerability in session tickets. This example shows how a standard can positively affect a large-scale deployment of a cryptographic protocol.

Another positive development connected to session tickets is their usage in TLS 1.3, which reduces the impact of session ticket implementation dangers. First, TLS 1.3 servers send session tickets over an encrypted TLS channel, allowing attackers to obtain them only during the session resumption. Second, TLS 1.3 session tickets can be used such that revealing their contents does not break forward secrecy and only affects the resumed session. These two major differences positively affect the security of TLS 1.3 and should be considered when designing new cryptographic protocols.

Our work has also shown that the freedom in the specification led to the usage of different session ticket structures and cryptographic algorithms. These changes can have an impact on the security of session tickets. Given the previous works on security bounds affecting the indistinguishability and integrity of encryption schemes in different contexts [15, 29, 48, 59], we would like to encourage implementors to be mindful when selecting different algorithms and to have these limits in mind.

Chapter 4
Session Resumption and Authentication

In the previous chapter, we analyzed how single hosts handle session tickets. There, we considered the security of session tickets in isolation and analyzed how they are constructed and handled by single servers. In reality, server infrastructures are often more complex. A single server may host multiple websites, a feature known as virtual hosting. The websites may be unrelated or have different security requirements, and the server needs to handle these different requirements.

In addition, the server may be part of a larger infrastructure, for example, a Content Delivery Network (CDN). CDNs typically host servers around the world to minimize latency for users around the globe. To further minimize latency and computational load, the servers often use session resumption. CDNs have an incentive to enable session resumption across different servers, as this allows clients to resume sessions even if they later connect to a different server (e.g., due to load balancing). This can be achieved by sharing the STEK across all servers, which allows each server to handle session tickets from any other server. However, this can lead to security issues, as Delignat-Lavaud and Bhargavan [30] have shown. They uncovered vulnerabilities that bypassed server authentication, which they called *virtual host confusion*. Their attack exploits the fact that a server accepting a ticket may serve different content than the server that issued the ticket. This enables an attacker to trick a user into connecting to a server that serves malicious content, while the user believes they are connecting to a benign website, completely undermining TLS server authentication.

In their attack, Delignat-Lavaud and Bhargavan [30] used session ID-based resumption. Since their publication in 2015, the TLS ecosystem has changed significantly: HTTPS is more widespread [W132], Cloudflare started offering free TLS in 2014 [W130], Let's Encrypt started in 2015 [8], and TLS 1.3 was first standardized in 2018 [S118]. Furthermore, session tickets are nowadays preferred over session IDs, as they reduce the load on the server and allow for more flexible session resumption. In this chapter,

we analyze how session resumption affects the security of TLS in virtual hosting scenarios, leading to the following research question:

> **RQ4:** What are the broader implications of attacks exploiting session resumption with session tickets in virtual hosting environments, and how do they affect TLS authentication guarantees?

In this chapter, we present two session ticket confusion attacks. Both attacks rely on the fact that neither party is re-authenticated during resumption. If the server does not properly isolate the session tickets of different virtual hosts, an attacker can exploit this to impersonate a server or a client.

Server Authentication Attacks. Our first attack extends the work of Delignat-Lavaud and Bhargavan [30]. We consider that an attacker hosts their domain in a virtual hosting environment along with the victim domain, for example, in a CDN. We assume that the domains are hosted on different IPs and that the attacker can reroute traffic from the victim client on the IP layer. During the resumption connection of the victim, the attacker reroutes the TLS traffic to the server on which the attacker's domain is hosted. If the server does not properly isolate the session tickets of the different virtual hosts, it will accept the session ticket and serve attacker-controlled content to the victim. Because the server is not re-authenticated, the client believes it is connected to their intended domain. This enables the attacker to obtain sensitive information from the victim, like passwords or cookies.

Client Authentication Attacks. Our second attack aims to circumvent client authentication. We consider a scenario where a server hosts multiple virtual hosts. We assume that the attacker is allowed to connect to one of the virtual hosts but not the other, and that this is enforced via client authentication. In our attack, the attacker first connects to the first virtual host and obtains a session ticket. Then, the attacker connects to the second virtual host and uses the session ticket obtained from the first virtual host to resume the session. If the server does not properly isolate the session tickets of the different virtual hosts, it will accept the session ticket and allow the attacker to connect to the second virtual host, bypassing client authentication.

These two attacks show that session resumption can have significant implications for the security of TLS. However, it is unclear whether these attacks are relevant in practice, as servers may properly isolate the session tickets of different virtual hosts. This leads to the following research question:

RQ5: Do these session ticket confusion vulnerabilities translate to real-world server implementations and virtual hosting environments?

Evaluation of Open-Source Implementations. To answer this question, we analyzed whether open-source HTTPS servers are vulnerable to the attacks. In the original publication [O3], we analyzed four open-source HTTPS servers: Apache httpd, Caddy, nginx, and (Open)LiteSpeed. We found none of them vulnerable to server authentication attacks, but all of them vulnerable to client authentication attacks. For LiteSpeed, the attack does not require session resumption and can be performed in a full TLS handshake.

In this thesis, we extended the client authentication analysis by analyzing Apache Tomcat and Microsoft IIS, finding issues in both of them. Similar to LiteSpeed, Apache Tomcat is vulnerable to the client authentication attack without requiring session resumption. For Microsoft IIS, we can also circumvent client authentication under specific circumstances, which we discuss in Section 4.3.2.2. The results for Apache Tomcat and Microsoft IIS were first described by Willi Hensch [46] (see Section 1.5.1).

Large-Scale Server Evaluation. To evaluate whether the server-authentication attack is relevant in practice, we performed a large-scale scan of the TLS ecosystem. Springall et al. [71] estimated the prevalence of STEK sharing based on prefixes to show the impact on forward secrecy. Delignat-Lavaud and Bhargavan [30] proved the existence of this vulnerability through manual testing because, as they pointed out, evaluating these vulnerabilities on a large scale is challenging. We overcome these challenges by creating a list of candidate domains hosted on the same servers, a large-scale collection of session tickets, and evaluating responses to our resumed requests. Using the developed methodology, we detected

vulnerabilities affecting six clusters of providers. Among them were Fastly and DDoS-Guard.

Contributions.

- *Attacks*: We systematically analyzed the impact of session tickets on the TLS connection and presented novel attacks that bypass client and server authentication in virtual hosting scenarios (Section 4.2).
- *Open-Source Analysis*: We analyzed leading open-source TLS implementations to assess their handling of virtual host isolation in session resumption. Our analysis uncovered critical vulnerabilities in widely used software, including Apache httpd, Apache Tomcat, Caddy, nginx, and (Open)LiteSpeed (Section 4.3).
- *Large-Scale Analysis*: We proposed an efficient methodology for large-scale scanning of server authentication bypass vulnerabilities. Using this approach, we identified multiple vulnerable CDNs, including Fastly. Additionally, we uncovered a virtual host confusion vulnerability unrelated to tickets in Cloudflare SaaS (Section 4.4).

Disclaimer on Originality

This chapter is mostly based on [O3]. For more details, see Section 1.5.1.

4.1 Background on Virtual Hosting and CDNs

Web servers often employ virtual hosting to host multiple websites on the same machine. For any incoming connection, the server delegates the request to the responsible virtual host, which serves the requested resource. HTTP uses the `Host` header, which specifies the domain and port of the requested service, allowing the virtual host responsible for the domain to be chosen [S110].

A specific form of virtual hosting is reverse proxying. Here, a server acts as a middlebox between clients and one or more origin servers. Instead of directly managing different virtual hosts, the reverse proxy forwards requests to other servers and relays the response. Typical use cases include caching static resources, load balancing, filtering specific traffic, or rewriting specific requests before delegating any requests to the responsible origin server(s). To this end, the proxy needs to access the plaintext requests. That is, it needs to act as the TLS server and provide a

valid certificate for any virtual host. As the certificate is required before receiving the HTTP request, the server cannot use the `Host` header.

One approach to providing virtual hosting would be to use a certificate that covers all domains hosted on a server via the Subject Alternative Name (SAN) field. However, using a single certificate for multiple (possibly unrelated) domains may cause security issues [30]. As an alternative, the client can include the Server Name Indication (SNI) extension in their TLS ClientHello to indicate the desired domain name [S91]. The server can then select a valid certificate during the handshake, allowing one certificate to be issued per domain. The SNI extension is mandatory to implement since TLS 1.3 [S119].

CDNs. Content Delivery Networks (CDNs) are commonly used to speed up website access by deploying reverse proxies physically positioned near the user. These proxies are called *edge servers*. CDNs deploy many edge servers on distinct IPs and locations to improve redundancy and response times in different locations. By relying on this dedicated CDN infrastructure, companies can improve a website's performance in different locations and benefit from additional security measures, such as protection against Distributed Denial of Service (DDoS) attacks. CDNs typically employ virtual hosting to serve many customers using limited hardware. For enterprise-level customers, more specialized setups are possible. For example, to support legacy clients, Cloudflare allows customers to request a dedicated IP address for their domain that does not rely on SNI [C167].

Autonomous System. An Autonomous System (AS) is a collection of IP addresses and their associated routing policies, typically operated as a single administrative entity. CDNs typically operate many IP addresses distributed across multiple ASes to achieve geographic coverage and redundancy. The information to which AS an IP address belongs is publicly available. This information is useful for our analyses, as it helps us identify which IP addresses belong to the same operator.

4.2 Vulnerabilities from STEK Sharing

CDNs benefit from sharing their STEK across their infrastructure; a client routed to a different node due to load balancing can still resume their session. Similarly, single servers may use the same STEK for all

virtual hosts to simplify key management. In this work, we describe two *session ticket confusion* attacks caused by STEK sharing that circumvent either server or client authentication. Both attacks use tickets issued for one domain and resume them while confusing the server into serving a different domain. We consider two attack scenarios with varying attacker capabilities and impact. The first attack is similar to the scenario described by Delignat-Lavaud and Bhargavan [30], in which server authentication is bypassed, enabling TLS-MitM attacks. In our second attack, we circumvent client authentication, allowing an attacker to access restricted sites.

4.2.1 Server Authentication Bypass

The first attack we analyze in this work exploits an issue in a CDN to attack a client visiting a website hosted by that CDN. The attacker's goal is to retrieve the plaintext request and respond to it, effectively acting as a TLS-MitM.

Victim Behavior. A victim connects to a website using TLS and is issued a session ticket. Within this connection, the victim validates the TLS certificate and ensures they are retrieving the correct website. The victim, within the session ticket's lifetime, performs another request to the website and resumes the session ticket for this request. As this is a resumption, the server does not present its certificate again.

Attacker Goal and Impact. The attacker's goal is to act as a MitM on the TLS layer for the second request. This exposes any information within this request to the attacker, including (session) cookies and any data the user entered on the website. The attacker can also respond to this request, allowing them to serve malicious content without breaking TLS authentication; for example, respond with a website containing malicious scripts, effectively achieving Cross-Site Scripting (XSS).

Attacker Capabilities. To achieve their goals, we consider an active attacker who can reroute the victim's network traffic on the IP layer. This can, for example, be achieved by changing the addresses in the IP header, as this is not protected by TLS. We further assume that the attacker hosts a backend server and registers this at the CDN. The attacker may use any service the CDN offers, including enterprise-level features (see Section 4.1), such as a dedicated IP address for their domain.

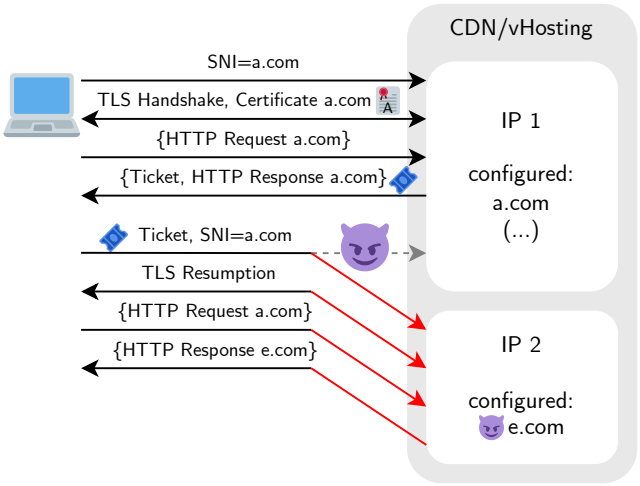


Figure 13: Server authentication bypass. The CDN/virtual hosting is configured to host `a.com` on IP 1 and the attacker-controlled `e.com` on IP 2. Upon resumption, the attacker reroutes the request to IP 2. As the CDN shares the STEK between all virtual hosts, the ticket can be decrypted on IP 2, causing the CDN to serve `e.com`.

Attack Description. We provide an overview of the attack in Figure 13. We assume the victim wants to visit `a.com`, which is hosted on IP 1. The attacker owns `e.com`, which the CDN hosts on IP 2. As outlined above, the victim performs two requests to `a.com`. The attacker reroutes the second request to IP 2. As both servers are operated by the CDN and share a STEK, the server at IP 2 can decrypt the ticket and resume the session. However, since the server at IP 2 is only configured to host `e.com`, it may ignore the SNI or HTTP `Host` header that specifies `a.com` and instead forward the request to the attacker-controlled backend of `e.com`.

4.2.2 Client Authentication Bypass

Our second attack also exploits an isolation issue on the server side. In this case, the victim is a website hosted on the server. The goal of the attacker is to circumvent TLS client authentication.

Victim Behavior. The victim's host has two websites: one that the attacker can access and one that is restricted and inaccessible to the attacker. The attacker-accessible website may or may not use TLS client authentication. The inaccessible website is protected using TLS client authentication.

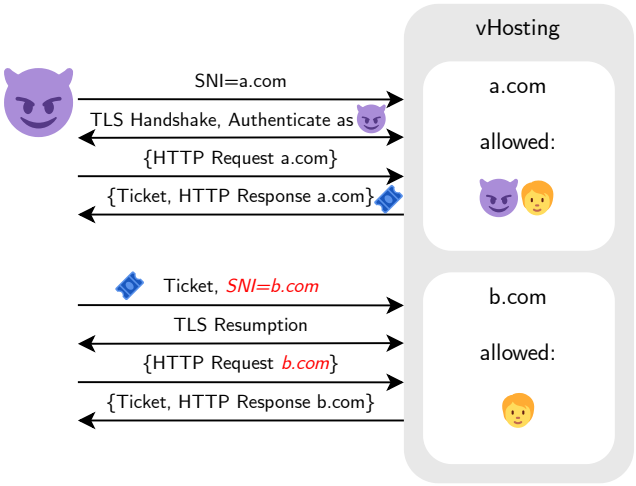


Figure 14: Client authentication bypass. The attacker can access `a.com` using its client certificate, but not `b.com`. The attacker performs a malicious handshake, modifying the SNI extension and HTTP request. This allows them to circumvent the authentication and access `b.com` in a resumption.

Attacker Goal and Impact. The attacker wants to visit the inaccessible website, which they should not be able to. This can be classified as a privilege escalation or an authentication bypass.

Attacker Capabilities. The attacker can perform TLS connections to the server. If the accessible website is also protected by TLS client authentication, we assume the attacker possesses a certificate to access this website.

Attack Description. We provide an overview of the attack in Figure 14. The attacker first visits the accessible website and is issued a session ticket by the server. They then attempt to resume the session ticket and change the SNI and `Host` header (in Figure 14, we show both as `b.com`). If the server accepts and resumes the session ticket, the client certificate authentication is skipped, and the server may serve the inaccessible website to the unauthenticated attacker.

4.3 Open-Source Analysis

We analyzed open-source server implementations to determine whether they are affected by our proposed session ticket confusion attacks. We started with the server authentication bypass and analyzed the behavior when clients change the SNI and `Host` header in resumed sessions. While this analysis exceeds the attacker's capabilities in the server authentication bypass model (Section 4.2.1; the attacker cannot modify the `Host`

header or SNI), it still provides an overview of potential entry points for client authentication bypasses.

We configured each server to host at least two different domains as virtual hosts: the victim's host and the attacker's host. We covered cases where either of these two is the default host, as well as the case where a third host is the default host. Each virtual host used its certificate, which only covers their domain. Generally, we configured all virtual hosts to use the same STEK. In our evaluation of the server authentication bypass, we also evaluated configuring the virtual hosts with different STEKs. During our analysis, we ensured that we covered all resumption combinations: using a ticket from a default host, resuming a ticket at a default host, and resuming a ticket from a non-default host at another non-default host. We evaluated each server in TLS 1.2 and TLS 1.3. We will outline further parameters in the respective sections.

Once we have deployed a server configuration, we retrieve a ticket for a configured virtual host. In resumption attempts, we changed the SNI value and `Host` header to be either the ticket-issuing host *I* or the resumption host *R* configured on the server. We also considered omitting the SNI extension, which we denote as SNI=None when presenting the results. The `Host` header was never omitted, as servers expect it and respond with a `400 invalid request` if it is missing.

We chose open-source web server implementations with a market share over 10 %: Apache httpd, nginx, and OpenLiteSpeed [W148]. In addition, we chose Caddy, as it claims easy configuration [C201] and was referenced positively in previous works [8, 55, 71]. All implementations support TLS session tickets and virtual hosting. Only OpenLiteSpeed does not support client authentication, but its enterprise version, LiteSpeed, does.[22] We used LiteSpeed when analyzing client authentication with the trial version included in the official Docker image. We used the official Docker images for all implementations in their latest stable version at the time of our evaluation, which we list in Table A2. In this thesis, we extended the client authentication analysis by including Apache Tomcat and Microsoft IIS (using "IIS Client Certificate Mapping Authentication" [C181]). We

22. We found contradictory information regarding the support for client authentication in OpenLiteSpeed. We ultimately concluded it is not supported [C156].

discuss the results of Microsoft IIS separately in Section 4.3.2.2, as it works differently than the other implementations.

We additionally considered *strict* versions of the following two implementations: For Apache httpd, we enabled `SSLStrictSNIVHostCheck`, which we denote as *Apache httpd (strict)*. For nginx, we considered two cases where we defined a default virtual host that handles any hostname otherwise undefined. This host either returns an HTTP `404` error or has the option `ssl_reject_handshake` enabled and uses a unique certificate. We denote these configurations as *nginx (strict HTTP)* and *nginx (strict SNI)*, respectively. We tested OpenLiteSpeed with and without the `sslStrictSni` option. This option did not impact our results, and therefore, we omit any distinction. OpenLiteSpeed provides an admin web interface that also uses TLS. We considered this an additional virtual host for which we attempted to perform a server authentication bypass attack.[23]

We found two approaches to configure Microsoft IIS more strictly. One approach is to disable "Anonymous Authentication", which we denote as *Microsoft IIS (no anon)*. The second approach is to change the access permissions on the file system, which we denote as *Microsoft IIS (ACL)*. We explain this in more depth in Section 4.3.2.2.

4.3.1 Server Authentication

In addition to the general configuration options outlined previously, we evaluated each implementation in the following scenarios: Multiple domains may be hosted on a single server instance on the same port, on different ports, or on two different server instances. We also considered scenarios where two hosts are configured with unique STEKs.

Table 3 presents the results of our evaluation and shows the content served (if any) within such a manipulated resumption attempt. Note that although the server authentication bypass scenario only allows SNI=I and `Host`=I (highlighted column), other cases may become relevant when combined with additional attacks or more complex setups.

23. As the Docker image uses a default certificate, we do not consider resumption attempts from one admin interface to another, but just resumptions between the admin interface and a virtual host.

Table 3: Server behavior when resuming a ticket. All websites were hosted on a single server. I is the ticket-issuing host, and R is another virtual host. We show whether the server resumes the ticket and, if it is resumed, which virtual host is served. We highlight the column covering the server authentication bypass attacker model (SNI=I, Host=I).

		SNI=I		SNI=R		SNI=None	
		Host=I	Host=R	Host=I	Host=R	Host=I	Host=R
Apache httpd	TLS 1.2 TLS 1.3	I	421	*not resumed*[F]		I[†]	421[†] R[†]
Apache httpd (strict)	TLS 1.2 TLS 1.3	I	421	*not resumed*[F]		I[†] 403[†]	421[†] 403[†]
Caddy	TLS 1.2 TLS 1.3	I	421	421	R	*not resumed*[BF]	
nginx	TLS 1.2 TLS 1.3	I	R	I	R	I	R
nginx (strict HTTP)	TLS 1.2 TLS 1.3	I	R	I	R	I	R
nginx (strict SNI)	TLS 1.2 TLS 1.3	I	R	I	R	I *not resumed*[A]	R
(Open)LiteSpeed	TLS 1.2 TLS 1.3	I	R	*not resumed*[F]		I[†]	R[†]

I: Ticket issuer. R: (Other) Resumption host. 403/421: HTTP error code.
A: Raises TLS alert `unrecognized_name` (112).
B: Raises TLS alert `internal_error` (80) if no default host is defined.
F: The server falls back to a full handshake, according to the standard.
†: Only resumes the tickets issued by the default host in the config.

Results. We found no software vulnerable to circumventing server authentication simply by rerouting traffic to a different server. Hence, all implementations are secure against our server authentication bypass attacker model. However, our extensive tests still uncovered cases where the server resumed session tickets and served a different host. All resumptions occurred when all domains were hosted on a single instance on the same port. We found that a single instance using multiple ports behaves the same as two instances on different IPs. When two instances did not share a host with the same certificate, we could not observe any resumptions, even if the STEK is shared. The results do not change when configuring a unique STEK per virtual host.[24]

Apache httpd properly detects misdirected requests. If the SNI does not match the ticket issuer, Apache httpd rejects the ticket and falls back to a full handshake. Apache httpd further checks that the `Host` header is set to the same value as the SNI and otherwise returns the HTTP error `421 Misdirected Request`. If the SNI is omitted, Apache httpd resumes only the tickets of the first configured host (i.e., the default host). However, for these tickets, Apache httpd behaves inconsistently across TLS versions.

24. (Open)LiteSpeed and Caddy do not support a STEK per virtual host but only one global STEK per server.

In TLS 1.2, it behaves as it does in the SNI=I case: It detects when the host header is set to another host and returns an HTTP error. In TLS 1.3, the result depends on the `SSLStrictSNIVHostCheck` setting. If enabled, requests without an SNI are rejected. If disabled (which it is by default), Apache httpd may serve a different host depending on the `Host` header.

Caddy always requires the SNI to be present. It resumes all tickets. If the SNI and `Host` header match, it returns the content based on these. Otherwise, it returns an HTTP `421` error.

When analyzing nginx, we noticed that nginx uses the same STEK for all virtual hosts. Further, nginx ignores the SNI and only relies on the `Host` header to determine which content to serve. The only case where nginx did not resume a session was in the nginx (strict SNI) configuration, where a missing SNI led to a TLS alert.

We found that OpenLiteSpeed only resumes tickets if the SNI during resumption matches the SNI from the ticket issuance. However, OpenLiteSpeed determines the served content based on the `Host` header. If the client specifies another virtual host in the `Host` header, OpenLiteSpeed returns the content for that virtual host. We could not resume tickets at or from the admin interface in this scenario. We observed LiteSpeed to behave identically to its open-source counterpart.

4.3.2 Client Authentication

As we did not observe ticket resumptions across different virtual hosts when they were hosted on multiple instances in Section 4.3.1, we chose to limit the setups to a single server instance hosting all virtual hosts on a single port on the same instance. We assumed one virtual host to be accessible to the attacker, the initial host *I*, which they use to obtain session tickets. We evaluated scenarios where this host does not require client authentication (“I is freely accessible”) and where it does (“I requires client authentication”). The attacker has a valid client certificate for this host, if required. The other virtual host, *R*, is protected with client authentication, and the attacker has no access to a certificate to authenticate against this host. If both hosts require client authentication, they use distinct CAs for client authentication.

Table 4: Server behavior when attempting to resume a ticket from I with different combinations of SNI and `Host` header. All websites were hosted on a single server on a distinct domain. R always requires client authentication. The configuration should allow the attacker to access only I but not R.

		I is freely accessible						**I requires client authentication**					
	SNI=	**I**		**R**		**None**		**I**		**R**		**None**	
	Host=	**I**	**R**	**I**	**R**	**I**	**R**	**I**	**R**	**I**	**R**	**I**	**R**
Apache httpd	TLS 1.2 TLS 1.3	I	421	*not resumed*[F]		I[†]	421[†] R[†]	I	421	*not resumed*[F]		I[†]	421[†] R[†]
Apache httpd (strict)	TLS 1.2 TLS 1.3	I	421	*not resumed*[F]		I[†] 403[†]	421[†] 403[†]	I	421	*not resumed*[F]		I[†] 403[†]	421[†] 403[†]
Apache Tomcat[*+]	TLS 1.2 TLS 1.3	I	R	*not resumed*[F]		I[†]	R[†]	I	R	*not resumed*[F]		I[†]	R[†]
Caddy	TLS 1.2 TLS 1.3	I	421	*not resumed*[F]		421[B] 421	421[B] 421	I	421	421	R	*not resumed*[BF]	
Microsoft IIS[+]	TLS 1.3	I	403/R[M]	*not resumed*[F]		I[†]	R[†]	I	R	*not resumed*[F]		I[†]	R[†]
Microsoft IIS (ACL)[+]	TLS 1.3	I	403	*not resumed*[F]		I[†]	403[†]	I	401	*not resumed*[F]		I[†]	401[†]
Microsoft IIS (no anon)[+]	TLS 1.3	401	403	*not resumed*[F]		401[†]	403[†]	I	401	*not resumed*[F]		I[†]	401[†]
nginx	TLS 1.2 TLS 1.3	I	421	I	421 400	I	421 400	I	421	I 421	421 R	I	421 R
nginx (strict HTTP)	TLS 1.2 TLS 1.3	I	421	I	421 400	I	421 400	I	421	I 421	421 R	I	421 R
nginx (strict SNI)	TLS 1.2 TLS 1.3	I	421	I	421 400	I *not resumed*[A]	421	I	421	I 421	421 R	I *not resumed*[A]	421
LiteSpeed[*]	TLS 1.2 TLS 1.3	I	R	*not resumed*[F]		I	R	I	R	*not resumed*[F]		I	R

I: Ticket issuer. R: (Other) Resumption host. 400/403/421: HTTP error code.
+: Added server in this thesis (compared to [O3]).
*: Behavior is not session resumption specific. Changing the `Host` header in a normal connection also leaks R.
A: Raises TLS alert `unrecognized_name` (112).
B: Raises TLS alert `internal_error` (80) if no default host is defined.
F: The server falls back to a full handshake, according to the standard.
†: Only resumes the tickets issued by the default host in the config.
M: Microsoft IIS allows accessing the other host if any system-trusted certificate is presented.

4.3.2.1 Results

We found multiple cases where we could circumvent client authentication. Most required the ticket-issuing host to use client authentication, thus classifying the vulnerability as a privilege escalation. Only in LiteSpeed could we bypass authentication altogether without needing any prior authentication. We summarize the results in Table 4.

As expected, all web servers accept the resumption when the SNI matches the initial request. When the `Host` header also matched the initial domain, all servers replied with the content of the initial website. When the `Host` header was set to the domain of the secured resumption host, all servers except LiteSpeed replied with the HTTP status code `421`. This indicates that web servers implemented checks that link the authentication in the TLS layer to the request on the HTTP layer.

Unprivileged Access. Apache Tomcat and LiteSpeed are the only servers in our test that allowed us to access the restricted content without any

prior authentication. After investigation, we found that both are vulnerable without using tickets. Changing the `Host` header during the initial connection also yields the protected host.

Privilege Escalation. For all other servers, we found vulnerabilities that require the attacker to use client authentication at the accessible host. Note that the attacker does not possess a certificate valid for the inaccessible host. The combination of SNI and `Host` header necessary for the attack to work varies between the servers.

Apache httpd incorrectly allowed access to the second site when the client omitted the SNI in the resumption handshake and then requested the target site's domain in the `Host` header. This only worked if the accessible host was the default host (the first in the config) and only in TLS 1.3. In TLS 1.2, the server returned an HTTP `421` status code. The strict config variant prevented this, and the server responded with an HTTP `403` instead.

Caddy is vulnerable if both the SNI and `Host` header within the resumption are set to the inaccessible domain. Note that Caddy has a default setting to prevent changing the `Host` header to bypass client authentication [C164], but it seemingly does not consider session resumption.

We found nginx to be vulnerable if the SNI in the resumption is set to the inaccessible domain or omitted entirely, and the `Host` header is set to the inaccessible domain. The attack only worked for TLS 1.3 resumptions. Using nginx (strict SNI) only prevented the resumption if the SNI was missing. Since the vulnerability also occurred when the SNI was set to the domain of the target site, this did not completely prevent the attack.

Disclosure. We disclosed our findings to the respective developers. We notified Apache httpd, Caddy, and nginx on November 25th, 2024. Apache httpd fixed the issue and assigned CVE-2025-23048 [W139]. Caddy disabled session tickets when using client authentication.[25] This bug was later reported to the GoTLS team, who fixed it in Golang 1.24.13/1.25.7 [C153]. nginx fixed the issue and assigned CVE-2025-23419 [W140]. We reported the issue to LiteSpeed on January 23rd, 2025, and they fixed it on February 4th, 2025. We reported the issue to Apache Tomcat on October 15th, 2025. They fixed the issue and assigned CVE-2025-66614 [W141].

25. Introduced in commit `ed1c594c`.

4.3.2.2 Microsoft IIS

Microsoft IIS, using the "IIS Client Certificate Mapping Authentication" feature [C181], works differently than the other implementations. It does not grant access based on the certificate. Instead, it maps the certificate to a user on the system. IIS then accesses the files on the filesystem acting as this user. If a certificate is requested, IIS checks whether it is trusted by the system. If it is trusted, IIS checks whether it finds a rule that maps this certificate to a user on the system. If it does not find a mapping, it uses the anonymous user to access the files. This allows bypassing client authentication if the files are accessible by the anonymous user. In our testing, we were able to use a certificate issued by a generally trusted CA (HARICA TLS RSA Root CA 2021) to bypass client authentication, as this certificate is system-trusted but not mapped to a user on the system. Similarly, if multiple domains with different CAs are configured, the system needs to trust all of these CAs. An attacker in our model, who has a valid certificate for the accessible host, can use their certificate to bypass client authentication on the inaccessible host. When changing file permissions so that the anonymous user does not have access to the files, the bypass no longer works (see *Microsoft IIS (ACL)*).

Alternatively, the fallback to the anonymous user can be turned off (see *Microsoft IIS (no anon)*). This is done by turning off the `anonymousAuthentication` setting, which is enabled by default. However, in our testing, this also prevented access to the public site, since this would normally use the anonymous user.

Note that this behavior is not related to session resumption but can be exploited in the full handshake. Further, the behavior is very dependent on the system configuration, including file permissions and system users. We therefore do not consider this as vulnerable as the other implementations, but rather as a server that can easily be misconfigured.

4.3.3 Inconsistent Behavior

Apart from the immediate vulnerabilities, we observed inconsistent behavior between and within web servers.

Between Web Servers. No two web servers behaved the same within our tests. We found that web servers did not agree on when to resume session tickets. When changing the SNI to the inaccessible host, Apache httpd and

LiteSpeed rejected the ticket. Caddy rejected the session ticket in some cases, depending on whether the ticket was issued from another host using client authentication. Meanwhile, nginx resumed nearly all tickets except in one configuration with TLS 1.3.

Within Web Servers. More notable are the inconsistencies within single web servers. We expected servers to behave the same way, regardless of the TLS version or whether the accessible host uses client authentication.

However, we observed inconsistencies between TLS 1.2 and 1.3. Interestingly, we found that all servers are vulnerable in TLS 1.3, but not all are vulnerable in TLS 1.2. We further found all web servers except LiteSpeed to show different behavior in different TLS versions. Caddy seemingly uses separate SNI logic in TLS 1.2 and will raise a TLS alert `internal_error` if no fallback domain was provided. Further, nginx (strict SNI) did not resume tickets in TLS 1.3 when the SNI was omitted but did so in TLS 1.2.

All servers except LiteSpeed behaved differently depending on whether the accessible host required client authentication. Apache httpd, Caddy, and nginx only seemed to verify whether a client was authenticated during the initial connection, but not which certificate was used.

4.4 Real-World Evaluation

To evaluate the impact of session ticket confusion, we performed a large-scale scan of deployed servers. We only evaluated the server authentication bypass on a large scale. Scanning for client authentication bypasses is not feasible on a large scale, as popular public servers rarely use client authentication. Further, our open-source analysis revealed that findings are more likely if the attacker has prior privileges, which we do not possess. Nonetheless, we manually evaluated specific CDN services.

4.4.1 Server Authentication Scan

Methodology. To recreate the attacker scenario, we would need to buy enterprise-level features with dedicated IPs at each virtual hosting provider. However, as this is infeasible on a large scale, we aim to find sites that could already perform the server authentication bypass. That is, we want to find servers that resume tickets but serve content different from that of the ticket issuer. Concretely, we want to find a domain hosted on one IP address and resume the ticket at a different IP address. To find such

servers, we first create groups of servers we presume are likely to resume tickets. We then request a domain from the first server, retrieve a ticket, and attempt to resume it on the second server. We evaluate whether the second response matches the requested domain or whether the server responded with the content for another domain, indicating that the request was misrouted to a third party. Due to the inconsistencies between TLS 1.2 and TLS 1.3 we observed in Section 4.3.1, we perform the whole process in parallel for both versions.

Creating Candidates. We aim to evaluate popular websites, which often use CDNs to accelerate page load times. To this end, we use the Tranco Top 1M list [68] as a basis. We obtain the IP addresses for the domains on that list and probe whether the servers support TLS and session tickets by sending an HTTPS request. Additionally, we deepened the exploration by also considering related domains listed in the SANs of the certificates encountered during our scan. This increases the number of domains potentially sharing technical infrastructure and possibly their STEK. We iteratively repeated this process until no further domains were found. During this process, we store the tickets we observed in all requests.

We group the domains and IP addresses based on the retrieved tickets. Most implementations include an identifier of the used STEK at the beginning of the ticket. This key name is commonly 16 or 4 bytes long [O2]. We use the first four bytes to group servers that may use the same STEK and might allow resumption.

This still results in too many possible candidates to evaluate exhaustively.[26] Therefore, we chose to use a sampling-based approach. For each IP a domain resolves to, we propose up to 10 IPv4 and up to 10 IPv6 addresses to resume a ticket. These resumption IP addresses must share the ticket prefix in the TLS version, and the domain in question must not be hosted on these IP addresses. We instruct our database to return the IP addresses at random to reduce the likelihood of selecting only similar IP addresses.

We provide an example of selecting target addresses in Figure 15. In the example, we would attempt to resume a ticket from `a.com@1.1.1.1` at

26. We also checked this for 16B key names, and this resulted in a similar number of candidates. We chose 4B to cover more implementations.

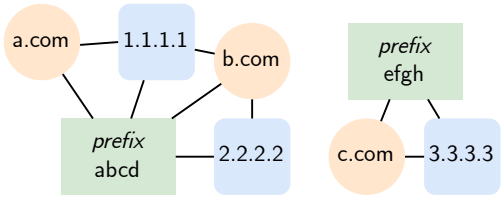


Figure 15: Exemplary grouping of domains. Domains `a.com` and `b.com` use a common session ticket prefix, and both are hosted on `1.1.1.1`. The domain `b.com` is additionally hosted on `2.2.2.2`. The domain `c.com` uses a distinct prefix and IP. We would only attempt to resume a ticket from `a.com@1.1.1.1` at `2.2.2.2`, but no other combinations in this example.

`2.2.2.2`, as it might fall back to serving different content. We would not attempt to resume a ticket from `b.com` at `1.1.1.1` or `2.2.2.2`, as we would expect both IP addresses to properly handle `b.com` in the resumption. As `c.com` does not share the prefix with another domain, we would not attempt to resume a ticket at another address.

Attempting Resumption. Each candidate in the sampling approach consists of a domain, an IP that issues the ticket, and a set of IPs at which we attempt to resume the ticket. We emulate the attack by performing an HTTP `GET /` request with the domain included as the SNI and `Host` header to the server to retrieve the ticket and HTTP response. This equals the user visiting the website. For each resumption IP, we attempt to resume the ticket and send the same HTTP `GET /` request. This equals the attacker redirecting a subsequent connection of the client. We store whether the ticket was resumed and the HTTP response.

Evaluating the Responses. We classify whether a server is vulnerable based on the response to our resumption attempt. If a ticket is rejected, the attempt is considered secure. Similarly, if the HTTP request is rejected, for example, with a status code like `421 Misdirected Request`, we consider it secure. We also assume a resumption attempt to be secure when the content served on resumption is identical or if both contain redirects to the same destination. We do not analyze attempts where either the initial or the resumption connection did not retrieve an HTTP body (e.g., only one is a redirect, or a 5xx error was returned).

For the remaining attempts, where the ticket was accepted and the server performed an abbreviated handshake, we compare the returned HTTP bodies. In case the resumption body is similar to the initial body, we

can consider the resumption as likely secure. If the body differs, we cannot automatically consider this behavior vulnerable, as some websites may serve different content on each load (e.g., CSRF tokens in HTML, dynamic content). Therefore, we aim to find the origin of the returned content. To this end, we compare the returned body with other bodies observed in domains hosted on the resumption IP. To compare the bodies, we chose the normalized Levenshtein ratio once over the whole HTML document and once only over the HTML header. We look for low similarity between the initial and resumed pages and high similarity between the resumed and another proposed origin. If the proposed origin uses the same certificate as in the initial connection, we consider the resumption secure. For other origins, we manually review the resumption. During this, we cluster the resulting candidates by CDN or AS to aid the process.

Implementation. We implemented our scanning methodology using tools from the ZMap Project [C155]. Concretely, we used ZDNS [50, C202] to resolve DNS entries to IPv4 and IPv6 addresses. We use ZMap [37, C204] with IPv6 support[27] to evaluate whether the default TLS port is open on a web server. We use ZGrab2 [34, C203] with TLS 1.3 support[28] to perform TLS handshakes and send HTTP requests to the servers.

4.4.1.1 Scan Results

We performed our large-scale evaluation from December 2024 to January 2025 using the Tranco list [68].[29] We show the number of collected hosts in Figure 16 and the classification of the sampled pairs in Figure 17. The threshold for manual review was a maximum similarity of 0.6 between the initial HTML and the resumed HTML and a minimum of 0.9 between the resumed HTML and the HTML of the assumed origin. We chose these thresholds based on an initial pass through the results, aiming to capture most cases while keeping the manual review effort reasonable. To aid the review process, we clustered the results by the AS of the involved IP addresses.

27. We used a forked ZMap for IPv6 support: see https://github.com/UPB-SysSec/zmap/ (accessed: 2026-07-19).
28. We used a forked ZGrab2 for TLS 1.3 and better session ticket support: see https://github.com/UPB-SysSec/zgrab2/ (accessed: 2026-07-19).
29. Available at https://tranco-list.eu/list/V9V2N (accessed: 2026-07-19), generated on December 5th, 2024.

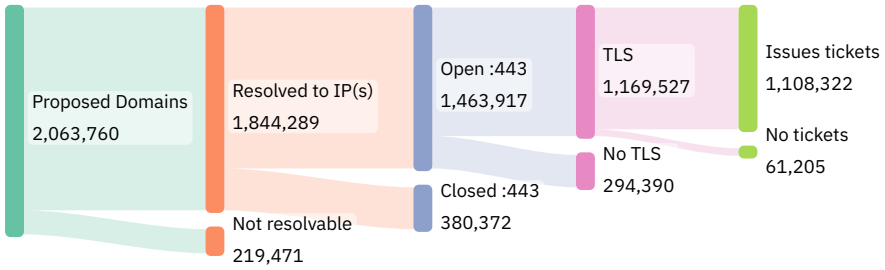


Figure 16: Number of Hosts during the discovery of ticket-issuing hosts.

We uncovered several instances of vulnerable session resumptions spanning multiple ASes. In one notable cluster, ten ASes, operated by different companies, were found to allow resumptions across ASes. Despite the organizational differences, all these companies utilized DDoS-Guard's[30] DDoS protection, strongly suggesting that the root cause lies within DDoS-Guard's service. Additionally, further vulnerable resumptions were observed within this cluster, which were not considered clearly vulnerable. One additional AS that initially appeared unconnected was also determined on further investigation to use DDoS-Guard's services, reinforcing the hypothesis of a centralized issue. A second, smaller cluster involving four ASes across three companies revealed resumptions with varying degrees of vulnerability. Two of these ASes appeared to leverage DDoS protection services from Variti,[31] the third company, suggesting another potentially misconfigured provider. Beyond these clusters, we also discovered isolated resumptions within three independent ASes, with no apparent ties to known CDN providers.

Furthermore, one instance of a technically vulnerable resumption was observed within West Texas A&M University's network, although all sessions remained confined to the same organization. We did not pursue this finding further, as the two sites seemed to belong together. Several cases remained inconclusive due to insufficient information. This included 106 resumptions across 27 ASes, most of which were located in China. One apparent false positive was identified in the Verizon Business AS, and additional false positives were observed in the networks of Google and Akamai, likely stemming from temporary error pages.

30. https://ddos-guard.net/ (accessed: 2026-07-19)
31. https://variti.io (accessed: 2026-07-19)

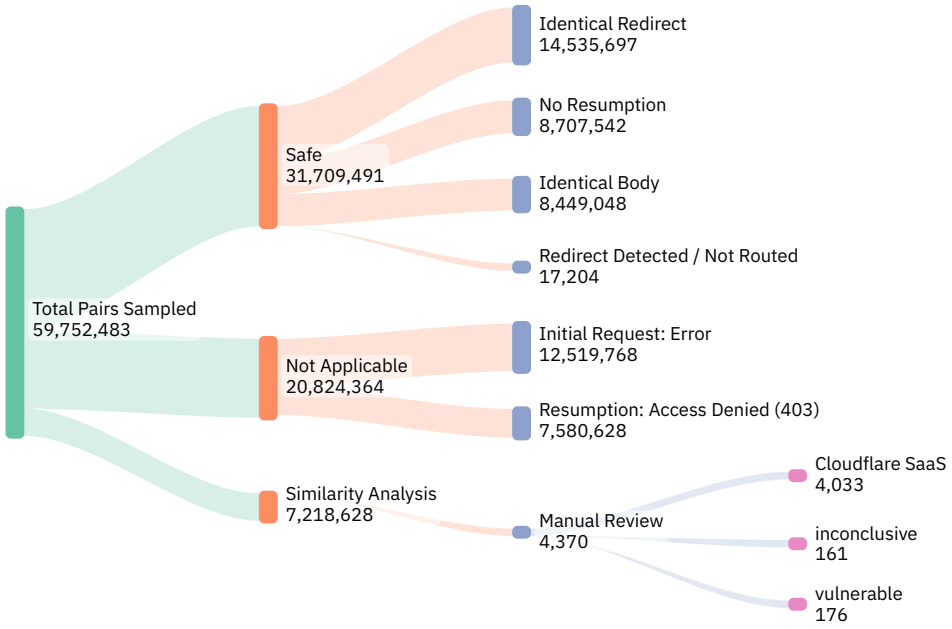


Figure 17: Analyzed session resumptions.

In a preliminary scan [72], we additionally discovered that Fastly's CDN service would resume tickets regardless of SNI but serve content based on SNI. By abusing their dedicated IP offering [C171], meant for legacy non-SNI traffic, we were able to fully enact the server authentication bypass attack.

Findings Unrelated to Tickets. Most of our findings were attributed to Cloudflare's network. Upon closer review, we determined that this issue was unrelated to session resumption and had a broader authentication impact. Cloudflare allows specific website providers, so-called *SaaS providers*, to use dedicated IPs [C165]. Any traffic to these IPs is TLS-terminated by Cloudflare, and the request is forwarded to the backend for the SaaS providers. The intention is that customers of the SaaS provider can register their domain to point to this IP address. It is then the SaaS provider's job to serve the correct content.

In our scan results, we found that when requesting any domain registered at Cloudflare on a SaaS IP, we received the correct certificate for the domain. However, the server responded with an error page from the SaaS provider, suggesting that the request had been internally forwarded to the SaaS provider. This allows the SaaS provider to read the full request and answer with any content (e.g., an XSS attack payload), enabling them to perform a full TLS-MitM. Although this vulnerability was unrelated

to our attack model and falls outside the scope of session resumption, it highlights broader security challenges and suggests that our methodology may have applicability beyond its intended purpose.

Disclosure. We disclosed our findings to the affected providers. We contacted Fastly directly in our pre-scan phase in September 2023. They fixed the issue by binding tickets to the certificate that would be sent in a full handshake (see Section 4.6). Variti and DDoS-Guard did not respond to our reports. We disclosed our findings to Cloudflare via their bug bounty program. According to Cloudflare, this behavior is part of a deprecated version that is only available to existing customers with an enterprise contract. They confirmed that the request was routed to the SaaS provider, theoretically enabling the SaaS provider to perform a TLS-MitM attack.

4.4.1.2 Email Scan

We also wanted to evaluate whether the session ticket confusion issues we found in HTTPS also exist in other protocols that use TLS, such as email protocols. Depending on the protocol, the attacker could have different goals:

(1) For protocols where the user authenticates themselves to the server (IMAP, POP3, and SMTP submission), the attacker could try to obtain the user's credentials.
(2) In the mail retrieval protocols (IMAP and POP3), the attacker could trick the user into reading spoofed emails by replying with a crafted inbox.
(3) When sending mail (SMTP submission/relay), the attacker could try to reroute the mail such that it is sent to the attacker's inbox instead of the intended recipient.

Based on the data from the IPmail scan (Section 3.5), we created candidates where a STEK might be shared between email servers. While we found candidates and could resume sessions between them, we could not validate whether this constitutes a real-world vulnerability.

In the HTTPS case (see Figure 13), we could directly compare the returned content to determine whether the server was tricked into serving content from another server (response for e.com in the figure), giving a potential attacker complete control over the response. We could further infer that the server serving the content likely also received the plaintext

request, giving the attacker the ability to impersonate the server. In the email protocols, we could not validate the attacker's goals. While the servers accepted the resumed session and served a different server banner upon connecting, we could not validate whether this constitutes a real-world vulnerability.

(1) Without access to the server's backend, we cannot determine whether an attacker could extract the user's credentials from the server. While the protocols allow for plain password authentication [S106, S120, S121], we cannot determine whether the attacker, as a customer, can observe the authentication process or whether the service provider handles it.

(2) Intuitively, the second goal of serving a crafted inbox seems similar to the HTTPS case. Unfortunately, we cannot evaluate this because we do not have credentials for the servers.

(3) We also cannot evaluate the third goal of rerouting emails without access to the server's backend. We do not want to send emails to arbitrary recipients, as this could be considered spam. Even if we sent an email and the server accepted it, we would not know whether it reached the attacker's inbox or the intended recipient, since we do not have access to the server's backend. We attempted to evaluate this by specifying the recipient address during a resumption, but most addresses were rejected with an error stating that relaying is disabled. We cannot determine whether this behavior is adjustable by the customer, which could make an attack feasible, or only by the service provider.

We concluded that evaluating our attacks on a large scale is not feasible, and we did not pursue this line of research further. We do not rule out the possibility that future work could develop a methodology to evaluate session-resumption-based authentication bypasses against email protocols automatically.

4.4.2 Client Authentication Bypass

We performed a manual evaluation of three free CDN services to determine whether they are susceptible to client authentication bypasses using session resumption. We analyzed Google's Load Balancer [C174], the Azure App Gateway [C180], and Cloudflare's API Shield [C166].

For the Google Cloud Load Balancer, we found that session tickets are disabled when client authentication is enabled. The Azure App Gateway supports session resumption with client authentication. However, we could not resume tickets across two configured gateways.

At Cloudflare, we used two accounts to set up client authentication ("mTLS") for two distinct domains. Using the same attacker model as in Section 4.3.2, we authenticated against the first domain (I) and were issued a ticket. We ran a set of tests from Table 4, intending to access the second domain (R). Our evaluation revealed that we could resume the ticket specifying SNI=R and Host=R and were served the supposedly protected content (R).

Disclosure. We disclosed this to Cloudflare developers using their bug bounty program, who disabled session resumption when using mTLS [W126].

4.4.3 Limitations of our Evaluation

Our large-scale evaluation of server authentication is constrained by several factors that may result in us underestimating the true scale of the vulnerabilities. First, we employed random sampling, which may have caused vulnerable pairs to be excluded by chance. Second, we assume that the key name exists, is located at the beginning of a ticket, and that the same STEK is always accompanied by the same key name. This may not always hold, potentially leading to further overlooked cases. Finally, our similarity score methodology may produce false negatives, further limiting the comprehensiveness of our findings. Together, these factors suggest that our analysis presents only a lower bound on the actual prevalence of the vulnerabilities. However, note that we only need to identify some and not all cases to determine if a cluster is vulnerable.

The evaluation of client authentication in CDNs was not meant to be exhaustive. We only considered a few CDNs where we could easily find documentation on client authentication. We did not consider CDNs where client authentication is a paid feature (e.g., Fastly [C170]) or the documentation indicates secure behavior (e.g., AWS ALB [C158]: session resumption is not supported in combination with client authentication). We performed our analysis manually and only evaluated a single mTLS configuration for each CDN. However, CDNs are complex, allow for a

variety of configurations, and may offer multiple services that use client authentication. Due to the complexity of CDNs (cf. [W124]), we reckon a deeper analysis may uncover vulnerable configurations.

4.5 Discussion of Root Causes

The interaction between TLS session resumption and hostnames has been a topic of some consideration in existing standards. Our findings highlight significant gaps that compromise security in modern virtual hosting and CDN environments. RFC 6066 for TLS 1.2 mandates that when an SNI is provided, servers must not accept session resumption under a different hostname than the initial SNI [S95]. In TLS 1.3, the explicit linkage between SNIs and sessions is removed. Instead, resumption is tied to the certificate presented during the initial handshake. RFC 9846 still suggests, as a "performance optimization", to only try to resume sessions if the SNI matches the initial one to improve performance [S119: Section 4.7.1]. However, this requirement is explicitly placed on the client, overlooking scenarios in which an attacker redirects requests without the client's knowledge or the client being malicious itself.

The standard completely relieves the server from any responsibility in validating session resumption, as "there is no need for the server to associate an SNI value with the ticket." [S119: Section 4.3.11] Furthermore, the standard claims that "normally, there is no reason to expect that different servers covered by a single certificate would be able to accept each other's tickets." [S119: Section 4.7.1] However, in modern CDN and virtual hosting environments, servers may accept each other's tickets even when multiple certificates are involved, potentially creating a gap between theoretical expectations and practical realities. All in all, the attacks we presented are not considered in the standard. The differences in the TLS 1.2 and TLS 1.3 standards also partly explain the observed behavior inconsistencies described in Section 4.3.3.

Another contributing factor is the intersection of TLS and HTTP standards, which operate at different layers but are intrinsically linked in practice. TLS uses the SNI for hostname indication, while HTTP relies on the `Host` header. Our open-source analysis revealed that servers sometimes fail to validate that these two hostname indications match, even though TLS only provides authentication for the SNI hostname.

This discrepancy can result in broken authentication, allowing attackers to exploit the mismatch. Additionally, RFC 9846 mandates that only the resumption SNI be exposed to application layers [S119: Section 4.7.1]. This underscores the need for TLS to ensure that authentication is maintained during session resumption. Still, this resumption SNI should be matched against the HTTP `Host` header to ensure authenticity throughout the whole request, not only within TLS.

Lastly, our attacks exploit misconfigurations in virtual hosting environments, which, while preventable through proper configuration, are a persistent and well-documented issue. Web server misconfiguration has been a factor in previous security breaches (e.g., [55]), and our server authentication bypass analysis shows that it continues to exist in the wild. Specifically, the concept of a "default host" in virtual hosting environments, where any unconfigured hostname defaults to a generic handler, is particularly critical. Large CDN providers must carefully evaluate how misdirected or improperly routed requests are managed to avoid enabling authentication bypass or other vulnerabilities.

4.6 Countermeasures

The vulnerabilities identified in this chapter stem from the inconsistent isolation of virtual hosts following TLS session resumption. Addressing these issues requires guarantees that the identities asserted during the initial handshake remain consistent throughout resumed sessions. A robust solution involves binding session tickets to all exchanged certificates and asserted identities from the initial handshake. Upon resumption, the server must ensure that these certificates and identities remain valid. Concretely, we recommend distinct countermeasures for the server and client authentication bypasses.

Server Authentication. We recommend binding the ticket to the server certificate chain. This ensures that the identity the client assumed during the initial handshake remains valid. If the certificate changes at the server, the server should not resume the session; instead, it should perform a full handshake, presenting the new certificate to the client. This allows the client to inspect the new certificate and decide whether they accept it.

Client Authentication. We recommend that the server store the certificate presented by the client in the session ticket, as recommended by the standard [S96]. This way, the server can revalidate the certificate in a resumption handshake, ensuring the authentication remains valid for the resumption.

Alternatively, the server could bind the ticket to the rules that the certificate passed initially. If the server expects different rules to pass upon resumption, it can fall back to a full handshake. This approach may be easier to implement but has worse performance for resumptions where the rules differ.

Considered Alternatives. We also considered whether binding the ticket to the SNI is a viable alternative. While it can prevent some client authentication bypasses we observed (Table 4), fallback mechanisms in multi-server scenarios may still allow authentication to be bypassed, as observed in the server authentication bypasses during the large-scale evaluation. Further, session resumption across hostnames may be used intentionally within the same certificate [75].

We also considered how binding to the server certificate interacts with certificates spanning multiple domains (A and B). In this case, our attack is not necessary; the attacker can reroute the client (requesting A) to a different server using the same certificate. If that second server is misconfigured to fall back to another virtual host (B), it will serve the certificate valid for both domains (A and B), which the client accepts.

4.6.1 Implementation Pitfalls of the Countermeasures

While the countermeasures seem simple to implement, they require a good understanding of the TLS library used. We explored how to implement our recommendations using OpenSSL 3.5 [C185]. OpenSSL offers the concept of a session context [C187]. A session context is an arbitrary 32-byte string. Tickets are bound to this context. That is, a ticket issued in one context can only be resumed in the same context and will be rejected otherwise. Servers must already set this if they wish to use client authentication and session resumption. We recommend using this to also fix the server authentication bypasses. To this end, we recommend including the server certificate in the session context.

For client authentication, OpenSSL provides functions to specify a list of certificates against which the client certificate is validated and a callback function that is called to validate certificates [C189]. Upon resumption, the certificate is not checked again, even if a different list of certificates to validate against is specified. Therefore, we recommend including the list of certificates in the session context too. Similarly, the callback is not called during resumptions. Therefore, implementations cannot rely on the callback to reject clients from accessing certain virtual hosts.

Implementing this has some caveats. The session context must be set before OpenSSL decides whether to resume the session. The SNI callback [C188] is too late, as OpenSSL has already decided whether to resume the session by the time it is called. Instead, the ClientHello callback [C186] must be used. This callback was introduced in OpenSSL 1.1.1 together with TLS 1.3 support in 2018 [C183, C184]. Within the callback, the context should be set as outlined above. As the context is limited in size, we recommend hashing all data and using the hash as the context. Furthermore, we recommend setting the context on the `SSL` object rather than the `SSL_CTX` object, as multiple contexts on a single socket may not function as expected.

We want to highlight here that the OpenSSL APIs behave differently for TLS 1.2 and TLS 1.3. OpenSSL in TLS 1.2 exposes the same SNI value to the application in the initial and resumed connection, regardless of the client's specified SNI value in the resumption handshake. Therefore, an application stays in the same security context for the resumed connection as it was in the initial handshake. In TLS 1.3, however, OpenSSL exposes the SNI value from the resumption handshake to the application. We brought this to the attention of the OpenSSL developers, who confirmed our observations and updated the documentation of the SNI callback [C188].

4.7 Further Related Work

Confusion Attacks. Although we specifically focus on session tickets, host confusion attacks involving both HTTP and TLS are nothing new: Delignat-Lavaud and Bhargavan [30] originally formalized virtual host confusion attacks in 2015, where they present a number of attacks abusing

insecure fallback hosts, shared TLS caches, or HTTP connection reuse. In particular, they highlight the difficulties of systematic evaluation and perform most analyses manually. In 2021, Brinkmann et al. [22] presented cross-protocol attacks that exploited certificates shared between services. Shared certificates have also been exploited between distinct subdomains using HTTP headers: Zhang et al. [84] described how insecure servers with a shared certificate can downgrade HTTPS connections or circumvent HTTP Strict Transport Security.

CDNs. CDNs require a lot of trust, as they act as a MitM between the content owner and client and often require the private key of the content owner. Liang et al. [58] analyzed 20 CDN HTTPS implementations and proposed solutions to sharing private keys. Ghaznavi et al. [41] categorized attacks against CDN into attacks on edge servers, routing, and origin servers. According to them, previous work on attacking CDN routing has largely focused on identifying routing infrastructure and denial of service. Although virtual host confusion has been explored within CDNs, it was based on weaknesses in HTTP request and header parsing [26]. Similar attacks that, through misconfiguration, lead to CDNs serving malicious content or leaking user traffic include web cache poisoning [W135] or deception [64, 65].

Client Authentication. Research on TLS client authentication is sparse: Parsovs [67] presented usability and security-related implementation issues in 2014. In recent years, Wachs et al. [81] and Foppe et al. [39] showcased the potential for tracking users with client certificates. A large-scale study performed by Xia et al. [83] showed that common issues within X.509 certificates themselves were also prevalent in client authentication. To our knowledge, we are the first to attempt to actively circumvent client authentication using application-level inconsistencies.

4.8 Conclusions

Our investigation into session resumption and its implications for TLS authentication guarantees revealed significant vulnerabilities. Addressing our first research question (RQ4), we explored attack models that exploit TLS session resumption to undermine authentication guarantees. Specifically, we introduced novel attacks capable of bypassing client authentication and revisited attacks on server authentication in the con-

text of session tickets and shared STEK. Our findings underscore that eliminating certificate exchanges during session resumption significantly complicates authentication by requiring authentication contexts to persist across connections. This reveals a fundamental challenge in maintaining robust security guarantees without a full handshake. In response to our second research question (RQ5), we confirmed that STEK-based vulnerabilities translate to real-world implementations and virtual environments. Our analysis uncovered authentication vulnerabilities in Apache httpd, Apache Tomcat, Caddy, nginx, and LiteSpeed and exposed faulty implementations in CDNs.

We discussed potential causes within the standards that fail to account for the attack scenarios we identified. Notably, our evaluation revealed that servers implementing TLS 1.3 exhibited higher susceptibility to these issues compared to TLS 1.2. This discrepancy may stem from explicit statements in the TLS 1.3 standard that the ticket does not need to be bound to the SNI, which is reflected in our results. TLS 1.3 was recently revised in RFC 9846 [S119], showing that updates to the standard are possible. However, the revised standard does not change the statements relevant to our findings. Even if the statements changed, adoption across deployed servers would take time. This knowledge therefore still informs future standards and implementation guidance to address these overlooked vulnerabilities.

We looked at the OpenSSL APIs and evaluated how to implement virtual hosting securely. In our testing, we found the APIs to be easy to misuse, exhibiting unintuitive behavior when sessions are resumed, and the wrong usage of the API does not throw errors. We urge library maintainers to implement misuse-resistant APIs.

Future research could extend this work by investigating the broader authentication guarantees of TLS, particularly in scenarios involving resumption across hostnames. A longitudinal study of misconfigurations and their evolution over time would also provide valuable insights into mitigating these vulnerabilities. Further, an analysis of STEK sharing based on location or service providers may allow insights into network infrastructure. We further propose researching the virtual hosting implementations of CDNs and the interaction with TLS.

Chapter 5
Partitioning Oracle Attack Against TLS

In Chapter 3, we have seen that all libraries use Authenticated Encryption (AE) to protect session tickets. The majority of libraries use encryption and an HMAC, as recommended [S96], while some use schemes such as AES-GCM or ChaCha20-Poly1305. These latter schemes have become more prevalent in recent years, due to their performance, ease of use, and protection against adaptive chosen-ciphertext attacks (e.g., padding oracle attacks [80]). An AE scheme ensures that a ciphertext is only decrypted if its authenticity has been verified. Intuitively, this suggests that a ciphertext can only be decrypted under one key. However, this is not true; Dodis et al. [31] showed that an attacker can create ciphertexts that are valid under two keys. The general core problem is that AE schemes, such as AES-GCM, are non-committing. That is, they allow a ciphertext to be valid under multiple keys, resulting in different plaintexts.

In 2021, Len et al. [57] exploited this non-committing property of AE schemes and introduced a key-recovery attack called partitioning oracle attack. It is a chosen ciphertext attack targeting AE schemes such as AES-GCM [S94] and ChaCha20-Poly1305 [16, 17, S111]. The attack allows an attacker to create an authenticated ciphertext valid under multiple keys. Using an oracle that accepts the ciphertext and outputs whether the targeted key is in an attacker-defined set of keys, an attacker can recover the key using binary search. Len et al. [57] showed that this attack is applicable to the Shadowsocks [C154] and OPAQUE [52] protocols, which use keys derived from passwords.

TLS [S115, S119] also supports AES-GCM and ChaCha20-Poly1305. Len et al. [57] mentioned that if OPAQUE is used with TLS, the attack may still be applicable. Armour and Cid [11] mentioned TLS in the context of IPsec VPNs and OpenVPN. They stated that AES-GCM and ChaCha20-Poly1305 are not allowed to be used with manual keys and hence would not be applicable. Neither group made a statement about TLS on its own.

TLS uses a Key Derivation Function (KDF) with random inputs from both parties to generate (pseudo-)random session keys in each connection.

Since malformed messages cause the connection to be closed, only one query can be made per connection and key. It therefore seems that the attacks do not apply to TLS. However, we have seen that the STEK is not always of high entropy and that some libraries use AES-GCM and ChaCha20-Poly1305 to protect session tickets (cf. Chapter 3). Key-recovery attacks, such as partitioning oracle attacks [57], pose a significant risk, as a successful attack would allow an attacker to recover the STEK.

In this chapter, we analyze whether the partitioning oracle attack can be used to recover the STEK, what preconditions are necessary for this attack to work, and how to quantify the impact of such an attack. This leads us to the following research question:

RQ6: Are partitioning oracle attacks practically applicable to TLS?

To answer this research question, we analyzed open-source libraries and performed a large-scale server scan. In our analysis of open-source libraries, we investigated whether the provided TLS servers allow an attacker to distinguish valid from invalid ciphertexts and practically allow for the construction of partitioning oracles. To this end, we created session tickets with AE ciphertexts valid under multiple keys, sent them to the servers, and analyzed their behavior. Interestingly, the behavior of the analyzed servers strongly differed. Vulnerable servers responded with different alerts or indicated the ticket validity through different handshake behavior. Other servers were shown to be secure because of their strict format checks on the decrypted tickets and strict limitations, practically showing how to apply effective partitioning oracle attack countermeasures. We could adjust our attack to this such that we can exploit vulnerable library behavior in the lab setting. Based on this knowledge, we prepared a large-scale scan to evaluate whether vulnerable libraries are used and exploitable in the wild. We found the market share of affected libraries to be small and did not find low-entropy STEKs in the wild that would enable a practical attack.

Contributions.

- *Attacks*: We identified how partitioning oracle attacks are applicable against TLS session tickets (Section 5.2).

- *Open-Source Analysis*: We analyzed seven TLS libraries that issue session tickets encrypted using AES-GCM or ChaCha20-Poly1305, the algorithms we can target using the partitioning oracle attack (Section 5.3). Four libraries provided an oracle, and a fifth library provided a limited oracle that works only with a small target key set. The reasons for not providing an oracle are session ticket length limitations and strict plaintext format requirements.
- *Large-Scale Analysis*: We performed a large-scale scan to determine the real-world applicability of our attacks (Section 5.4). We found no deployed servers that were affected and that used a STEK contained in our tested key set.

In our original publication, we also slightly improved the heuristic of the splitting ciphertext algorithm for ChaCha20-Poly1305 [O7: Appendix A.3].

Disclaimer on Originality

This chapter is mostly based on [O7]. For more details, see Section 1.5.1.

5.1 Background on Partitioning Oracle Attacks

Partitioning oracle attacks were introduced by Len, Grubbs, and Ristenpart in 2021 [57]. They target the AEAD schemes AES-GCM and ChaCha20-Poly1305. Their attack allows recovering the symmetric key used by a server. For this, the attacker creates ciphertexts that validly decrypt under multiple keys. That is, the decryption of the ciphertext does not raise errors under several keys but returns a plaintext. They call these ciphertexts *splitting ciphertexts* (or key-colliding ciphertexts). The attacker sends the splitting ciphertext to the server, observing its behavior. If the attacker can distinguish between cases where the ciphertext was decrypted but the plaintext was possibly malformed and cases where the ciphertext failed to decrypt, they can recover the key using binary search.

Since Carter-Wegman MACs are polynomial MACs, splitting ciphertexts can be created by solving linear equations. Armour and Cid [11] improved this attack for attackers that can obtain a valid ciphertext (including nonce and tag). Their approach allows precomputing a *ciphertext mask* for a given key set without needing the ciphertext. Once the valid ciphertext is obtained, the mask can be used to create a new splitting

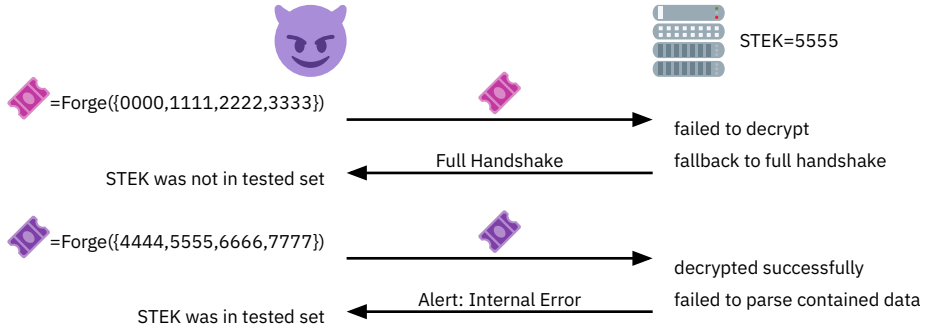


Figure 18: Exemplary overview of a partitioning oracle attack on TLS session tickets. The attacker forges two tickets using different key sets. The server behaves differently when the key set contains the correct STEK. Using binary search, the attacker can recover the STEK.

ciphertext that validly decrypts as long as the decryption key is in our key set. They further outline how to pass format checks by including prefix blocks from the valid ciphertext. They also state that bit flips can be induced in this process to help pass format checks.

5.2 Partitioning Oracle Attacks on TLS

Our main goal is to apply partitioning oracle attacks to TLS, in particular to TLS session tickets. Partitioning oracle attacks allow generating splitting ciphertexts that validly decrypt under multiple keys. We utilize the algorithms outlined in Section 5.1 to generate splitting ciphertexts. As TLS servers send a ticket, we can use parts of their ciphertext when generating a splitting ciphertext. We send the modified ticket to the server and observe its behavior to classify whether the STEK was within the tested key set. We provide a schematic overview of the attack in Figure 18.

Attacker Model. We assume the attacker can connect to the TLS server and retrieve tickets by performing a full TLS handshake. The ticket format is known to the attacker. They know which data from the ticket is used to decrypt and validate it, for example, whether the key name is used as AAD. The attacker can send a (possibly modified) ticket to the server and observe the server response.

Impact. The attacker aims to recover the key (STEK) used to encrypt the ticket. With the STEK, the attacker can decrypt tickets. On its own, this is not useful, but if the attacker can observe (passively) or modify (actively) other connections, they can use the STEK to mount further attacks (see

Chapter 3). A passive attacker can decrypt TLS traffic, as the key exchange was skipped and the key material from the ticket is used. An active attacker can impersonate the server, as the authentication is delegated to the STEK.

Oracle. The behavior difference may be as simple as the server using the ticket to resume the session or rejecting the ticket and performing a full handshake. We can observe this by checking which TLS messages the server responds with. In other cases, though, the server might reject the ticket for resumption, even if the key set contained the correct STEK. This may be because the server requires the decrypted bytes to follow a specific structure.

Depending on the ticket construction, the strength of the oracle varies. The strongest form is an oracle that allows the ticket to be completely precomputed. This is the case if the server uses no server-specific AAD. The attacker can then precompute the splitting ciphertexts and send them to a multitude of servers to recover the STEK. The oracle is weaker if the server uses server-specific AAD. In this case, the attacker can only precompute the splitting ciphertexts for this specific AAD. A common example is the key name, which is unique per server. In this case, the attacker first has to retrieve the key name from a valid ticket before they can compute the splitting ciphertexts. An oracle can also limit the attack by limiting the ciphertext length it accepts. Since a splitting ciphertext's length grows linearly with the number of keys, a limit on the ciphertext length reduces the number of keys that can be tested in one query.

5.3 Library Evaluation

Recall that there are two prerequisites for successfully performing partitioning oracle attacks. First, the target server must support AES-GCM or ChaCha20-Poly1305 for protecting session tickets. Second, the server implementation must provide an oracle that allows an attacker to distinguish valid and invalid ciphertexts. This section analyzes open-source TLS libraries and their potential vulnerabilities to the described attacks.

5.3.1 Methodology

We only evaluated TLS libraries using AES-GCM or ChaCha20-Poly1305 (see Figure 7). For example, we did not test popular libraries like OpenSSL or BoringSSL, as they use AES-CBC with HMAC-based authentication.

This resulted in seven open-source libraries (see Table 5). Each library includes a test server program that we used to test the library's default behavior. We configured each server with a known STEK and created two sets of keys: The first one does not contain the known STEK. Tickets created using this set should fail decryption. The second key set also contained the known STEK, where decryption should succeed.

We then analyzed the session ticket format. Here, we focus on checking which data is included in the MAC computation. Depending on whether AAD is used, the impact of the oracle changes. If the MAC is only computed over data we know prior to querying a server (e.g., only the ciphertext, no AAD), we can precompute a splitting ciphertext and use it against every instance of this library. If further data (e.g., key name or prefix blocks from a valid session ticket) is also used as input for the MAC, we have to adjust the splitting ciphertext for each different input. We retrieved a session ticket from the server to obtain necessary information, like the current key name. We generate two session tickets from splitting ciphertexts matching the server's required format of a session ticket using both key sets. The session ticket from the first set should be rejected by the oracle. The other one should be accepted by the oracle. We observed differences in the server's behavior to evaluate whether the server provides an oracle for the attack. Our analysis was aided by reviewing the source code.

5.3.2 Results

We summarize the behavior of the analyzed libraries in Table 5. The table highlights the server behavior when processing session tickets encrypted with valid and invalid keys. Interestingly, each library behaved differently.

Servers providing partitioning oracles. *MatrixSSL* provides an oracle by responding with different alert messages. If the server cannot decrypt the ticket but the key name matches the current one, it responds with a *Bad Record MAC* alert. Otherwise, if the server can decrypt the ticket but cannot process it further, it responds with an *Internal Error* alert. The nonce and key name are not authenticated (see Figure 7), allowing an attacker to precompute the ciphertext for a ticket once and use it against any server. Note that we could observe this behavior only for TLS 1.3;

Table 5: List of analyzed libraries, including their version. We list whether they can be used as a partitioning oracle and detail how the responses differ. An oracle is completely precomputable if we can precompute a splitting ciphertext for a server without first connecting to the server. This is the case if the server only includes AAD that is known prior to querying the server (e.g., key name).

Library	Version	Algorithm	Oracle	Response to ...	
				Valid Ticket	Invalid Ticket
Botan	3.1.0	AES-GCM	—[a]	—	—
JSSE (OpenJDK)+	26.0.1+8	AES-GCM	●	Alert: Handshake Failure	Full Handshake
MatrixSSL	4.6.0	AES-GCM	●	Alert: Internal Error	Alert: Bad Record MAC
mbedTLS	3.4.0	*configurable*	◐	Abbr. Handshake	Full Handshake
Rustls	0.21.1	ChaCha20-Poly1305	◐	Abbr. Handshake	Full Handshake
s2n	1.3.46	AES-GCM	—[b]	—	—
WolfSSL	5.6.0	*configurable*	○[c]	NewSessionTicket	No NewSessionTicket

a : Complex plaintext format check (ASN.1).
b : Implicit AD and exact length check.
c : Ticket length limited to 178B.
+: Added compared to [O7].

● Completely precomputable oracle.
◐ Provides Oracle.
○ Provides limited Oracle.
— No Oracle.

MatrixSSL does not use AES-GCM in TLS 1.2 but uses AES-CBC with HMAC-SHA256. Hence, the partitioning oracle only exists in TLS 1.3.

JSSE (OpenJDK) uses AES-GCM for session ticket encryption and authentication. If the server can decrypt the ticket, it fails to parse the session, causing a TLS alert to be sent. Otherwise, the server ignores the ticket and continues with a full TLS handshake. Therefore, the server provides an oracle for valid and invalid tickets. Due to a programming mistake, the AAD is empty.[32] This allows an attacker to fully precompute a splitting ciphertext and use it against any server.

mbedTLS supports several AEAD algorithms, including AES-GCM and ChaCha20-Poly1305. It provides an oracle; depending on the ticket's validity, the server continues with an abbreviated or a full TLS handshake. However, the real attack is limited by the ticket structure. The nonce and key name are authenticated (see Figure 7). Hence, an attacker can only precompute a ciphertext mask, but not a colliding ciphertext.

Another challenge we needed to deal with was the strict internal ticket format. The plaintext of mbedTLS session tickets contains another length field for an internal buffer, usually set to 0. After parsing the plaintext,

32. The code prepares a `ByteBuffer` object into which it inserts the key number and the compressed field. However, the buffer is incorrectly passed to the `updateAAD` method, resulting in empty AAD. This bug was introduced in May 2025 (commit `99048c3d`).

mbedTLS verifies that no data is left to parse. Increasing the ticket size with the partitioning oracle attack would invalidate the ticket, resulting in a full TLS handshake. To create a valid session ticket with a splitting ciphertext, we exploited the counter mode malleability. As AES-GCM internally uses AES in counter mode, the resulting ciphertext is malleable. That is, any bit flipped in the ciphertext is also flipped in the plaintext with no other bits being modified. Given a valid session ticket provided by an mbedTLS server, we could flip the internal length field, causing an internal buffer to consume excess data, and successfully create a valid session ticket with a splitting ciphertext.

Rustls provides an oracle. Similarly to mbedTLS, the ticket validity can be detected by observing whether the server performs an abbreviated or a full TLS handshake. Since Rustls uses ChaCha20-Poly1305, the attack is limited by the splitting ciphertext algorithm performance for ChaCha20-Poly1305, which we cover in Section 5.3.4. To pass plaintext checks, we must include five prefix blocks from a valid ticket in our computation.

WolfSSL uses ChaCha20-Poly1305 by default, but it may be configured to use AES-GCM. It provides an oracle for both configurations. If the server receives a ClientHello message with a validly encrypted session ticket, it generates a new ticket and sends it to the client in the TLS handshake in a NewSessionTicket message. Otherwise, the server does not send a NewSessionTicket message. While this server behavior provides an oracle, the real attack impact is strongly limited. The limitation is caused by the maximum session ticket length; WolfSSL only accepts tickets with a length of up to 178 bytes. Due to the properties of the partitioning oracle attack, the number of keys we can test is limited to six.

Disclosure. We disclosed our findings to the maintainers of MatrixSSL, mbedTLS, Rustls, and WolfSSL on June 24th, 2024. MatrixSSL did not respond to our disclosure, which we attribute to the fact that the library is no longer maintained. mbedTLS accepted our report, but they did not follow up on this. We did not find any changes in the session ticket encryption in the library since our disclosure. Rustls added the key name to the session ticket format, included it in the AAD, and now rejects long tickets [C163]. They additionally added an alternative implementation using AES-256-CBC with HMAC-SHA256 for session ticket encryption.

WolfSSL added an internal length check [C191] and an option to use AES-CBC with HMAC-SHA256 for session ticket encryption [C192].

We disclosed our findings to the maintainers of JSSE (OpenJDK) on June 22nd, 2026. The disclosure process is still ongoing at the time of writing this thesis.

Secure servers. We could not trigger any different behaviors when testing *Botan*. We attribute this to the session ticket's strict ASN.1 BER [S103] encoding format with expected types, lengths, and end verification. While fields of arbitrary lengths exist, we did not find any means to exploit them like with mbedTLS. We cannot rule out that it is possible to forge meaningful plaintexts. Another potential possibility for creating an oracle would be to use precise timing attacks to distinguish different code paths. However, timing attacks are out of the scope of this thesis.

s2n contains two effective countermeasures against partitioning oracle attacks. First, the server only allows a ticket length of exactly 105 bytes,[33] 61 of which are the encrypted state, limiting the key set to three keys. Second, it uses implicit additional data. AAD is part of the authentication tag calculation. Usually, they must be sent along with the message because the decrypting party needs the same AAD to verify the tag and, with it, the message's integrity and authenticity. With session tickets, however, the encrypting and decrypting parties are the same entity. The server can therefore include locally stored AAD in the tag calculation without sending it to the client. When it later receives the ticket, it can use that same value again to verify the tag. This AAD does not need to be unique per ticket and can be stored alongside the STEK. It can then be used for all tickets encrypted with that key. We found that s2n implements this approach by generating a random 12-byte value called `implicit_aad` when it creates a STEK. It then uses the key name and this implicit AAD as AAD when encrypting a session ticket. Because this implicit AAD is never sent to the client, it remains unknown to both clients and attackers. Without this value, we cannot construct a valid splitting ciphertext.

33. The format was changed in August 2024 to be nonce-reuse resistant in https://github.com/aws/s2n-tls/pull/4663 (accessed: 2026-07-19). After this change, tickets are 138 bytes long.

5.3.3 Attack Performance

Our results show that multiple libraries are potentially vulnerable to partitioning oracle attacks. However, given a session ticket, it is always possible to use bruteforce to find the used key [O2]. We therefore compare the performance of a partitioning oracle attack with that of a bruteforce attack to determine whether our attack actually provides an advantage.

Three of five libraries required us to set prefix blocks or AAD. Therefore, the splitting ciphertexts must be adapted to each server to perform the attack. We evaluated the time required to generate a splitting ciphertext using the method of Armour et al. (see Section 5.1) from a given ciphertext mask. We also evaluated the time needed to perform a bruteforce attack to decrypt a given message with the same number of keys as the corresponding splitting ciphertext generation test. The generation of splitting ciphertexts with one, three, and five prefix blocks was evaluated.[34] The bruteforce attack decrypts a message of length 96 bytes, excluding the tag and no AAD. We performed the evaluation using our attack implementation and Java's built-in cryptography APIs. The machine used for the evaluation was a DELL Precision 3630 with Intel(R) Xeon(R) CPU E5-2695 and 4 GB of RAM. Our implementation made use of the built-in AES instruction set (AES-NI). Besides, we did not use any parallelization or any form of optimization.

Figure 19 shows the results of our evaluation. Our performance test shows that a partitioning oracle attack is only beneficial if we use one splitting ciphertext against multiple servers without needing adaptations. This might not only be the case in our field of interest but is likely a general property of the attack. We still want to highlight that when targeting many servers, for example, the whole IPv4 space, a one-time precomputation of a splitting ciphertext may be more efficient than a bruteforce attack.

5.3.4 Attack Limitations Against ChaCha20/Poly1305

While Rustls always uses Poly1305 to secure its session tickets, WolfSSL and mbedTLS may use it depending on their configurations. Unfortunately, the attack against Poly1305 is limited. The algorithm of Len et al. [56] can only compute splitting ciphertexts for sets of 10 keys with a success rate of 5 %. We improved the attack performance to reliably

34. Note that prefix blocks and AAD are the same performance-wise.

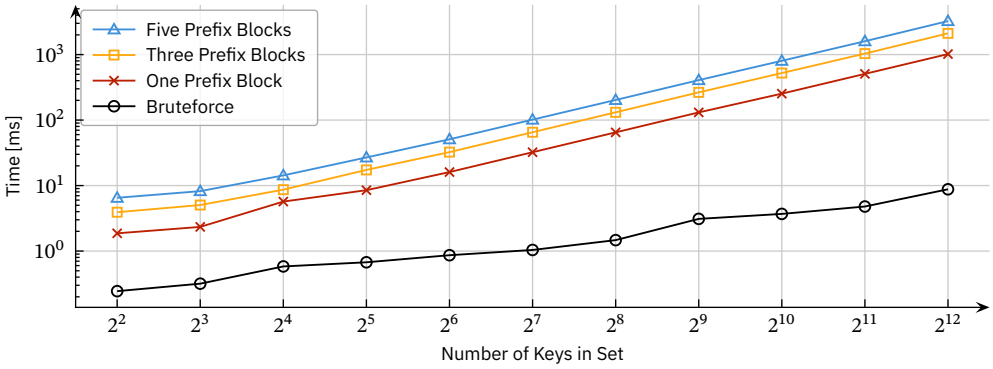


Figure 19: Time required to compute a splitting ciphertext from a precomputed ciphertext mask using the method by Armour and Cid [11] compared to the time required to decrypt a given message (96 bytes plus tag) with all keys in the set (i.e., bruteforce). Network time to query the oracle is not included.

construct splitting ciphertexts for sets of ten keys with arbitrary nonces. For the details, we refer to our original publication [O7: Appendix A.3].

5.4 Large-Scale Evaluation

To assess the real-world impact of partitioning oracle attacks, we performed large-scale scans. We evaluated how commonly the affected libraries are used in the wild to estimate the theoretical impact. We then performed a partitioning oracle attack against them with a fixed key set to evaluate whether the deployed libraries are vulnerable and use a weak STEK.

Note that we performed the evaluation for the original publication in 2024 [O7]. We only discovered the vulnerability in JSSE (OpenJDK) when revisiting the attack for this thesis. Therefore, this large-scale evaluation does not scan for JSSE (OpenJDK).

5.4.1 Library Identification

Session tickets are opaque to the client. They do not contain any information identifying the issuing server or software. However, in the case of the seven libraries we evaluated, the session ticket format allows us to identify four with reasonable certainty. In particular, we are interested in identifying servers running either mbedTLS or MatrixSSL. As shown in the previous section, those two libraries provide an oracle and use AES-GCM, which enables a performant partitioning oracle attack.

Botan prepends a string of magic bytes to its session tickets, allowing easy identification. MatrixSSL (in TLS 1.3), WolfSSL, and mbedTLS have length fields in their session ticket. These are two-byte fields indicating either the total length of a ticket or a part of it. To test whether we have one of those libraries, we can evaluate the bytes where we expect a length field and check whether they match the ticket length. If they do, we can obtain a second and third ticket and check the presence of the length field with them to ensure this is not a random occurrence. We assume that three attempts are sufficient. MatrixSSL's length field is at position 28, WolfSSL's is at position 32, and mbedTLS's is at position 16. Therefore, we can distinguish and identify those three libraries with reasonable certainty.

The library s2n before August 2024 only issues session tickets of precisely 105 bytes in length. We assume that any ticket of length 105 bytes is issued by an s2n instance. While this is not unambiguous, we use this metric as it is the best option.

While our scan did not consider JSSE (OpenJDK), we still outline how to identify its tickets. JSSE (OpenJDK) includes a byte to indicate whether the session ticket is compressed. This is positioned at byte 20 (after the key name and nonce). All ticket formats have random data at this position. Therefore, finding a server that repetitively issues tickets with the same value at this position is a strong indicator of JSSE (OpenJDK).

Rustls does not have any identifying features.

5.4.2 Scan Setup

Our attacks were implemented using TLS-Attacker [O1, 70, C196]. The large-scale scans were done using TLS-Scanner [C198] and TLS-Crawler [62], based on the TLS-Attacker framework. We used the Tranco Top 1M list [68].[35] Additionally, since we target servers running either mbedTLS or MatrixSSL, both of which are designed for embedded systems, we scanned one million random IPv4 hosts that responded to a ZMap [37, C204] scan on port 443, aiming to find more of those.

35. Available at https://tranco-list.eu/list/KJJ6W (accessed: 2026-07-19), generated on March 23rd, 2024.

Table 6: Results from our scan.

	Tranco	Random IPs
Total hosts scanned	764,078	703,575
Supports Ticket Resumption	650,477	432,475
Classified as …		
… MatrixSSL (●)	0	1
… mbedTLS (◐)	2	52
… WolfSSL (○)	11	144
… s2n (—)	68,890	126,877

●, ◐, ○, —: Whether the library offers an oracle. See Table 5.

5.4.3 Methodology

We tested session tickets with splitting ciphertexts that were generated from two key sets. The first key set is the same as used in our previous study in 2023 [O2], which we list in Appendix A.2. The second key set contains all keys from the first key set and additional constant keys, counting keys, and keys with all bytes zero except the first byte. We list the second set in Appendix A.3. Since the length of the splitting ciphertext is proportional to the number of keys tested, and some servers fail to handle long tickets [O7: Appendix A.1], we opted for a smaller and a larger set.

During our scan, we analyzed supported protocol versions for each server. For TLS 1.2[36] and (if supported) TLS 1.3, we tested whether the server issues session tickets and allows resumption. Based on the received ticket, we identified the (likely) used library using the methodology described above.

As the computation of a colliding ciphertext is expensive, we aimed to minimize these computations. Based on our analysis from Section 5.3, we can perform the following precomputations: For MatrixSSL, we can precompute the whole contained ciphertext as neither the key name nor the nonce is authenticated (see Figure 7). For mbedTLS, we need to include the key name and nonce in the AAD. Therefore, we can only precompute a ciphertext mask. Further, as mbedTLS supports AES-128-GCM, AES-192-GCM, and AES-256-GCM, we precompute a mask for each.

5.4.4 Scan results

Table 6 lists the results of our scan. We found several s2n servers, which, however, are not vulnerable to partitioning oracle attacks. Additionally, we

36. If TLS 1.2 is not supported, TLS 1.1 or TLS 1.0 may be tested instead. Only the highest version of TLS 1.0–TLS 1.2 supported by the server is tested.

found some WolfSSL servers, which have only very limited vulnerabilities and were not tested further. We found two servers during the Tranco scan and 52 servers during the Random IP scan that used mbedTLS. When queried with splitting ciphertext session tickets, they all responded with the expected oracle reject behavior (fallback to a full handshake). Therefore, we conclude that none of these used a STEK from our test key sets. We found only one server using MatrixSSL, which also responded with an oracle reject when queried with a splitting ciphertext session ticket. Again, we concluded that it does not use a STEK from our test key sets.

The scans show that the market share of the affected libraries is relatively small. We only found a handful of servers using vulnerable libraries, and for those, we could not recover their STEK.

5.5 Related Work on Partitioning Oracle Attacks

Partitioning oracle attacks were introduced in 2021 by Len et al. [57]. They showed how to create splitting ciphertexts without needing a valid ciphertext. When the server decrypts this ciphertext, the resulting message is pseudorandom. Armour and Cid [11] improved this attack to give control over the plaintext. To execute plaintext modifications, the attacker requires one valid ciphertext. Neither work explicitly considers the applicability to TLS. Armour and Cid [11] stated that manual keys in TLS-based VPNs are prohibited, and hence partitioning oracle attacks are not applicable.

Partitioning oracle attacks exploit the fact that AES-GCM and ChaCha20-Poly1305 are AEAD schemes without key commitment. That is, it is possible that a ciphertext decrypts without error under multiple keys. Various works have exploited them and proposed how to construct key-committing schemes. Dodis et al. [31] were the first to exploit this property against Facebook's message franking [43]. They provided proof of concept that a ciphertext can validly be decrypted under two keys to two different meaningful plaintexts. Albertini et al. [9] exploited these schemes in a variety of systems. When first exploiting AES-GCM, Dodis et al. [31] also show how to construct key-committing AEAD schemes. Albertini et al. [9] show how to fix existing schemes to be key-committing. Bellare and Hoang [13] show how to modify existing schemes to be key-committing. Tordsson [77] proposed to replace the underlying

mathematical fields of AES-GCM and ChaCha20-Poly1305 with non-field rings.

Another way to prevent partitioning oracle attacks is to use high-entropy key material. We discuss studies that identified insufficient entropy for generated key material in Section 3.6.4.

5.6 Conclusions and Future Work

We investigated the possibility and effectiveness of using a partitioning oracle attack against TLS. While TLS uses fresh, high-entropy key material, in Chapter 3 we have seen that this is not necessarily the case for session tickets [O2]. We have shown that partitioning oracle attacks are possible against session tickets, answering RQ6. However, it is only feasible where format checks are not too strict. When length limits are in place, they severely limit the attack. Moreover, one library (s2n) included further implicit additional data, which thwarted our attack. For other libraries, we can bypass certain plaintext format requirements using prefix blocks or malleability. If the format was too strict, no attack was possible. We can adapt our splitting ciphertext session tickets to a library's ticket format, to the additional data expected by the server, and to the prefix blocks necessary to pass plaintext format checks. However, the attack is slower than a bruteforce attack against a single server.

Our results show that partitioning oracle attacks are only effective if a once-generated splitting ciphertext can be used against many servers without modification or recomputation. Therefore, a partitioning oracle attack should be directed against systems that do not expect specific/unique additional data or certain format requirements in the underlying plaintext within the same application that decrypts the message. We still evaluated whether the attack applies in the real world. Due to the low market share of the affected libraries in HTTPS servers, we could not find vulnerable servers with low-entropy key material. Further, due to the performance, we consider partitioning oracle attacks against TLS to not be feasible for most libraries.

The finding that partitioning oracle attacks are viable against TLS through session tickets highlights how a lenient standard can affect security. The algorithms recommended in the standard are not vulnerable.

However, since server implementations are free to choose any algorithm, we were able to find vulnerable implementations.

Future work could regard improved algorithms to generate splitting ciphertexts. While the algorithm proposed by Armour and Cid [11] showed significant improvements for AES-GCM, we still cannot generate splitting ciphertexts for ChaCha20-Poly1305 that target more than ten keys.

To protect against partitioning oracle attacks, the approach by Tordsson [77] to replace the fields of AES-GCM and Poly1305 with non-field rings should be investigated further. If the security of that approach can be proven to be equal to the current versions of AES-GCM and Poly1305, those variants should be standardized.

Chapter 6
Discussion

Throughout our works we have confirmed previous criticism of session tickets [30, 71, 74, W145, W147]. We have shown that this is not only a theoretical concern, but that critical vulnerabilities existed in real-world implementations and deployments. These completely undermined the security guarantees of TLS, in particular the authenticity and confidentiality of the connection.

Concrete Findings. We found several instances of low-entropy key material, even affecting large vendors like AWS, and one instance of a reused keystream (Section 3.5). This indicates that the key material was not properly initialized and that its entropy was insufficient. We found that the session tickets of different virtual hosts were not properly isolated, which allowed us to bypass server authentication in several large CDNs (Section 4.4), among them Fastly. We were also able to bypass client authentication in all analyzed open-source servers (Section 4.3) and Cloudflare (Section 4.4.2). For server authentication, we found that the standard does not clarify when servers are allowed to resume a session. For client authentication, we found that the standard recommends including the client's identity in the ticket. However, some implementations (e.g., OpenSSL) do not re-evaluate the client's identity when resuming a session (Section 4.6.1). Our last finding was that partitioning oracles are, contrary to prior belief, applicable to TLS. These attacks are enabled by session tickets, again demonstrating the security risks introduced by the TLS session ticket mechanism (Section 5.3).

Dangerous APIs. A recurring theme throughout the chapters was that the APIs of the underlying libraries could be misused. In Chapter 3, low-entropy key material was generated, and the TLS library accepted this insecure key material without any warning. In Chapter 4, the APIs did not enforce isolation between different virtual hosts and resumed tickets across authentication boundaries. In Chapter 5, the algorithms used for encryption and authentication have specific weaknesses, which the libraries do not defend themselves against.

While we acknowledge that low-level TLS libraries inherently require some expertise to use, we argue that the libraries should be designed to be hard to misuse. Critical misconfigurations should not be possible without explicitly enabling insecure behavior. These cryptographic libraries are widely used. Therefore, it is partially their responsibility to enforce secure behavior. A positive example are nonce-reuse-resistant algorithms [42, 44, 47, S99] and the nonce-reuse-resistant ticket constructions of Botan and s2n. Even if a nonce is generated twice, it is unlikely that the security of the connection is compromised. Generally, defense-in-depth mechanisms in the libraries could have prevented our uncovered vulnerabilities, and we urge library maintainers to implement such mechanisms.

Additionally, we have observed that APIs changed over time. This caused implementations that were developed against older versions of the APIs to become insecure after the APIs were updated. In nginx, the internal format of the key material changed, which could cause the AES key to be uninitialized (see Section 3.6.1). In OpenSSL, the APIs changed such that sessions may unintentionally be resumable across different virtual hosts (see Section 4.6.1). We argue that updates should rather break the functionality than silently break security. Otherwise, security issues may go unnoticed for a long time.

Lax Standard. Apart from the libraries, we also found that the standards themselves are too lax. RFC 5077 [S96] allows for arbitrary ticket constructions, which can be secure but can also fail. While the standard provides a secure construction, it does not explain the design choices, nor does it state requirements for the format or algorithms. We argue that when the standard allows for deviations, it should be more explicit about the security requirements. For example, the ticket specification should state that the authenticated encryption algorithm must be key-committing, that is, a ciphertext must not be valid under multiple keys. The lack of key commitment enabled partitioning oracle attacks to be applicable to TLS via session tickets.

Protocol Design. A key problem with session tickets is that they are not auditable and can silently break security. This is a problem in the protocol design, as the STEK is only known to the server. All other symmetric keys in TLS are derived from the key exchange and therefore known to

both parties. While the server may generate weak private keys in the key exchange, the client can audit the public key, as done by previous works [32, 45, 66, 78, 86]. As discussed in Section 3.6.4, this is not the case for the STEK. The only way to audit the STEK is to bruteforce the format and key. It can therefore silently undermine all security guarantees of TLS. We have seen that this is not only a theoretical concern, but that it has happened on a large scale in the wild. Protocols should be designed such that it is feasible to audit all security-critical components to prevent them from silently undermining the security.

Another problem we have seen is that the standards do not consider the interaction of different protocols. TLS is a transport-layer protocol, and HTTP is an application-layer protocol. Both protocols have their own hostname indication: TLS has the SNI, and HTTP has the `Host` header. The standards do not provide sufficient guidance on how these two protocols interact. As we have seen, this led to broken authentication.

The standard also does not consider the interaction of session resumption with changes in the indicated hostnames. It merely dismisses them, as the standard assumes two unrelated domains would not be able to resume sessions. We have shown this assumption does not hold in practice and can break the authentication guarantees of TLS. Standards should be more explicit about the interaction of different protocols. In this case, the standard should explicitly state when sessions are allowed to be resumed and how to handle changes in the indicated hostnames on the TLS and HTTP layers.

Limitations. Our works focused on open-source and HTTPS implementations, which are only a small portion of the TLS ecosystem. We did not cover a variety of other proprietary implementations or other TLS-enabled protocols. Zhang et al. [85] have shown that HTTPS is comparatively secure, while other ecosystems often are not encrypted at all. Our focus on open-source implementations might have missed proprietary implementations that are vulnerable to the same attacks. Furthermore, we could not automatically evaluate our client authentication attack at scale and only evaluated it manually in a few CDNs. Our findings are therefore non-exhaustive and should be understood as a lower bound of the security issues present in the TLS ecosystem.

Applicability to DTLS and QUIC. In this thesis, we focused on session resumption in TLS (over TCP). Since DTLS and QUIC support the same session resumption mechanism, our results also apply to these protocols. Additionally, these protocols need to protect themselves against being used for DoS amplification. For this, they need to validate the sender's network address and use a mechanism that works similarly to session resumption: The server issues a token that the client must echo back to the server, proving that the client can receive messages at the claimed network address [S104: Section 8, S116: Section 5.1]. To reduce server-side state, the server may encode the necessary information in the token by encrypting and authenticating it using a key shared across all tokens, similar to session tickets. We expect that some of our findings can also be applied to this mechanism.

Chapter 7
Conclusions and Future Work

Our analyses have uncovered exploitable vulnerabilities in session tickets, making them a risky feature of TLS. We have seen APIs of underlying libraries change over time. After the update, using the old APIs led to security vulnerabilities. Since tickets are unauditable by design, security issues may go unnoticed for a long time (e.g., over half a year at AWS [W123]). This unauditable nature of tickets also posed a limitation on our analyses, as we could not analyze proprietary implementations in depth. Despite this limitation, we still uncovered critical vulnerabilities in open-source implementations and large-scale deployments. We cannot estimate how many similar issues remain undetected and argue that tickets are a high-risk feature of TLS.

Our analyses mainly focused on open-source implementations and the HTTPS/1.1 ecosystem. A natural extension would be to broaden this scope: Proprietary implementations could be analyzed through reverse engineering or collaboration with vendors. Other ecosystems beyond HTTPS could be analyzed, for example, FTP. HTTP/2 introduced streams, where multiple requests can be multiplexed over a single connection. This may allow for new virtual host confusion attacks, as the server may not be able to distinguish between different requests and their associated virtual hosts. Additionally, other TLS-related protocols could be analyzed, for example, DTLS and QUIC. Similarly, mechanisms similar to session tickets, such as the DoS amplification protection in DTLS and QUIC, could be analyzed for similar vulnerabilities. To broaden the scope further, TLS-related standards could be systematically analyzed, for example, using natural language processing, to identify further potential pitfalls or inconsistencies. We also propose studies on the usability of cryptographic APIs and collaboration with open-source library maintainers to improve the APIs.

The above-mentioned extensions would not address the root cause of the issues we have found. Session tickets are one of the few features of the TLS handshake where the server can freely design the construction.

Generally, the design of cryptographic protocols is complex and should be reviewed thoroughly. Implementations can then be evaluated against the reviewed design in a black-box manner, as implemented by TLS-Anvil [04]. For tickets, however, the server can freely design the construction without any review. This prevents any automatic evaluation of the construction's security. We conclude that a better design of session tickets is needed, allowing for automatic evaluation of the construction's security and preventing the server from silently undermining the security guarantees of TLS.

Future work should therefore not only focus on broadening the scope of our analyses (as proposed above) but also on designing a new construction for session tickets. Such a construction would be constrained by the current requirements of session resumption using tickets: The goal is to reduce the computational overhead of the handshake, and tickets relieve the server from holding any state. Adding new cryptographic operations to tickets may thwart these goals. We leave it as an open question whether it is possible to design a construction that is both secure and efficient.

Bibliography

Own Publications

[O1] Fabian Bäumer, Marcus Brinkmann, Nurullah Erinola, **Sven Hebrok**, Nico Heitmann, Felix Lange, Marcel Maehren, Robert Merget, Niklas Niere, Maximilian Radoy, Conrad Schmidt, Jörg Schwenk, and Juraj Somorovsky. 2024. TLS-Attacker: a dynamic framework for analyzing TLS implementations. In *Proceedings of cybersecurity artifacts competition and impact award (ACSAC '24)*, 2024. Hawaii. Retrieved from https://www.acsac.org/2024/program/artifacts_competition/comp-acsac24-final11.pdf | *Referenced on pages 11, 26, 100*

[O2] **Sven Hebrok**, Simon Nachtigall, Marcel Maehren, Nurullah Erinola, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. 2023. We Really Need to Talk About Session Tickets: A Large-Scale Analysis of Cryptographic Dangers with TLS Session Tickets. In *32nd USENIX Security Symposium (USENIX Security '23)*, 2023. USENIX Association, Anaheim, CA, 4877–4894. Retrieved from https://www.usenix.org/conference/usenixsecurity23/presentation/hebrok | *Referenced on pages 4, 5, 6, 7, 9, 31, 33, 51, 75, 98, 101, 103*

[O3] **Sven Hebrok**, Tim Leonhard Storm, Felix Matthias Cramer, Maximilian Radoy, and Juraj Somorovsky. 2025. STEK Sharing is Not Caring: Bypassing TLS Authentication in Web Servers using Session Tickets. In *34th USENIX Security Symposium (USENIX Security '25)*, 2025. USENIX Association, Seattle, WA, 8017–8034. Retrieved from https://www.usenix.org/conference/usenixsecurity25/presentation/hebrok | *Referenced on pages 4, 5, 6, 8, 9, 61, 62*

[O4] Marcel Maehren, Philipp Nieting, **Sven Hebrok**, Robert Merget, Juraj Somorovsky, and Jörg Schwenk. 2022. TLS-Anvil: Adapting combinatorial testing for TLS libraries. In *31st USENIX Security Symposium (USENIX Security '22)*, August 2022. USENIX Association, Boston, MA, 215–232. Retrieved from https://www.usenix.

org/conference/usenixsecurity22/presentation/maehren | *Referenced on pages 10, 110*

[O5] Robert Merget, Nurullah Erinola, Marcel Maehren, Lukas Knittel, **Sven Hebrok**, Marcus Brinkmann, Juraj Somorovsky, and Jörg Schwenk. 2026. Opossum Attack: Application Layer Desynchronization using Opportunistic TLS. In *35th USENIX Security Symposium (USENIX Security '26)*, 2026. USENIX Association, Baltimore, MD. Retrieved from https://www.usenix.org/conference/usenixsecurity26/presentation/merget | *Referenced on pages 2, 5, 12*

[O6] Niklas Niere, **Sven Hebrok**, Juraj Somorovsky, and Robert Merget. 2023. Poster: Circumventing the GFW with TLS Record Fragmentation. *CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (November 2023), 3528–3530. https://doi.org/10.1145/3576915.3624372 | *Referenced on page 10*

[O7] Maximilian Radoy, **Sven Hebrok**, and Juraj Somorovsky. 2024. In Search of Partitioning Oracle Attacks Against TLS Session Tickets. In *29th European Symposium on Research in Computer Security*, 2024. Bydgoszcz, Poland, 320–340. https://doi.org/10.1007/978-3-031-70896-1_16 | *Referenced on pages 4, 5, 6, 9, 91, 99, 101*

Academic Publications

[8] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth Schoen, and Brad Warren. 2019. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, November 2019. Association for Computing Machinery, New York, NY, USA, 2473–2487. https://doi.org/10.1145/3319535.3363192 | *Referenced on pages 59, 67*

[9] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. 2022. How to Abuse and Fix

Authenticated Encryption Without Key Commitment. In *31st USENIX Security Symposium (USENIX Security '22)*, 2022. 3291–3308. Retrieved June 14, 2026 from https://www.usenix.org/conference/usenixsecurity22/presentation/albertini | *Referenced on page 102*

[10] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. 2019. The privacy of the TLS 1.3 protocol. July 2019. https://doi.org/10.2478/popets-2019-0065 | *Referenced on page 4*

[11] Marcel Armour and Carlos Cid. 2021. Partition Oracles from Weak Key Forgeries. In *Cryptology and Network Security*, 2021. Springer International Publishing, Cham, 42–62. https://doi.org/10.1007/978-3-030-92548-2_3 | *Referenced on pages 89, 91, 99, 102, 104*

[12] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS using SSLv2. In *25th USENIX Security Symposium (USENIX Security '16)*, August 2016. USENIX Association, Austin, TX, 689–706. Retrieved from https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram | *Referenced on pages 5, 40*

[13] Mihir Bellare and Viet Tung Hoang. 2022. Efficient Schemes for Committing Authenticated Encryption. In *Advances in Cryptology – EUROCRYPT 2022*, 2022. Springer International Publishing, Cham, 845–875. https://doi.org/10.1007/978-3-031-07085-3_29 | *Referenced on page 102*

[14] Mihir Bellare and Chanathip Namprempre. 2008. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *Journal of Cryptology* 21, 4 (October 2008), 469–491. https://doi.org/10.1007/s00145-008-9026-x | *Referenced on page 14*

[15] Mihir Bellare and Björn Tackmann. 2016. The Multi-user Security of Authenticated Encryption: AES-GCM in TLS 1.3. In *Advances in Cryptology – CRYPTO 2016*, 2016. Springer, Berlin, Heidelberg, 247–276. https://doi.org/10.1007/978-3-662-53018-4_10 | *Referenced on pages 34, 57*

[16] Daniel J. Bernstein. 2005. The Poly1305-AES Message-Authentication Code. In *Fast Software Encryption*, 2005. Springer, Berlin, Heidelberg, 32–49. https://doi.org/10.1007/11502760_3 | *Referenced on pages 14, 89*

[17] Daniel J Bernstein. 2008. ChaCha, a variant of Salsa20. Retrieved July 5, 2026 from https://cr.yp.to/chacha/chacha-20080128.pdf | *Referenced on pages 14, 89*

[18] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017. 483–502. https://doi.org/10.1109/SP.2017.26 | *Referenced on page 2*

[19] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella-Béguelin. 2014. Proving the TLS Handshake Secure (As It Is). In *Advances in Cryptology – CRYPTO 2014*, 2014. Springer, Berlin, Heidelberg, 235–255. https://doi.org/10.1007/978-3-662-44381-1_14 | *Referenced on page 2*

[20] Daniel Bleichenbacher. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology — CRYPTO '98*, 1998. Springer, Berlin, Heidelberg, 1–12. https://doi.org/10.1007/BFb0055716 | *Referenced on page 2*

[21] Joachim Breitner and Nadia Heninger. 2019. Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies. In *Financial Cryptography and Data Security*, 2019. Springer International Publishing, Cham, 3–20. https://doi.org/10.1007/978-3-030-32101-7_1 | *Referenced on page 55*

[22] Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and

Sebastian Schinzel. 2021. ALPACA: Application Layer Protocol Confusion - Analyzing and Mitigating Cracks in TLS Authentication. In *30th USENIX Security Symposium (USENIX Security '21)*, 2021. 4293–4310. Retrieved from https://www.usenix.org/conference/usenixsecurity21/presentation/brinkmann | *Referenced on pages 2, 5, 40, 87*

[23] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return of Bleichenbacher's oracle threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security '18)*, August 2018. USENIX Association, Baltimore, MD, 817–849. Retrieved from https://www.usenix.org/conference/usenixsecurity18/presentation/bock | *Referenced on pages 2, 5, 40*

[24] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. 2016. Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016. Retrieved May 31, 2026 from https://www.usenix.org/conference/woot16/workshop-program/presentation/bock | *Referenced on pages 33, 34*

[25] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. 2003. Password Interception in a SSL/TLS Channel. In *Advances in Cryptology - CRYPTO 2003*, 2003. Springer, Berlin, Heidelberg, 583–599. https://doi.org/10.1007/978-3-540-45146-4_34 | *Referenced on page 36*

[26] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. 2016. Host of Troubles: Multiple Host Ambiguities in HTTP Implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, October 2016. ACM, Vienna Austria, 1516–1527. https://doi.org/10.1145/2976749.2978394 | *Referenced on page 87*

[27] Felix Matthias Cramer. 2024. Evaluating the Interaction of TLS Client Certificates and Session Tickets in Virtual Hosting. Bachelor's Thesis. Paderborn University. | *Referenced on page 8*

[28] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A Comprehensive Symbolic Analysis

of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, October 2017. Association for Computing Machinery, New York, NY, USA, 1773–1788. https://doi.org/10.1145/3133956.3134063 | *Referenced on page 2*

[29] 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 (CCS '21)*, November 2021. Association for Computing Machinery, New York, NY, USA, 1981–2003. https://doi.org/10.1145/3460120.3484814 | *Referenced on pages 34, 57*

[30] Antoine Delignat-Lavaud and Karthikeyan Bhargavan. 2015. Network-based Origin Confusion Attacks against HTTPS Virtual Hosting. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*, May 2015. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 227–237. https://doi.org/10.1145/2736277.2741089 | *Referenced on pages 3, 4, 59, 60, 61, 63, 64, 86, 105*

[31] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. 2018. Fast Message Franking: From Invisible Salamanders to Encryptment. In *Advances in Cryptology – CRYPTO 2018*, 2018. Springer International Publishing, Cham, 155–186. https://doi.org/10.1007/978-3-319-96884-1_6 | *Referenced on pages 89, 102*

[32] Kristen Dorey, Nicholas Chang-Fong, and Aleksander Essex. 2017. Indiscreet Logs: Diffie-Hellman Backdoors in TLS. In *Proceedings 2017 Network and Distributed System Security Symposium*, 2017. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2017.23006 | *Referenced on pages 5, 55, 107*

[33] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. 2021. A Cryptographic Analysis of the TLS 1.3 Handshake Protocol. *Journal of Cryptology* 34, 4 (July 2021), 37. https://doi.org/10.1007/s00145-021-09384-1 | *Referenced on page 2*

[34] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, October 2015. ACM, Denver Colorado USA, 542–553. https://doi.org/10.1145/2810103.2813703 | *Referenced on pages 26, 45, 77*

[35] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. 2013. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 conference on Internet measurement conference (IMC '13)*, October 2013. Association for Computing Machinery, New York, NY, USA, 291–304. https://doi.org/10.1145/2504730.2504755 | *Referenced on pages 5, 40*

[36] 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 *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*, November 2014. Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755 | *Referenced on pages 5, 40*

[37] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *22nd USENIX Security Symposium (USENIX Security '13)*, 2013. 605–620. Retrieved from https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric | *Referenced on pages 26, 27, 45, 77, 100*

[38] Nadhem J. Al Fardan and Kenneth G. Paterson. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy*, May 2013. 526–540. https://doi.org/10.1109/SP.2013.42 | *Referenced on pages 35, 36, 57*

[39] Lucas Foppe, Jeremy Martin, Travis Mayberry, Erik C. Rye, and Lamont Brown. 2018. Exploiting TLS Client Authentication for Widespread User Tracking. *Proceedings on Privacy Enhancing*

Technologies 2018, 4 (October 2018), 51–63. https://doi.org/10.1515/popets-2018-0031 | *Referenced on page 87*

[40] Kai Gellert and Tobias Handirk. 2021. A Formal Security Analysis of Session Resumption Across Hostnames. In *Computer Security – ESORICS 2021*, 2021. Springer International Publishing, Cham, 44–64. https://doi.org/10.1007/978-3-030-88418-5_3 | *Referenced on page 5*

[41] Milad Ghaznavi, Elaheh Jalalpour, Mohammad A. Salahuddin, Raouf Boutaba, Daniel Migault, and Stere Preda. 2021. Content Delivery Network Security: A Survey. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2166–2190. https://doi.org/10.1109/COMST.2021.3093492 | *Referenced on page 87*

[42] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. 2016. Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption. In *Advances in Cryptology – EUROCRYPT 2016*, 2016. Springer, Berlin, Heidelberg, 263–293. https://doi.org/10.1007/978-3-662-49890-3_11 | *Referenced on pages 54, 106*

[43] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. 2017. Message Franking via Committing Authenticated Encryption. In *Advances in Cryptology – CRYPTO 2017*, 2017. Springer International Publishing, Cham, 66–97. https://doi.org/10.1007/978-3-319-63697-9_3 | *Referenced on page 102*

[44] Shay Gueron and Yehuda Lindell. 2015. GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle per Byte. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, October 2015. ACM, Denver Colorado USA, 109–119. https://doi.org/10.1145/2810103.2813613 | *Referenced on pages 54, 106*

[45] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2012. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium (USENIX Security '12)*, August 2012. USENIX Association, Bellevue, WA, 205–220. Retrieved from https://www.usenix.org/conference/usenixsecurity

12/technical-sessions/presentation/heninger | *Referenced on pages 5, 40, 55, 56, 107*

[46] Willi Hensch. 2025. Further evaluation of the Interaction between TLS Client Certificates and session Tickets in Virtual Hosting. Master's thesis. Paderborn University. | *Referenced on pages 8, 61*

[47] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. 2015. Robust Authenticated-Encryption AEZ and the Problem That It Solves. In *Advances in Cryptology – EUROCRYPT 2015*, 2015. Springer, Berlin, Heidelberg, 15–44. https://doi.org/10.1007/978-3-662-46800-5_2 | *Referenced on pages 54, 106*

[48] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. 2018. The Multi-user Security of GCM, Revisited: Tight Bounds for Nonce Randomization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, October 2018. Association for Computing Machinery, New York, NY, USA, 1429–1440. https://doi.org/10.1145/3243734.3243816 | *Referenced on pages 34, 57*

[49] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. 2016. TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication. In *Proceedings 2016 Network and Distributed System Security Symposium*, 2016. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2016.23055 | *Referenced on page 5*

[50] Liz Izhikevich, Gautam Akiwate, Briana Berger, Spencer Drakontaidis, Anna Ascheman, Paul Pearce, David Adrian, and Zakir Durumeric. 2022. ZDNS: a fast DNS toolkit for internet measurement. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*, October 2022. Association for Computing Machinery, New York, NY, USA, 33–43. https://doi.org/10.1145/3517745.3561434 | *Referenced on pages 26, 77*

[51] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. 2012. On the Security of TLS-DHE in the Standard Model. In *Advances in Cryptology – CRYPTO 2012*, 2012. Springer, Berlin,

Heidelberg, 273–293. https://doi.org/10.1007/978-3-642-32009-5_17 | *Referenced on page 2*

[52] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks. In *Advances in Cryptology – EUROCRYPT 2018*, 2018. Springer International Publishing, Cham, 456–486. https://doi.org/10.1007/978-3-319-78372-7_15 | *Referenced on page 89*

[53] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. 2018. Coming of Age: A Longitudinal Study of TLS Deployment. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*, October 2018. Association for Computing Machinery, New York, NY, USA, 415–428. https://doi.org/10.1145/3278532.3278568 | *Referenced on pages 5, 14, 40*

[54] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. 2013. On the Security of the TLS Protocol: A Systematic Analysis. In *Advances in Cryptology – CRYPTO 2013*, 2013. Springer, Berlin, Heidelberg, 429–448. https://doi.org/10.1007/978-3-642-40041-4_24 | *Referenced on page 2*

[55] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. 2017. "I have no idea what i'm doing" - on the usability of deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security '17)*, August 2017. USENIX Association, Vancouver, BC, 1339–1356. Retrieved from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/krombholz | *Referenced on pages 67, 84*

[56] Julia Len, Paul Grubbs, and Thomas Ristenpart. 2020. Partitioning Oracle Attacks. Retrieved June 14, 2026 from https://eprint.iacr.org/2020/1491 | *Referenced on page 98*

[57] Julia Len, Paul Grubbs, and Thomas Ristenpart. 2021. Partitioning oracle attacks. In *30th USENIX Security Symposium (USENIX Security '21)*, August 2021. USENIX Association, 195–212. Re-

trieved from https://www.usenix.org/conference/usenixsecurity21/presentation/len | *Referenced on pages 4, 6, 89, 90, 91, 102*

[58] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, Tao Wan, and Jianping Wu. 2014. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *2014 IEEE Symposium on Security and Privacy*, May 2014. 67–82. https://doi.org/10.1109/SP.2014.12 | *Referenced on page 87*

[59] Atul Luykx and Kenneth G. Paterson. 2024. Limits on Authenticated Encryption Use in TLS. Retrieved May 31, 2026 from https://eprint.iacr.org/2024/051 | *Referenced on pages 34, 57*

[60] Wilfried Mayer, Aaron Zauner, Martin Schmiedecker, and Markus Huber. 2016. No Need for Black Chambers: Testing TLS in the E-mail Ecosystem at Large. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, August 2016. IEEE, Salzburg, Austria, 10–20. https://doi.org/10.1109/ARES.2016.11 | *Referenced on page 5*

[61] 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)*, 2021. 213–230. Retrieved from https://www.usenix.org/conference/usenixsecurity21/presentation/merget | *Referenced on pages 2, 5, 40, 42*

[62] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. 2019. Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities. In *28th USENIX Security Symposium (USENIX Security '19)*, 2019. 1029–1046. Retrieved from https://www.usenix.org/conference/usenixsecurity19/presentation/merget | *Referenced on pages 2, 5, 26, 36, 40, 57, 100*

[63] Christopher Meyer, Juraj Somorovsky, Eugen Weiss, Jörg Schwenk, Sebastian Schinzel, and Erik Tews. 2014. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *23rd USENIX Security Symposium*

(USENIX Security '14), 2014. 733–748. Retrieved June 22, 2026 from https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer | *Referenced on page 2*

[64] Seyed Ali Mirheidari, Sajjad Arshad, Kaan Onarlioglu, Bruno Crispo, Engin Kirda, and William Robertson. 2020. Cached and Confused: Web Cache Deception in the Wild. In *29th USENIX Security Symposium (USENIX Security '20)*, 2020. 665–682. Retrieved from https://www.usenix.org/conference/usenixsecurity20/presentation/mirheidari | *Referenced on page 87*

[65] Seyed Ali Mirheidari, Matteo Golinelli, Kaan Onarlioglu, Engin Kirda, and Bruno Crispo. 2022. Web Cache Deception Escalates!. In *31st USENIX Security Symposium (USENIX Security '22)*, 2022. 179–196. Retrieved from https://www.usenix.org/conference/usenixsecurity22/presentation/mirheidari | *Referenced on page 87*

[66] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. 2017. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, October 2017. Association for Computing Machinery, New York, NY, USA, 1631–1648. https://doi.org/10.1145/3133956.3133969 | *Referenced on pages 55, 107*

[67] Arnis Parsovs. 2014. Practical Issues with TLS Client Certificate Authentication. *Proceedings 2014 Network and Distributed System Security Symposium* (2014). https://doi.org/10.14722/ndss.2014.23036 | *Referenced on page 87*

[68] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A research-oriented top sites ranking hardened against manipulation. In *26th annual network and distributed system security symposium san diego, california, USA, february 24-27, 2019 (NDSS 2019)*, 2019. The Internet Society, San Diego, CA, USA. Retrieved from https://www.ndss-symposium.org/ndss-paper/tranco-a-research-oriented-top-sites-ranking-hardened-against-manipulation/ | *Referenced on pages 27, 45, 75, 77, 100*

[69] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. 2019. The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations. In *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019. 435–452. https://doi.org/10.1109/SP.2019.00062 | *Referenced on page 2*

[70] Juraj Somorovsky. 2016. Systematic Fuzzing and Testing of TLS Libraries. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, October 2016. Association for Computing Machinery, New York, NY, USA, 1492–1504. https://doi.org/10.1145/2976749.2978411 | *Referenced on pages 11, 26, 35, 36, 57, 100*

[71] Drew Springall, Zakir Durumeric, and J. Alex Halderman. 2016. Measuring the Security Harm of TLS Crypto Shortcuts. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*, November 2016. Association for Computing Machinery, New York, NY, USA, 33–47. https://doi.org/10.1145/2987443.2987480 | *Referenced on pages 2, 4, 29, 54, 56, 61, 67, 105*

[72] Tim Leonhard Storm. 2023. Large Scale Scanning of TLS Session Ticket Confusion. Master's thesis. Paderborn University. https://doi.org/10.17619/UNIPB/1-1770 | *Referenced on pages 8, 79*

[73] George Arnold Sullivan, Jackson Sippe, Nadia Heninger, and Eric Wustrow. 2022. Open to a fault: On the passive compromise of TLS keys via transient errors. In *31st USENIX Security Symposium (USENIX Security '22)*, 2022. 233–250. Retrieved June 10, 2026 from https://www.usenix.org/conference/usenixsecurity22/presentation/sullivan | *Referenced on page 5*

[74] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users across the Web via TLS Session Resumption. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*, December 2018. Association for Computing Machinery, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708 | *Referenced on pages 4, 56, 105*

[75] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2020. Enhanced performance for the

encrypted web through TLS resumption across hostnames. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES '20)*, August 2020. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3407023.3407067 | *Referenced on pages 5, 85*

[76] Ye Tang, Huiyun Li, and Guoqing Xu. 2015. Cache Side-Channel Attack to Recover Plaintext against Datagram TLS. In *2015 5th International Conference on IT Convergence and Security (ICITCS)*, August 2015. 1–6. https://doi.org/10.1109/ICITCS.2015.7292961 | *Referenced on page 36*

[77] Pontus Tordsson. 2021. *Partitioning oracle attacks against variants of AES-GCM and ChaCha20-Poly1305*. Retrieved June 14, 2026 from https://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-104355 | *Referenced on pages 102, 104*

[78] Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, and Nadia Heninger. 2017. Measuring small subgroup attacks against Diffie-Hellman. In *Proceedings 2017 Network and Distributed System Security Symposium*, 2017. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2017.23171 | *Referenced on pages 5, 40, 55, 107*

[79] Luke Valenta, Nick Sullivan, Antonio Sanso, and Nadia Heninger. 2018. In Search of CurveSwap: Measuring Elliptic Curve Implementations in the Wild. April 2018. IEEE Computer Society, 384–398. https://doi.org/10.1109/EuroSP.2018.00034 | *Referenced on pages 5, 40, 55*

[80] Serge Vaudenay. 2002. Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS... In *Advances in Cryptology — EUROCRYPT 2002*, 2002. Springer, Berlin, Heidelberg, 534–545. https://doi.org/10.1007/3-540-46035-7_35 | *Referenced on pages 2, 14, 32, 35, 41, 89*

[81] Matthias Wachs, Quirin Scheitle, and Georg Carle. 2017. Push away your privacy: Precise user tracking based on TLS client certificate authentication. In *2017 Network Traffic Measurement*

and Analysis Conference (TMA), June 2017. 1–9. https://doi.org/10.23919/TMA.2017.8002897 | *Referenced on page 87*

[82] Mark N. Wegman and J. Lawrence Carter. 1981. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences* 22, 3 (June 1981), 265–279. https://doi.org/10.1016/0022-0000(81)90033-7 | *Referenced on page 14*

[83] Wei Xia, Wei Wang, Xin He, Gang Xiong, Gaopeng Gou, Zhenzhen Li, and Zhen Li. 2021. Old Habits Die Hard: A Sober Look at TLS Client Certificates in the Real World. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, October 2021. 83–90. https://doi.org/10.1109/TrustCom53373.2021.00029 | *Referenced on page 87*

[84] Mingming Zhang, Xiaofeng Zheng, Kaiwen Shen, Ziqiao Kong, Chaoyi Lu, Yu Wang, Haixin Duan, Shuang Hao, Baojun Liu, and Min Yang. 2020. Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*, November 2020. Association for Computing Machinery, New York, NY, USA, 1939–1952. https://doi.org/10.1145/3372297.3417252 | *Referenced on page 87*

[85] Wenyi Morty Zhang, Annie Dai, Keegan Ryan, Dave Levin, Nadia Heninger, and Aaron Schulman. 2025. Don't Look Up: There Are Sensitive Internal Links in the Clear on GEO Satellites. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25)*, November 2025. Association for Computing Machinery, New York, NY, USA, 3960–3974. https://doi.org/10.1145/3719027.3765198 | *Referenced on page 107*

[86] Petr Švenda, Matúš Nemec, Peter Sekan, Rudolf Kvašňovský, David Formánek, David Komárek, and Vashek Matyáš. 2016. The Million-Key Question—Investigating the Origins of RSA Public Keys. In *25th USENIX Security Symposium (USENIX Security '16)*, 2016. 893–910. Retrieved June 10, 2026 from https://www.usenix.org/conference/usenixsecurity

16/technical-sessions/presentation/svenda | *Referenced on pages 56, 107*

Standards and References

[S87] Christopher Allen and Tim Dierks. 1999. *The TLS Protocol Version 1.0*. https://doi.org/10.17487/RFC2246 | *Referenced on pages 16, 19*

[S88] Arm Limited and/or its affiliates. 2026. 10.6 Authenticated encryption with associated data (AEAD). Retrieved May 31, 2026 from https://arm-software.github.io/psa-api/crypto/1.4/api/ops/aead.html | *Referenced on page 38*

[S89] Elaine Barker, Allen Roginsky, and Richard Davis. 2020. *Recommendation for cryptographic key generation*. Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.800-133r2 | *Referenced on page 57*

[S90] Richard Barnes, Martin Thomson, Alfredo Pironti, and Adam Langley. 2015. *Deprecating Secure Sockets Layer Version 3.0*. https://doi.org/10.17487/RFC7568 | *Referenced on page 16*

[S91] Simon Blake-Wilson, Jan Mikkelsen, Magnus Nyström, David Hopwood, and Tim Wright. 2003. *Transport Layer Security (TLS) Extensions*. https://doi.org/10.17487/RFC3546 | *Referenced on page 63*

[S92] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. https://doi.org/10.17487/RFC5280 | *Referenced on page 15*

[S93] Tim Dierks and Eric Rescorla. 2006. *The Transport Layer Security (TLS) Protocol Version 1.1*. https://doi.org/10.17487/RFC4346 | *Referenced on pages 16, 19*

[S94] Morris Dworkin. 2007. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. https://doi.org/10.6028/NIST.SP.800-38D | *Referenced on pages 14, 34, 89*

[S95] D. Eastlake. 2011. Transport Layer Security (TLS) Extensions: Extension Definitions. https://doi.org/10.17487/rfc6066 | *Referenced on page 83*

[S96] Pasi Eronen, Hannes Tschofenig, Hao Zhou, and Joseph A. Salowey. 2008. *Transport Layer Security (TLS) Session Resumption without Server-Side State*. https://doi.org/10.17487/RFC5077 | *Referenced on pages 2, 20, 23, 24, 29, 30, 33, 35, 37, 54, 57, 85, 89, 106*

[S97] Daniel Kahn Gillmor. 2016. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*. https://doi.org/10.17487/RFC7919 | *Referenced on page 17*

[S98] Peter Gutmann. 2014. *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. https://doi.org/10.17487/RFC7366 | *Referenced on pages 14, 36, 57*

[S99] Dan Harkins. 2008. *Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)*. https://doi.org/10.17487/RFC5297 | *Referenced on pages 54, 106*

[S100] Paul E. Hoffman. 2002. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. https://doi.org/10.17487/RFC3207 | *Referenced on page 1*

[S101] Russ Housley. 2009. *Cryptographic Message Syntax (CMS)*. https://doi.org/10.17487/RFC5652 | *Referenced on pages 31, 41*

[S102] ITU-T. X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. Retrieved July 13, 2026 from https://www.itu.int/rec/T-REC-X.509 | *Referenced on page 15*

[S103] ITU-T. X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Retrieved June 14, 2026 from https://www.itu.int/rec/T-REC-X.690 | *Referenced on page 97*

[S104] Jana Iyengar and Martin Thomson. 2021. *QUIC: A UDP-Based Multiplexed and Secure Transport*. https://doi.org/10.17487/RFC9000 | *Referenced on pages 19, 108*

[S105] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. 1997. *HMAC: Keyed-Hashing for Message Authentication*. https://doi.org/10.17487/RFC2104 | *Referenced on page 50*

[S106] Alexey Melnikov and Barry Leiba. 2021. *Internet Message Access Protocol (IMAP) - Version 4rev2*. https://doi.org/10.17487/RFC9051 | *Referenced on page 81*

[S107] Keith Moore and Chris Newman. 2018. *Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access*. https://doi.org/10.17487/RFC8314 | *Referenced on page 1*

[S108] Kathleen Moriarty and Stephen Farrell. 2021. *Deprecating TLS 1.0 and TLS 1.1*. https://doi.org/10.17487/RFC8996 | *Referenced on page 16*

[S109] Chris Newman. 1999. *Using TLS with IMAP, POP3 and ACAP*. https://doi.org/10.17487/RFC2595 | *Referenced on page 1*

[S110] Henrik Nielsen, Jeffrey Mogul, Larry M. Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 1999. *Hypertext Transfer Protocol – HTTP/1.1*. https://doi.org/10.17487/RFC2616 | *Referenced on page 62*

[S111] Yoav Nir and Adam Langley. 2015. *ChaCha20 and Poly1305 for IETF Protocols*. https://doi.org/10.17487/RFC7539 | *Referenced on pages 14, 89*

[S112] NIST. 2023. Digital Signature Standard (DSS). https://doi.org/10.6028/NIST.FIPS.186-5 | *Referenced on page 57*

[S113] NIST. 2023. Advanced Encryption Standard (AES). https://doi.org/10.6028/NIST.FIPS.197-upd1 | *Referenced on pages 43, 140*

[S114] Tim Polk and Sean Turner. 2011. *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. https://doi.org/10.17487/RFC6176 | *Referenced on page 16*

[S115] Eric Rescorla and Tim Dierks. 2008. *The Transport Layer Security (TLS) Protocol Version 1.2*. https://doi.org/10.17487/RFC5246 | *Referenced on pages 2, 15, 16, 19, 24, 31, 41, 89*

[S116] Eric Rescorla, Hannes Tschofenig, and Nagendra Modadugu. 2022. The datagram transport layer security (DTLS) protocol version 1.3. https://doi.org/10.17487/RFC9147 | *Referenced on pages 18, 108*

[S117] Eric Rescorla. 2000. *HTTP Over TLS*. https://doi.org/10.17487/RFC2818 | *Referenced on page 1*

[S118] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3*. https://doi.org/10.17487/RFC8446 | *Referenced on pages 16, 59*

[S119] Eric Rescorla. 2026. *The Transport Layer Security (TLS) Protocol Version 1.3*. https://doi.org/10.17487/RFC9846 | *Referenced on pages 15, 16, 22, 63, 83, 84, 88, 89*

[S120] Marshall T. Rose and John G. Myers. 1996. *Post Office Protocol - Version 3*. https://doi.org/10.17487/RFC1939 | *Referenced on page 81*

[S121] Rob Siemborski and Alexey Melnikov. 2007. *SMTP Service Extension for Authentication*. https://doi.org/10.17487/RFC4954 | *Referenced on page 81*

Websites and Blogs

[W122] A research-oriented top sites ranking hardened against manipulation - Tranco. Retrieved July 19, 2026 from https://tranco-list.eu/ | *Referenced on page 27*

[W123] Amazon. 2021. Resolved: Application Load Balancer Session Ticket Issue. Retrieved June 2, 2026 from https://aws.amazon.com/security/security-bulletins/AWS-2021-002/ | *Referenced on pages 51, 109*

[W124] Vaisha Bernard. 2024. From Simulation to Tenant Takeover. Retrieved April 29, 2025 from https://media.ccc.de/v/38c3-from-simulation-to-tenant-takeover | *Referenced on page 83*

[W125] Andrew Birkett. Win32 Debug CRT Heap Internals. Retrieved June 2, 2026 from https://www.nobugs.org/developer/win32/debug_crt_heap.html | *Referenced on page 43*

[W126] Matt Bullock, Rushil Mehra, and Alessandro Ghedini. 2025. Resolving a Mutual TLS session resumption vulnerability. Retrieved June 17, 2026 from https://blog.cloudflare.com/resolving-a-mutual-tls-session-resumption-vulnerability/ | *Referenced on page 82*

[W127] Hanno Böck. badkeys. Retrieved July 18, 2026 from https://badkeys.info/ | *Referenced on page 55*

[W128] cleca. 2014. WARNING: Enigmail 1.7 *completely* *broken*. Retrieved May 31, 2026 from https://sourceforge.net/p/enigmail/forum/support/thread/3e7268a4/ | *Referenced on page 33*

[W129] Cloudflare. Adoption & Usage Worldwide | Cloudflare Radar. Retrieved May 19, 2026 from https://radar.cloudflare.com/adoption-and-usage | *Referenced on page 1*

[W130] Sean Gallagher. 2014. Cloudflare gives Internet a present: free, no-hassle “Universal” SSL. Retrieved May 19, 2025 from https://arstechnica.com/information-technology/2014/09/cloudflare-gives-internet-a-present-free-no-hassle-universal-ssl/ | *Referenced on page 59*

[W131] Google. Email encryption in transit – Google Transparency Report. Retrieved May 19, 2026 from https://transparencyreport.google.com/safer-email/overview | *Referenced on page 1*

[W132] Google. HTTPS encryption on the web – Google Transparency Report. Retrieved May 19, 2026 from https://transparencyreport.google.com/https/overview | *Referenced on pages 1, 59*

[W133] greenluigi1. 2022. How I Hacked my Car. Retrieved June 2, 2026 from https://programmingwithstyle.com/posts/howihackedmycar/ | *Referenced on pages 43, 140*

[W134] IANA. Service Name and Transport Protocol Port Number Registry. Retrieved May 19, 2026 from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml | *Referenced on pages 1, 139*

[W135] James Kettle. 2018. Practical Web Cache Poisoning. Retrieved January 13, 2025 from https://portswigger.net/research/practical-web-cache-poisoning | *Referenced on page 87*

[W136] Fiona Klute. 2020. CVE-2020-13777: TLS 1.3 session resumption works without master key, allowing MITM (#1011) · Issues · gnutls / GnuTLS · GitLab. Retrieved September 5, 2025 from https://gitlab.com/gnutls/gnutls/-/issues/1011 | *Referenced on pages 3, 30, 33*

[W137] Mozilla. ssl.resumed_session | GLAM. Retrieved May 19, 2026 from https://glam.telemetry.mozilla.org/fog/probe/ssl_resumed_session/explore?activeBuckets=%5B%22false%22%2C%22true%22%5D&app_id=release | *Referenced on page 2*

[W138] NIST. 2020. CVE-2020-13777. Retrieved September 5, 2025 from https://nvd.nist.gov/vuln/detail/CVE-2020-13777 | *Referenced on pages 3, 30, 33*

[W139] NIST. 2025. CVE-2025-23048. Retrieved July 20, 2026 from https://nvd.nist.gov/vuln/detail/CVE-2025-23048 | *Referenced on page 72*

[W140] NIST. 2025. CVE-2025-23419. Retrieved July 20, 2026 from https://nvd.nist.gov/vuln/detail/CVE-2025-23419 | *Referenced on page 72*

[W141] NIST. 2025. CVE-2025-66614. Retrieved July 20, 2026 from https://nvd.nist.gov/vuln/detail/CVE-2025-66614 | *Referenced on page 72*

[W142] Qualys SSL Labs. SSL Pulse. Retrieved May 29, 2026 from https://www.ssllabs.com/ssl-pulse/ | *Referenced on page 16*

[W143] SEC Consult. 2017. Fake Crypto: Microsoft Outlook S/MIME Cleartext Disclosure (CVE-2017-11776). Retrieved May 31, 2026 from https://sec-consult.com/blog/detail/fake-crypto-microsoft-outlook-smime-cleartext-disclosure-cve-2017-11776/ | *Referenced on page 33*

[W144] Nick Sullivan. 2017. Introducing Zero Round Trip Time Resumption (0-RTT). Retrieved May 19, 2026 from https://blog.cloudflare.com/introducing-0-rtt/ | *Referenced on page 2*

[W145] Tim Taubert. 2014. Botching Forward Secrecy - The sad state of server-side TLS Session Resumption implementations. Retrieved August 5, 2024 from https://timtaubert.de/blog/2014/11/the-sad-state-of-server-side-tls-session-resumption-implementations/ | *Referenced on pages 2, 4, 105*

[W146] Filippo Valsorda. 2017. Ticketbleed (CVE-2016-9244). Retrieved August 5, 2024 from https://filippo.io/Ticketbleed/ | *Referenced on page 5*

[W147] Filippo Valsorda. 2017. We need to talk about Session Tickets. Retrieved July 3, 2024 from https://words.filippo.io/we-need-to-talk-about-session-tickets/ | *Referenced on pages 2, 4, 29, 56, 105*

[W148] W3Techs. Usage Statistics and Market Share of Web Servers, June 2026. Retrieved June 16, 2026 from https://w3techs.com/technologies/overview/web_server | *Referenced on page 67*

[W149] Wikipedia Contributors. 2022. Magic number (programming). Retrieved June 2, 2026 from https://en.wikipedia.org/w/index.php?title=Magic_number_(programming)&oldid=1084703437#Debug_values | *Referenced on pages 43, 140*

[W150] David Ziemann. 2020. Analysis of the GnuTLS Session Ticket Bug (CVE-2020-13777). Retrieved June 30, 2026 from https://web.archive.org/web/20260214142624/https://hackmanit.de/de/blog/118-analysis-of-the-gnutls-session-ticket-bug-cve-2020-13777/ | *Referenced on pages 3, 30, 33*

Code, Software, and Documentation

[C151] LibreSSL. Retrieved July 20, 2026 from https://www.libressl.org/ | *Referenced on pages 1, 142*

[C152] nginx. Retrieved July 20, 2026 from https://nginx.org/ | *Referenced on page 142*

[C153] [security] Go 1.25.7 and Go 1.24.13 are released. Retrieved July 20, 2026 from https://groups.google.com/g/golang-announce/c/K09ubi9FQFk?pli=1 | *Referenced on page 72*

[C154] Shadowsocks. Retrieved June 10, 2026 from https://shadowsocks.org/ | *Referenced on page 89*

[C155] The ZMap Project. Retrieved June 16, 2026 from https://zmap.io/ | *Referenced on pages 26, 77*

[C156] Michael Alegre. lswsdocs_config_vhost_ssl – Client Verification. Retrieved June 16, 2026 from https://www.litespeedtech.com/docs/webserver/config/virtual-host-ssl#clientVerify | *Referenced on page 67*

[C157] Amazon. Create an HTTPS listener for your Application Load Balancer - Elastic Load Balancing. Retrieved June 22, 2026 from https://docs.aws.amazon.com/elasticloadbalancing/latest/application/create-https-listener.html | *Referenced on page 1*

[C158] Amazon. Mutual authentication with TLS in Application Load Balancer - Elastic Load Balancing. Retrieved May 8, 2025 from https://docs.aws.amazon.com/elasticloadbalancing/latest/application/mutual-authentication.html | *Referenced on page 82*

[C159] The Apache Software Foundation. Apache Tomcat® - Welcome!. Retrieved July 20, 2026 from https://tomcat.apache.org/ | *Referenced on page 142*

[C160] The Apache Software Foundation. Welcome! - The Apache HTTP Server Project. Retrieved July 20, 2026 from https://httpd.apache.org/ | *Referenced on page 142*

[C161] aws. aws/aws-lc. Retrieved June 22, 2026 from https://github.com/aws/aws-lc | *Referenced on pages 1, 142*

[C162] aws. aws/s2n-tls. Retrieved June 22, 2026 from https://github.com/aws/s2n-tls | *Referenced on pages 1, 142*

[C163] Joe Birr-Pixton. 2024. Partitioning Oracle Attacks against TLS Session Tickets · Issue #2023 · rustls/rustls. Retrieved July 20, 2026 from https://github.com/rustls/rustls/issues/2023 | *Referenced on page 96*

[C164] Caddy Web Server. Global options (Caddyfile) – strict_sni_host. Retrieved May 8, 2025 from https://caddyserver.com/docs/caddyfile/options#strict-sni-host | *Referenced on page 72*

[C165] Cloudflare. 2026. Cloudflare for SaaS. Retrieved June 17, 2026 from https://developers.cloudflare.com/cloudflare-for-platforms/cloudflare-for-saas/ | *Referenced on page 79*

[C166] Cloudflare. 2026. Enable mTLS. Retrieved June 17, 2026 from https://developers.cloudflare.com/ssl/client-certificates/enable-mtls/ | *Referenced on page 81*

[C167] Cloudflare. 2026. Browser compatibility - Non-SNI support. Retrieved June 24, 2026 from https://developers.cloudflare.com/ssl/reference/browser-compatibility/#non-sni-support | *Referenced on page 63*

[C168] Cloudflare. 2026. Cloudflare SSL/TLS. Retrieved June 22, 2026 from https://developers.cloudflare.com/ssl/ | *Referenced on page 1*

[C169] Fastly. Fastly TLS prerequisites and limitations | Fastly Documentation. Retrieved June 22, 2026 from https://www.fastly.com/documentation/guides/getting-started/domains/securing-domains/tls-prerequisites-and-limitations/ | *Referenced on page 1*

[C170] Fastly. Setting up Mutual TLS authentication. Retrieved May 8, 2025 from https://docs.fastly.com/en/guides/setting-up-mutual-tls-authentication | *Referenced on page 82*

[C171] Fastly. TLS service options – Dedicated IP addresses. Retrieved June 17, 2026 from https://docs.fastly.com/products/tls-service-options#dedicated-ip-addresses | *Referenced on page 79*

[C172] golang. tls package - crypto/tls - Go Packages. Retrieved June 22, 2026 from https://pkg.go.dev/crypto/tls | *Referenced on pages 1, 142*

[C173] Google Cloud. 2026. SSL certificates overview | Cloud Load Balancing | Google Cloud Documentation. Retrieved June 22, 2026 from https://docs.cloud.google.com/load-balancing/docs/ssl-certificates | *Referenced on page 1*

[C174] Google Cloud. 2026. Mutual TLS overview | Cloud Load Balancing | Google Cloud Documentation. Retrieved July 17, 2026 from https://docs.cloud.google.com/load-balancing/docs/mtls | *Referenced on page 81*

[C175] Google. boringssl. Retrieved June 22, 2026 from https://boringssl.googlesource.com/boringssl/ | *Referenced on pages 1, 142*

[C176] Jack Lloyd. Botan. Retrieved June 22, 2026 from https://botan.randombit.net/ | *Referenced on pages 1, 142*

[C177] Ben Laurie. 1999. Fix security hole. · openssl/openssl@b4cadc6. Retrieved September 29, 2025 from https://github.com/openssl/openssl/commit/b4cadc6e1343c01b06613053a90ed2ee85e65090 | *Referenced on page 4*

[C178] Linaro Limited. Mbed TLS, TF-PSA-Crypto. Retrieved June 22, 2026 from https://www.trustedfirmware.org/projects/mbed-tls/ | *Referenced on pages 1, 142*

[C179] LiteSpeed Technologies Inc. OpenLiteSpeed. Retrieved July 20, 2026 from https://openlitespeed.org/ | *Referenced on page 142*

[C180] Microsoft. 2026. Tutorial: Configure an Application Gateway with TLS termination using the Azure portal. Retrieved June 17, 2026 from https://learn.microsoft.com/en-us/azure/application-gateway/create-ssl-portal | *Referenced on pages 1, 81*

[C181] Microsoft. IIS Client Certificate Mapping Authentication <iisClientCertificateMappingAuthentication>. Retrieved July 1, 2026 from https://learn.microsoft.com/en-us/iis/configuration/system.webserver/security/authentication/iisclientcertificatemappingauthentication/ | *Referenced on pages 67, 73*

[C182] Mozilla. Network Security Services (NSS) — Firefox Source Docs documentation. Retrieved June 22, 2026 from https://firefox-source-docs.mozilla.org/security/nss/index.html | *Referenced on pages 1, 142*

[C183] OpenSSL. OpenSSL 1.1.1 Series Release Notes | OpenSSL Library. Retrieved June 29, 2026 from https://openssl-library.org/news/openssl-1.1.1-notes/index.html | *Referenced on page 86*

[C184] OpenSSL. openssl/CHANGES at OpenSSL_1_1_1 · openssl/openssl. Retrieved June 29, 2026 from https://github.com/

openssl/openssl/blob/OpenSSL_1_1_1/CHANGES | *Referenced on page 86*

[C185] OpenSSL. OpenSSL Library. Retrieved June 17, 2026 from https://openssl-library.org/ | *Referenced on pages 1, 85, 142*

[C186] OpenSSL. SSL_CTX_set_client_hello_cb - OpenSSL Documentation. Retrieved May 8, 2025 from https://docs.openssl.org/3.5/man3/SSL_CTX_set_client_hello_cb/ | *Referenced on page 86*

[C187] OpenSSL. SSL_CTX_set_session_id_context - OpenSSL Documentation. Retrieved May 8, 2025 from https://docs.openssl.org/3.5/man3/SSL_CTX_set_session_id_context/ | *Referenced on page 85*

[C188] OpenSSL. SSL_CTX_set_tlsext_servername_callback - OpenSSL Documentation. Retrieved May 8, 2025 from https://docs.openssl.org/3.5/man3/SSL_CTX_set_tlsext_servername_callback/ | *Referenced on page 86*

[C189] OpenSSL. SSL_CTX_set_verify - OpenSSL Documentation. Retrieved May 8, 2025 from https://docs.openssl.org/3.5/man3/SSL_CTX_set_verify/ | *Referenced on page 86*

[C190] Oracle. Java Secure Socket Extension (JSSE) Reference Guide. Retrieved June 22, 2026 from https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html | *Referenced on pages 1, 142*

[C191] Sean Parkinson. 2024. SSL default ticket encryption callback: check in len on decrypt by SparkiDev · pull request #7683 · wolfSSL/wolfssl. Retrieved July 20, 2026 from https://github.com/wolfSSL/wolfssl/pull/7683 | *Referenced on page 97*

[C192] Sean Parkinson. 2024. Default session ticket enc/dec: allow AES-CBC with HMAC by SparkiDev · Pull Request #7703 · wolfSSL/wolfssl. Retrieved July 20, 2026 from https://github.com/wolfSSL/wolfssl/pull/7703 | *Referenced on page 97*

[C193] Rambus. 2023. TLS Toolkit - Security IP. Retrieved July 20, 2026 from https://web.archive.org/web/20230208040159/https://www.rambus.com/security/software-protocols/tls-toolkit/ | *Referenced on pages 1, 142*

[C194] rustls. rustls/rustls. Retrieved June 22, 2026 from https://github.com/rustls/rustls | *Referenced on pages 1, 142*

[C195] Tim Rühsen, Daiki Ueno, Dmitry Baryshkov, and Zoltán Fridrich. GnuTLS. Retrieved July 20, 2026 from https://www.gnutls.org/ | *Referenced on pages 1, 142*

[C196] tls-attacker. TLS-Attacker. Retrieved May 29, 2026 from https://github.com/tls-attacker/TLS-Attacker | *Referenced on pages 26, 100*

[C197] tls-attacker. TLS-Crawler. Retrieved August 24, 2026 from https://github.com/tls-attacker/TLS-Crawler | *Referenced on page 26*

[C198] tls-attacker. TLS-Scanner. Retrieved May 29, 2026 from https://github.com/tls-attacker/TLS-Scanner | *Referenced on pages 26, 100*

[C199] Filippo Valsorda. 2023. crypto/tls: reduce session ticket linkability · golang/go@6b020be. Retrieved June 2, 2026 from https://github.com/golang/go/commit/6b020be648960440b1f02d28889a6e050f59c31c | *Referenced on page 39*

[C200] wolfSSL Inc. wolfSSL Embedded SSL/TLS Library - wolfSSL. Retrieved June 22, 2026 from https://www.wolfssl.com/ | *Referenced on pages 1, 142*

[C201] ZeroSSL. Caddy - The Ultimate Server with Automatic HTTPS. Retrieved January 6, 2025 from https://caddyserver.com/ | *Referenced on pages 67, 142*

[C202] zmap. ZDNS: A Fast DNS Toolkit for Internet Measurement. Retrieved June 16, 2026 from https://github.com/zmap/zdns | *Referenced on pages 26, 77*

[C203] zmap. ZGrab 2.0. Retrieved from https://github.com/zmap/zgrab2 | *Referenced on pages 26, 45, 77*

[C204] zmap. ZMap. Retrieved June 14, 2026 from https://github.com/zmap/zmap | *Referenced on pages 26, 45, 77, 100*

A Appendix

A.1 IANA TLS Services

To determine the number of protocols using TLS, we used the IANA port number registry [W134]. The registry lists well-known services and their associated ports. For each service, a name, a port, and a description can be provided.

We iterated over this list and automatically marked services that appear to be using TLS using the following rules:

suffix The name of the service ends with `-ssl`, `-tls`, or `-https`.

prefix The name of the service starts with `ssl-`, `tls-`, or `https-`.

description The service description contains `TLS`, `SSL`, `HTTPS`, or `secure socket`.

We found 152 services that may use TLS. We manually went over the list and determined that three protocols were false positives (`non-TLS` in the description, matching the description rule). For seven protocols, we were unsure whether they use TLS or are just TLS-related (the descriptions were not conclusive enough). We assume that the remaining 142 services use TLS. The concrete numbers of our classification are shown in Table A1.

Table A1: Number of services in the IANA port registry matching our outlined rules. We also list how we classified the services, whether they actually use TLS.

	Likely TLS	Unsure	False Positive
Description	71	6	3
Description + Prefix	1	0	0
Description + Suffix	56	1	0
Suffix Only	14	0	0
∑	142	7	3

As the descriptions in the list are not very detailed, and only a few protocols reference a standard, we cannot be sure to what extent these services use TLS. Hence, we are conservative and, in the introduction, only vaguely state that “over 100 well-known services […] use TLS” rather than stating a concrete number.

A.2 Parameters for Ticket Decryption

When trying to decrypt a ticket in our large-scale scan in Chapter 3, we tried using the following parameters for the decryption of the ticket.

Algorithms. We tested the following algorithms for the decryption of the ticket:

DES in ECB and CBC mode.

3DES in ECB and CBC mode.

AES with 128-bit and 256-bit keys in ECB, CBC, CTR, CCM, and GCM modes. Note that for CCM and GCM we ignored the authentication tag, as we only wanted to test the decryption of the ticket.

ChaCha20 on its own.

When testing for weak authentication keys, we always assumed that an HMAC is used. For the underlying hash algorithms, we tested MD5, SHA1, SHA256, SHA384, and SHA512.

Keys. We tested keys based on code examples, known debug values, and educated guesses.

48 × Constant Bytes We consider keys consisting of the same byte repeated (e.g., `0x00000000`). For these keys, we tested multiples of 16 ± 1 (i.e., 0, 1, 15, 16, 17, 31, 32, …).

48 × Incrementing Bytes We also chose keys consisting of consecutively increasing byte values (e.g., `0x00010203`). For the first starting byte we again tested multiples of 16 ± 1 (i.e., 0, 1, 15, 16, 17, 31, 32, …).

3 × Special Values We further include special keys of incrementing values. Two keys with a step size of 16, starting at 0 and 1, respectively (i.e., `0x00102030`... and `0x01112131`...). If the desired key length is 16 bytes, we included the key `0x00112233445566778899aabbccddeeff`.

41 × Debug values We used 41 known debug values [W149], some of which are used to mark uninitialized memory. We repeated these values to fill the desired key length.

4 × NIST Keys Inspired by [W133], we also included keys that are used as example keys in the NIST specification of AES [S113]. However, we only use them if the desired key length matches.

This results in up to 144 unique keys due to some overlap, and some keys only apply to certain key lengths.

A.3 Key Sets for Partitioning Oracles

For the large-scale evaluation, we used two key sets to generate the splitting ciphertexts. The first key set is the same as used in our previous study, listed in Appendix A.2. The second key set is an extension of the first and includes additional constant keys, counting keys, and keys with all bytes zero except the first byte. Concretely, we include the following additional keys:

208 × Constant Bytes We include all keys consisting of the same byte repeated (e.g., `0x00000000`).

208 × Incrementing Bytes We include all keys consisting of consecutively increasing byte values (e.g., `0x00010203`).

255 × First Byte We include all keys with all bytes zero except the first byte (e.g., `0x00000000`, `0x01000000`, …, `0xff000000`).

Depending on the key length, this results in up to 815 unique keys in the second key set.

A.4 Library Versions and Sources

Throughout our studies, we analyzed various TLS libraries and HTTPS servers. In Chapter 3 and Chapter 5, we used the source code of these libraries to analyze their implementations and evaluate their security. In Chapter 4, we analyzed the implementations in a black-box manner and used readily available Docker images, which we list in Table A2. For the source-code guided analyses, we used the source code from the sources listed in Table A3.

Table A2: Software versions used for the evaluation of session ticket confusion.

Software	Docker Image	Version
Apache httpd	httpd	2.4.62
Apache Tomcat	tomcat	11.0.12
Caddy	caddy	2.8.4
Microsoft IIS	mcr.microsoft.com/windows/servercore/iis	windowsservercore-ltsc2025 (2026-07-01)
OpenLiteSpeed	litespeedtech/openlitespeed	1.8.2-lsphp82
LiteSpeed	litespeedtech/litespeed	6.3.1-lsphp81
nginx	nginx	1.27.2

Table A3: Concrete versions of the analyzed source code.

	Library	Source	Commit / Tag
[C161]	AWS-LC	https://github.com/aws/aws-lc	`d50ded59` / `v1.72.0`
[C175]	BoringSSL	https://boringssl.googlesource.com/boringssl	`8415495e`
[C176]	Botan	https://github.com/randombit/botan	`c6a09af6` / `3.1.0` `86d10552` / `3.11.1`
[C195]	GnuTLS	https://gitlab.com/gnutls/gnutls	`34fb40fa` / `3.8.12`
[C172]	GoTLS	https://github.com/golang/go	`9c8bf0e7` / `go1.26.2`
[C190]	JSSE (OpenJDK)	https://github.com/openjdk/jdk26u	`fcc00fc4` / `jdk-26.0.1+8`
[C151]	LibreSSL	https://github.com/libressl/portable	*Release v4.3.1*
[C193]	MatrixSSL[37]	https://github.com/amitv87/matrixssl	`effeb142` / `4-6-0-open` `7ab63483` / `4-7-0-open`
[C178]	mbedTLS	https://github.com/Mbed-TLS/mbedtls	`1873d3bf` / `v3.4.0` `0fe989b6` / `v4.1.0`
[C182]	NSS	https://hg.mozilla.org/projects/nss	`cf204f07` / `NSS_3_123_1_RTM`
[C185]	OpenSSL	https://github.com/openssl/openssl	`11b7b6ea` / `openssl-4.0.0`
[C194]	Rustls	https://github.com/rustls/rustls	`975e1ca5` / `v/0.21.1` `b44c09fb` / `v/0.23.40`
[C162]	s2n	https://github.com/aws/s2n-tls	`e954ee5d` / `v1.3.46` `a71ea1f9` / `v1.7.2`
[C200]	WolfSSL	https://github.com/wolfSSL/wolfssl	`97970738` / `v5.6.0-stable` `1d363f3a` / `v5.9.1-stable`
[C160]	Apache httpd	https://github.com/apache/httpd	`1284d059` / `2.4.67`
[C159]	Apache Tomcat	https://github.com/apache/tomcat	`ccdf7a8c` / `11.0.22`
[C201]	Caddy	https://github.com/caddyserver/caddy	`cc58caa1` / `v2.11.3`
[C152]	nginx	https://github.com/nginx/nginx	`d4420528` / `release-1.31.1`
[C179]	OpenLiteSpeed	https://github.com/litespeedtech/openlitespeed	`28f7da33` / `v1.9.0.1`

37. The official MatrixSSL repository was taken down in July 2023. We used https://github.com/amitv87/matrixssl (accessed: 2026-07-19) and validated the commit hash against https://web.archive.org/web/20230126084700/https://github.com/matrixssl/matrixssl/ (accessed: 2026-07-19).