Securing SUIT-conform Firmware Update Management in the IoT with DNSSEC/DANE

Max Schrötter, Wolf-Jörgen Stange, Bettina Schnor
University of Potsdam, Germany
{schroetter,schnor}@cs.uni-potsdam.de

I. Motivation

As IoT devices become more prevalent, fixing security vulnerabilities in a timely manner is becoming crucial. The Federal Office for Information Security has published security recommendations for IoT devices (Appendix "SYS.4.4: Allgemeines IoT-Gerät" in [1]). This includes:

SYS.4.4.A1 The devices SHALL have update functions. The manufacturer SHALL offer an update process.

SYS.4.4.A3 If security vulnerabilities are identified, they SHALL be fixed as soon as possible. ... In general, care MUST be taken to obtain patches and updates from trusted sources only.

Typically, if vendors offer an over-the-air (OTA) update procedure, the new firmware is transferred over an encrypted connection using for example TLS or DTLS[2]. This requires a trustworthy Public Key Infrastructure (PKI) with pre-installed public vendor keys or certificates on the devices. Paracha et al. investigated the TLS behavior from Consumer IoT devices [3]. They report that deprecated and distrusted CA certificates are rarely removed from device root stores. But several incidents have proven in the past that the trustworthiness of PKIs is not always given, either due to governmental motives [4] or due to the disclosure of private keys [5, 6].

In addition, in case a vendor key is compromised, safe key-rollover procedures are missing. With a compromised vendor key an attacker is able to sign firmware updates and install malicious updates through a man in the middle attack. Looking at the situation for secure web communication, there exist several approaches to heal the situation like Certification Authority Authorization (CAA, 8654) records, certification revocation lists and the Online Certificate Status protocol (OCSP, RFC 8954). But none has shown to be effective. Google, for example decided in 2012 to default Chrome not to check for certificate revocation on non-extended certificates [7].

Several research groups have undertaken efforts to adapt update processes for the IoT. For example, the UpKit [8] implementation is based on CoAP, and MUP [9] adapts the UpKit protocol for MQTT-based IoT environments.

Recently, the IETF adopted two RFCs regarding updates for constrained IoT devices [10, 11]. RFC 9019 proposes an architecture for Software Updates for Internet of Things (SUIT) [10] and is called SUIT-architecture in the following. SUIT assumes asymmetric cryptography and a public key infrastructure. A data structure called manifest specifies 24 elements with detailed information about the firmware (see section 3 in RFC 9124 [11]).

All these approaches have in common that they follow a two-step approach where signed metadata (the so-called manifest) describing the new update is downloaded and checked first before the update file is downloaded and installed. In case of SUIT, this is at least an option: "Firmware images and manifests may be conveyed as a bundle or detached." The two-step approach is already used in The Update Framework (TUF) [12] which was the base of the update process for the Tor project. In case of the IoT, this approach is preferable, since it allows to reject invalid software at an early stage preventing unnecessary firmware transfers which saves energy [9].

The first step in acquiring the update manifest is to contact the vendor server. Therefore, a DNS query is issued. This query can be used to get more security relevant information about the communication partner. Instead of relying on a PKI alone, we propose to re-use security features of the Domain Name Service Security extensions (DNSSEC) (RFC 4033). While approaches like DoT (RFC 7858), DoH (RFC 8484) or DNS over CoAP [13] aim to secure the DNS communication itself, mainly for privacy issues, they all stick with a not well maintained PKI infrastructure (see for example [3]). On the other side, DNSSEC and DANE (RFC 7671) have proven to be successful in securing mail server communication. [14] shows that 82% of SMTP servers manage DANE correctly. Furthermore, they show that 99% of domains that outsource their SMTP servers manage DANE correctly.

Since DNS allows to store hash values and even keys within DNS records (see DNSSEC/DANE resp. IPSECKEY records), we propose to secure the firmware update process by DNSSEC/DANE in the following cases:

1) Vendor - Update Server communication with DANE/TLS: Instead of trusting the PKI, the public key within the certificate is validated by DANE.
2) Validating the firmware manifest with IPSECKEY records,
3) Validating the key rollover manifest with IPSECKEY records,

The next section describes the necessary extensions to secure an SUIT-conform update protocol. Our prototype implementation for MUP is described in Section III.

II. Securing Updates with DNSSEC/DANE

For many IoT devices the use of DNSSEC is beyond their compute capability or not possible as they are deployed in

a constrained network. Using IoT devices with no direct connection to the Internet is typical for industrial IoT where edge computing architectures are common. Even for consumer IoT devices the use of Thread [15] and Zigbee networks, which are based on IEEE 802.15.4 becomes increasingly common. These IoT devices communicate via a border router or an edge device with Internet services.

In the case of an update these edge devices must store and forward the update to the device. The SUIT-architecture sees these edge devices as part of the firmware server. The MYNO update protocol (MUP) [9] is an example that uses an edge node to distribute software updates. MUP specifies how firmware updates can be deployed and verified. These steps conform to the RFC 9019 requirements. For the rest of this paper the terminology and architecture of MUP will be used.

A. DANE

If an update becomes available or an IoT device requests an update, the edge device (Update Server) contacts the Vendor Server which starts with querying the DNS server as seen in Figure 1. The communication between the Update Server and the Vendor Server is TLS encrypted. We propose to verify the certificate of the vendor via DANE. Therefore, the according TLSA record which stores, for example the hash value of the vendor's public key, is also queried from the DNS server (Step 2 in Figure 1). This approach simplifies key revocation. In case of a private key disclosure, an update of the TLSA record would stop clients trusting an attacker using the compromised certificate.

B. Manifest Validation

In Step 4 of Figure 1 the manifest is downloaded. This manifest includes information to verify the integrity of the manifest (via signature) and the update (via digest). In MUP and SUIT the integrity of the manifest is verified by the IoT device with previously installed trust anchors. Since these trust anchors are usually installed with the last update IoT devices have no way to verify that the signing certificate has not been revoked in the meantime. Our proposal is that the Update Server checks that the signature was created by a still valid key. Therefore, the vendor is required to publish the current public key via DNS. We define the following DNS-Scheme: Key material will be stored under a subdomain like `keystore.vendordomain.tld`. This subdomain has a record for each public key used by the vendor. The used record names are key IDs uniquely identifying each key. Furthermore, a CNAME with a predefined name, here `main` must exist, pointing to the current valid public key. Of course, every other naming scheme on which Vendor and Update Server have agreed is possible.

Since the Update Server also needs to know the signature algorithm, the manifest is extended by a `KeyInfo` structure containing the following fields:

- `KeyID`: to identify the public key and to derive the FQDN for retrieving the public key
- `Algorithm`: the algorithm used to create signature
- `KeyType`: the type of public key

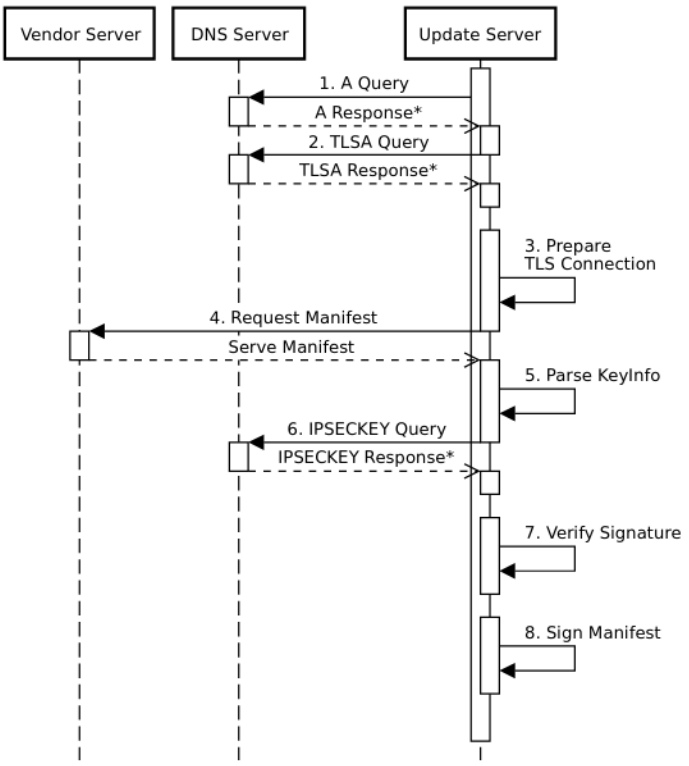

Figure 1. Flow between the Vendor Firmware Server and the Update Server (edge device)

- `KeyCurve`: the elliptic curve used

The values used for those fields correspond to the IANA COSE registries [16].

The Update Server uses the information from the `KeyInfo` structure to verify that the key specified by the manifest is the same key as the `main` CNAME refers to. For this, the Update Server retrieves the corresponding DNS record (see Step 6 in Figure 1). In our current implementation, we use IPSECKEY records to store the public keys. Alternatively text records or TLSA could be used to store public keys. However, IPSECKEY records have the advantage of using base64 encoding which reduces the key size by 66% compared to the hexadecimal representation used by TLSA. When the Update Server successfully validated the signature of the manifest (see Step 7 in Figure 1), the update protocol is resumed. Otherwise, the Update Process fails in this early stage without any energy consuming on the IoT device. In Step 8 the manifest is extended with a signature of the update server as described in MUP[9]. The IoT device will only accept updates that contain valid signatures from the vendor and the update server. The IoT device verifies these signatures with information of a local trust anchor.

C. Key Rollover

Here we show how to secure a key rollover with the above defined DNS scheme. For this process, we propose the Vendor Key Rollover Manifest shown in Table I. Figure 2 describes

Table I
VENDOR KEY ROLLOVER MANIFEST

Vendor Key Rollover Manifest	
field	**description**
App ID	unique identifier for the app for which the public key is updated
version	the version of the app the key rollover is applied
new `KeyInfo`	`KeyInfo` structure for the new public key
public key	the public key to be added to the trust anchor
old `KeyInfo`	`KeyInfo` structure for the old public key
vendor signature	signature of the vendor with the old key pair over the fields above
Manifest Extension	
IoT device nonce	a nonce generated by the IoT device to protect against replay attacks
update server signature	signature of the update server over all fields above

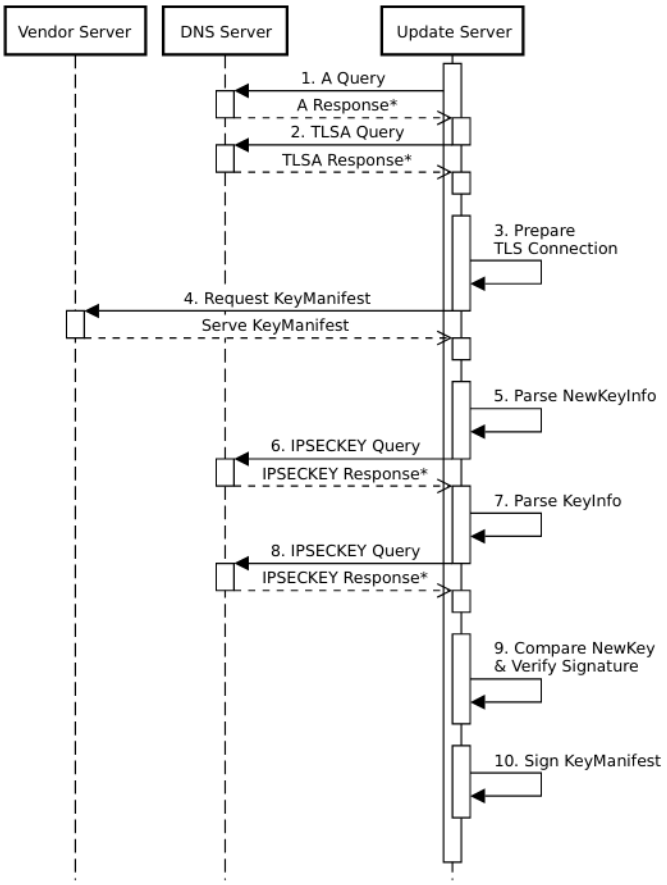

Figure 2. Flow between Vendor and Update Server for a Key Rollover

the communication between the Update Server and the Vendor Server in case of a vendor key rollover.

When a Key Rollover Manifest is published, the Update Server initiates a TLS encrypted connection, which is also secured via DANE as previously described in Step 1-3 in Figure 1. The Update Server downloads the Vendor Key Rollover Manifest (Step 4 in Figure 2). Next the Update Server parses the `KeyInfo` structs of the new and old public key (Step 5&7). It retrieves both public keys from the DNS-Server (Step 6&8). In Step 9 it verifies that the proposed new key is equal to the one, that the `main` CNAME is pointing to. The signature of the manifest is also verified in this step using the old public key. Compared to Figure 1 there is only one additional communication with the DNS server. This is necessary to retrieve the DNS record for the new key. Finally, the Update Server extends the manifiest by a nonce which was generated by the IoT device and signs all fields (see the Manifest Extension in Table I). The nonce is created by the IoT device on request by the Update Server to prevent replay attacks. The IoT device then receives the completed manifest from the Update Server, checks the nonce and both signatures (vendor and Update Server signature) with the public keys from its local trust anchor. If all verifications are successful, the IoT device installs the new public key in its trust anchor.

D. Security Discussion

The Update Server needs information like `KeyID` and `Algorithm` from the `KeyInfo` structure to validate the manifest. Therefore, this is included in the manifest. If an attacker takes over the vendor server, (s)he also controls and may manipulate the manifest. Hence, the signature validation of the Update Server is susceptible to a substitution attack: Two different manifests could theoretically have the same signature computed under different curves. The attacker may then specify a different curve in the `KeyInfo` field. This is also described in the security considerations of RFC 8152. The probability for two valid manifests to produce the same signature using different elliptic curves is highly unlikely. Furthermore, there is a second wall of defense on the IoT device. The manifest validation done by the IoT device is not susceptible to this attack since it does not use the algorithm specified in the manifest. It uses the algorithm specified in its local trust anchor.

III. IMPLEMENTATION

To validate our approach and estimate the implementation effort, we extended the MYNO Update Protocol (MUP). The MYNO-Framework is written in Python and uses the requests library to download manifests and firmware images. The Python libraries requests and urllib3 still do not support DANE functionality. The urllib3 library uses either the SSLContext provided by the Python implementation or pyopenssl. Both CPython, which is Python's reference implementation, and pyopenssl use the C OpenSSL library in the end which provides the ability to DANE enable a given SSLContext. Additionally, the corresponding TLSA-Records must be downloaded and added to the SSLContext.

Figure 3 shows our implementation via pyopenssl including all modified libraries. For the requests and urllib3 library we implemented options to enable and disable DANE support. The CFFI bindings of the library cryptography were also extended to provide python bindings for the necessary OpenSSL methods. Pyopenssl in turn was extended to DANE enable the

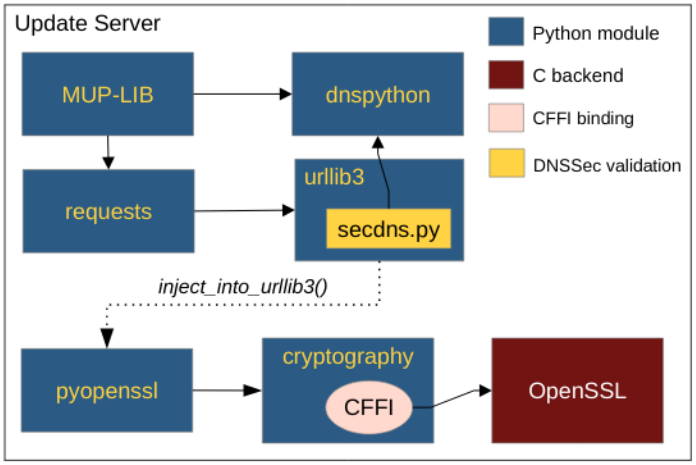

Figure 3. Integration of DANE Validation into MUP and Python

Table II
IMPLEMENTATION EFFORT IN LOC

Library	Added LoC
requests	7
urllib3 incl. secdns	328
pyopenssl	9
cryptography	8
dnspython	0
MUP-DANE	193

SSLContext if required. The TLSA record, which is added to the SSLContext in pyopenssl, is retrieved in urllib3.

To secure DNSSEC's last mile the DNSSEC verification is not done by the DNS resolver but done on the edge device. We implemented this in a new secdns module within urlib3 as seen in Figure 3. This needed 328 lines of code (see Table II).

Further, the MYNO-Framework was extended by a MUP-DANE module which includes the extended manifest downloader, the querying of the IPSECKEY records (public keys) and the verification of manifests (193 lines of code). All other modifications were minor changes within the Python libraries to make the OpenSSL DANE enabling functionality known.

We tested our implementation extensively. The MYNO-Framework including our MUP-DANE extension was installed on a Raspberry Pi 4, which was used as an edge-device. As IoT device the nRF52840-DK with Contiki-NG was used. Our implementation and the ontology of the IoT device were extended with the key rollover functionality. More implementation details are given in [17]. The update process and key rollovers were tested with the domain `mup.dnssec-uni-potsdam.de`. The testing of the update process and key rollovers were combined with DNSSEC ZSK and KSK key rollovers and TLSA/IPSECKEY record updates.

IV. DISCUSSION

Compared to other approaches like CAA and OCSP our proposal relies on a single source of truth, i.e. the domain name system. Instead of maintaining revocation lists or certificate root stores on the device the vendor only needs to maintain the relevant domain records.

It is well-known that the dissemination of DNSSEC is only slowly increasing and could be much further. Since using DANE without DNSSEC is not recommended, we want to discuss the experiences and typical prejudices regarding DNSsec:

1) DNSsec is a performance killer.
2) DNSsec management is too complicated.
3) DNSsec increases the risk of DDOS.

DNSSEC signed zones are about three times larger [18] and depend on the key size. Verisign for example, moved from a 1024-Bit to a 2048-Bit key and "The size of the root zone file jumped from 1.6 to 2.1 MB."[1] This is not exhausting our Internet infrastructure. Signing of zones up to 100 000 records happens within 1 minute on current hardware [18]. Further, the signature checks are distributed by design. Currently, there are 1372 of 1487 Top-Level-Domains signed, which shows the deployment status of DNSSEC[2].

No doubt, DNSSEC needs some training effort. But the management tools have significantly improved over the last 15 years [19].

In case of an incident a short TTL for the IPSECKEY and TLSA records is important, to ensure fast propagation of the updated records.

There still remains the problem with (unnecessary) open DNS resolvers which can potentially be misused in a reflection and amplification DDOS attack [20, 21]. Compared to DNS, DNSSEC returns a significantly larger response message since the records include the signatures. This is beneficial for the attacker. But again, the problem is not DNSSEC but the misconfiguration of the DNS server.

V. CONCLUSION

The firmware update of IoT devices is typically done via TLS encrypted communication. But the trustworthiness of PKIs is not always given and other approaches have to be considered. We propose to secure the firmware update process by integrating the validation of public keys with DNSSEC/DANE. Further, we demonstrate how DNSSEC and IPSECKEY records can be used to establish a key rollover process. This allows fast key revocation, since only the DNS records have to be updated with the new key material. This avoids the known overhead of certificate revocation lists and the related privacy issues.

We implemented a prototype of the DNSSEC/DANE-based approach for the MUP protocol, but it can be easily integrated with any other SUIT-conform update protocol. We showed that our approach works well for IoT devices in an edge computing architecture. But it also works for cloud based deployments, in which the IoT device itself is connected to the Internet and DNSSEC/DANE capable. In the cloud scenario, the task of the Update Server is done directly by the IoT device.

[1] Verisign Blog
[2] https://stats.research.icann.org/dns/tld_report/

The proposed DNS scheme can be easily scaled up for multiple different firmwares by including the App-ID in the FQDN. This keeps the zone for each app/firmware small and easily manageable via `nsupdate`.

References

[1] Bundesamt für Sicherheit in der Informationstechnik (BSI), "BSI Grundschutz, SYS.4.4: Allgemeines IoT-Gerät," 2020.

[2] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[3] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes, "Iotls: Understanding tls usage in consumer iot devices," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 165–178. [Online]. Available: https://doi.org/10.1145/3487552.3487830

[4] Owen Williams, "Google to drop china's cnnic root certificate authority after trust breach," https://thenextweb.com/news/google-to-drop-chinas-cnnic-root-certificate-authority-after-trust-breach, 2015, [20.06.2022].

[5] Jürgen Schmidt, "Nvidias geleakte code-signing-zertifikate missbraucht," https://www.heise.de/news/Nvidias-geleakte-Code-Signing-Zertifikate-missbraucht-6537255.html, 2022, [13.03.2022].

[6] Rob Wright, "23,000 symantec certificates revoked following leak of private keys," https://www.techtarget.com/searchsecurity/news/252436120/23000-Symantec-certificates-revoked-following-leak-of-private-keys, 2018, [13.03.2022].

[7] Seltzer, Larry, "Chrome does certificate revocation better," https://www.zdnet.com/article/chrome-does-certificate-revocation-better/, 2014, [20.06.2022].

[8] A. Langiu, C. A. Boano, M. Schuß, and K. Römer, "UpKit: An Open-Source, Portable, and Lightweight Update Framework for Constrained IoT Devices," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019, pp. 2101–2112.

[9] K. Sahlmann, V. Clemens, M. Nowak, and B. Schnor, "MUP: Simplifying Secure Over-The-Air Update with MQTT for Constrained IoT Devices," *Journal Sensors (Open Access)*, Dec 2020. [Online]. Available: https://www.mdpi.com/1424-8220/21/1/10

[10] B. Moran, H. Tschofenig, D. Brown, and M. Meriac, "A Firmware Update Architecture for Internet of Things," RFC, IETF, RFC 9019, April 2021.

[11] B. Moran, H. Tschofenig, and H. Birkholz, "An Information Model for Firmware Updates in Internet of Things (IoT) Devices," RFC, IETF, RFC 9124, January 2022.

[12] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable Key Compromise in Software Update Systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 61–72.

[13] M. S. Lenders, C. Amsüss, C. Gündogan, T. C. Schmidt, and M. Wählisch, "Securing Name Resolution in the IoT: DNS over CoAP," in *Proceedings of the CoNEXT Student Workshop*, ser. CoNEXT-SW '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 11–12. [Online]. Available: https://doi.org/10.1145/3488658.3493790

[14] H. Lee, I. Ashiq, M. Müller, R. van Rijswijk-Deij, T. Kwon, and T. Chung, "Under the hood of DANE mismanagement in SMTP," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/lee

[15] R. Alexander, S. Aliciuc, and S. Ashton, "Thread Specification," Online, Thread Group, Specification, 2017. [Online]. Available: https://www.threadgroup.org/ThreadSpec

[16] IANA, "Cbor object signing and encryption (cose)," https://www.iana.org/assignments/cose/cose.xhtml, 2022, [15.03.2022].

[17] Wolf-Jörgen Stange, "Absicherung von Firmware-Updates im Internet der Dinge mittels DNSSEC/DANE," University of Potsdam, Master Thesis, 2022.

[18] LRZ Dokumentationsplattform, "DNSSEC Grundlagen," https://doku.lrz.de/display/PUBLIC/DNSSEC+Grundlagen, 2022, [12.03.2022].

[19] Internet Society Foundation, "Dnssec tools," https://www.internetsociety.org/deploy360/dnssec/tools, 2022, [20.06.2022].

[20] Yazdani, Ramin, "Open DNS resolvers, from bad to worse," https://blog.apnic.net/2022/05/13/open-dns-resolvers-from-bad-to-worse/, 2022, [24.06.2022].

[21] R. Yazdani, R. van Rijswijk-Deij, M. Jonker, and A. Sperotto, "A Matter of Degree: Characterizing the Amplification Power of Open DNS Resolvers," in *Passive and Active Measurement*, O. Hohlfeld, G. Moura, and C. Pelsser, Eds. Cham: Springer International Publishing, 2022, pp. 293–318.