On Criteria and Tooling for Cryptographic Inventories

Nicolai Schmitt[1] Johanna Henrich[2] Dominik Heinz[3] Nouri Alnahawi[4] Alexander Wiesmaier[5]

Abstract: When cryptography becomes insecure, a migration to new schemes is required. Often the migration process is very complicated, but the time available is very limited. Only if the used cryptographic algorithms, protocols and configurations are known can a system be efficiently and fully adapted to changed security situations. This creates the need for a crypto-inventory that gathers this knowledge. Consequently, the question arises what criteria a crypto-inventory must fulfill to support this adaptation. It also highlights the need for tools to assist compilation. We therefore conducted a literature survey and extracted key requirements. Missing content was supplemented by expanding existing requirements or adding new ones. Furthermore, appropriate metrics were assigned to assess the fulfillment of the requirements for a certain crypto-inventory implementation. Regarding the tooling, we identified five major areas of interest — installed software, connected hardware, communication, stored data and source code scanning — and provide prototypes for semi-automatic creation of crypto-inventories for three of them. This provides organizations with a starting point to understand their cryptographic landscape as a prerequisite for crypto-agility and crypto-migration. However, theoretical design and prototypes have not yet been evaluated. This will be done as a follow-up to this work. All types of organizations are invited to participate.

Keywords: Crypto-Inventory; Crypto-Agility; Automated; Tooling; Requirements; Metrics; PQC; Migration; Cryptography

1 Introduction

Increasing digitalization and networking in a wide variety of areas not only offer great advantages, but also great potential threats. Cryptography includes mathematical functions as well as associated procedures and protocols ensuring security goals like confidentiality, integrity and authenticity. It is therefore an important tool for detecting and preventing attacks. However, cryptographic primitives and protocols always have and will be subject to the threat of compromise, which implies the need to substitute endangered schemes. The transition

[1] Darmstadt University of Applied Sciences, Department of Computer Science and Information Technology, Schöfferstraße 3, 64295 Darmstadt, Germany, nicolai.schmitt@h-da.de
[2] Darmstadt University of Applied Sciences, Department of Computer Science and Information Technology, Schöfferstraße 3, 64295 Darmstadt, Germany, johanna.henrich@h-da.de
[3] Darmstadt University of Applied Sciences, Department of Computer Science and Information Technology, Schöfferstraße 3, 64295 Darmstadt, Germany, dominik.heinz@h-da.de
[4] Darmstadt University of Applied Sciences, Department of Computer Science and Information Technology, Schöfferstraße 3, 64295 Darmstadt, Germany, nouri.alnahawi@h-da.de
[5] Darmstadt University of Applied Sciences, Department of Computer Science and Information Technology, Schöfferstraße 3, 64295 Darmstadt, Germany, alexander.wiesmaier@h-da.de

doi: 10.18420/sicherheit2024_003

from classical to quantum-safe cryptography is currently the most prominent and far-reaching example. Sufficiently powerful quantum computers (QC), which may be available in some years, would be able to break asymmetric schemes in polynomial time [Be09, BL17]. Table 1 (Appendix) shows which areas of cryptography are affected according to the current state of research. Alternative approaches urgently need to be identified and integrated into the existing infrastructure. In 2016, the NIST initiated the Post Quantum Cryptography (PQC) standardization process to develop and standardize new cryptographic schemes that are secure even when sufficiently powerful quantum computers become available [U.16]. Some algorithms have already been selected for standardization [Al22]. Similar examples, such as the migrations from SHA-1 to SHA-2, have shown that these processes are time and resource intensive [CLO21]. However, proactive design of software and digital networks and a carefully developed migration strategy can help to address this issue [HHW23]. For an efficient and error-free migration, systems should be as crypto-agile as possible. This means that when the security situation changes, a system is able to adapt as efficiently as possible without significant impact on the remainder [Al23, Pa22]. A first step on the way to crypto-agility is knowing what cryptography is currently in use throughout the IT-Landscape [HHW23, Pa22]. During this step, the used cryptographic algorithms and their application have to be gathered in a cryptography inventory. Requirements to crypto-inventories are not yet standardized. However, compiling a crypto-inventory within large organizations is difficult and time-consuming [HHW23]. Therefore it is recommended to be automated [CLO21], but there is no one-fits-all solution. Cryptographic primitives appear in a wide variety of contexts, including network traffic, software packages and specific cryptographic data, stored on disk. A uniform way for detecting cryptography seems unfeasible, as different approaches must be taken.

Although the topic of crypto-nventories gained more interest due to the quantum-threat, this paper solely addresses how crypto-nventories can be compiled and maintained as completely and accurately as possible. Hence, questions regarding an actual migration of cryptography based on an inventory is outside the scope of this work. Section 2 reviews the current state of the literature and describes existing approaches for crypto-inventory creation. Based on this, Section 3 lists requirements as well as different tooling categories. These key requirements were collected during a literature research. Where necessary, requirements have been merged or expanded. Appropriate descriptions and metrics for assessing a complete implementation have been added. In addition, two requirements were supplemented. They could not be merged with other requirements, although the properties of existing work related to crypto-migration and crypto-agility were found to be very relevant. The complete list of requirements is shown in Table 3. Based on this, five different tooling categories are given that should be considered when compiling the inventory. Furthermore, we provide practical examples of tools to demonstrate for three different categories how a concrete implementations for a semi-automated compilation can take place. Sections 4, 5 and 6 describe these tools, and therewith a starting point is given that already identifies a wide range of used cryptography. Finally, a conclusion and an outlook are given in Section 7.

2 Related Work

The NIST migration to Post-Quantum Cryptography project[6] aims to provide a guideline to increase crypto-agility and to prepare for a migration. A key point of this is discovering public key cryptography in use. The project furthermore aims to encourage the use of automated discovery tools. In [BPS21] Barker et al. describe the identification of usage of public-key cryptography as a prerequisite for the migration to PQC. Further aspects of the discovery of usage characteristics are discussed. In [Pa22] Sebastian Paul focuses on the preparation of migration to PQC in the IIOT (Industrial Internet of Things) domain. Thereby, the need for a crypto-inventory in order to become crypto-agile was highlighted. Furthermore, fundamental requirements and aspects for an inventory entry for cryptographic usage findings in source-code were discussed. However, the work lacks of a broader view that goes beyond static code analysis. It does not aim for the completeness of requirements and is missing concrete information or examples for later implementation. In [CLO21] Campagna et al. further identified the need for ”automated tools that inventory cryptography usage across compute infrastructure [...] and to test for the results of migration” [CLO21]. ETSI TR 103 619 [ET20] provides guidelines for a transition from a non-quantum-safe state to a quantum-safe state. Their first step is the compilation of a crypto-inventory. Furthermore, different aspects and requirements for crypto-inventories are featured. Also in 2020, Cryptosense published a white paper about the topic crypto-inventory [Cr20]. The white paper suggests the use of a crypto-inventory to support the migration to PQC, as it ”can help with this process by allowing the identification of cryptographic keys that need to be migrated and highlight what data is currently protected by encryption” [Cr20] and goes into detail about the requirements for crypto-inventories. The PQC-Migration-Handbook published by TNO, CWI and AIVD in 2023 introduced a persona-based model on how to migrate to PQC. The model assesses the urgency of the issue and the steps to be taken based on the role and data available to an organization [TCA23]. The handbook also advises the use of a crypto-inventory in order to reduce the risks and costs of the migration later on [TCA23]. In 2020 TNO published a position paper on the migration to quantum-safe cryptography [MdiMvH20]. It includes an action plan for successful migration in the form of a ladder model, that shows the steps required for a successful migration process. The model incorporates the idea of creating a crypto-inventory as one of the ”initial no-regret moves” [MdiMvH20, fig.1]. In [Ma21] Ma et al. describes a framework for risk assessment based on an asset inventory that incorporates the idea of a crypto-inventory. Thereby multiple aspects of a crypto-inventory are discussed. In [Ha23] Hasan et al. propose a framework helping to manage migration, that includes a crypto-inventory as well as the processes required to assess risks and start the transition of cryptography.
Regarding tools for automated compilation of crypto-inventories, Cryptosense also offers a cryptography-analyzer tool that performs dynamic and static analysis of software during development through CI buildchain integration and automated report generation to compile a crypto-inventory. [Cr20]. In addition, Cryptosense provides an online port scanner and

[6] `https://www.nccoe.nist.gov/crypto-agility-considerations-migrating-post-quantum-cryptographic-algorithms` (28.10.2023)

TLS security scan tool, that checks the cryptography and TLS version offered by the server to analyze [Cr20]. As of now, Cryptosense is part of SandboxAQ.[7]
IBM offers a roadmap on how to become quantum safe together with a set of tools, incooperating the idea of becoming crypto-agile. The roadmap consists of 3 stages, which are Discover, Observe and Transform. The tools help to move through these stages by discovering cryptographic usage through scanning of source code and dependencies, and by observing the network and IT environment. Findings are reported in the json based CBOM[8] format for cryptographic assets, and can be viewed in a graphical representation. [HM21]
The German Federal Office for Information Security (BSI) is developing a project called TaSK[9], which analyzed TLS servers, TLS clients as well as eID clients for compliance with the technical guideline TR 03116-TS [BS23]. The project consists of a framework with a testcase runner, that can generate reports and a standalone TLS test tool which can perform different TLS handshakes and acts as a TLS client or server.
In addition, other works emphasize the need for a crypto-inventory in order to ensure a successful and efficient migration, but do not go into detail about its implementation [HHW23, IS20, Al23].
Based on this, it is clear that the creation of a crypto-inventory can generally be seen as one of the first steps towards successful migration and crypto-agility. Automating the creation or maintenance of the crypto-inventory reduces the required effort and helps to keep it up-to-date and useful, as these are demanding tasks [CLO21]. However, none of the listed works covers the sum of all the requirements identified in the literature. In addition, the importance of some properties is emphasized without defining a specific requirement. As different notions of crypto-inventories exist having different requirements, there is a need to unify these definitions and to add missing aspects. Moreover, there is a lack of metrics to assess whether a requirement has been met, thus making the theoretical concept applicable in practice.

3 Theoretical Concept

In order to gain a comprehensive understanding of crypto-inventory requirements a literature survey was performed. Thereby, definitions, requirements and usage-scenarios in literature were examined. Further details are given in the following paragraphs. Afterwards, five categories are presented on how cryptography in use can be detected automatically. These categories are intended to work together, rather than to be seen as standalone choices. Also they may not provide full coverage, so further categories may be required depending on the concrete infrastructure.

Requirements In this paragraph the harmonized key requirements are described. In addition, missing aspects were added by extending requirements or creating new ones if no

[7] `https://www.sandboxaq.com/post/why-cryptosense-is-joining-sandboxaq` (11.10.2023)
[8] `https://github.com/IBM/CBOM` (11.10.2023)
[9] `https://github.com/BSI-Bund/TaSK` (01.11.2023)

suitable subject was available. Table 3 and Table 4 (Appendix) contain a more detailed list including a clear description as well as quantitative or qualitative metrics.
The harmonized key requirements are as follows: A crypto-inventory is generated in an automated way and has to identify: cryptography in use and its active usage, data protected by the cryptography, location of cryptographic objects, dependencies and how keys are managed. Furthermore, it has to be up-to-date, it has to enable arranging entries into categories, it has to be query-able to access information efficiently during a migration and it should point out obstacles to a migration.
Beyond that, Risk Assessment [ET20] and Sensitivity [Ma21] are only partially in scope: Both requirements describe the assessment of risk depending on the data protected. The Risk Assessment can be either identified as a step following the compilation of a crypto-inventory as described in [Ha23], or as a requirement for a crypto-inventory. However, since risk assessment is rather needed for an actual migration of a system, it was not considered as key requirements for a crypto-inventory. As the knowledge about protected data is a key requirement, the risk-assessment could be implemented building upon the data knowledge.
In addition, two new requirements were created. Firstly, in the event of a sudden threat, a system may need to be adapted quickly and yet correctly. For this purpose, the necessary measures should be known in advance, e.g., by means of a migration plan [Ma21, HHW23]. It follows that an inventory should already indicate what alternative cryptographic technologies could be used, or whether only a shutdown of the subsystem is possible. Information for action in case of threat should be directly available. Secondly, different works highlight the general advantages and necessity of hybrid procedures [Pa22, OPp19, Gi23, Ma21]. In this context, hybrid is defined as the combination of cryptographic methods serving the same purpose. The resulting cryptographic material is secure as long as one of the two underlying methods is unbroken [GHP18]. These information should be directly acquired during crypto-inventory creation.

Tooling Categories Compiling a crypto-inventory within large organizations is difficult and time consuming because many heterogeneous components use cryptography and must be considered. During the literature review it became clear that a system can be viewed from different angles. In the following, five major categories are presented as to how the cryptography in use can be identified. Our concept is based on the different migration scenarios described by NIST [BSN21]. However, our categories differ, because the selection was based on implementation similarities of analysis technologies.
First, it is important to consider what **software applications** have been installed. Based on this, insecure software can be identified. This can be done, e.g., by analyzing the package manager used by the system as described in Section 4. Alternatively, the application behavior can be analyzed in a dynamic way, as described by SandBoxAQ[10].
Besides the installed software, the used **hardware** must also be taken into account. For this purpose the hardware interfaces could be analyzed, e.g., by means of the underlying operating system. The **communication** between software and network components can also

[10] `https://www.sandboxaq.com/solutions/security-suite` (31.10.2023)

be considered. Insecure messages are identified and traced to their source. This can be done using network analysis or network sniffing tools. It should be noted, however, that traffic may be fragmented or encrypted. A comprehensive analysis via network sniffing therefore requires advanced knowledge of the system. Furthermore, it is important to consider what conclusions can be drawn from the **stored data**. E.g., protected and encrypted data, as well as stored keys and certificates provide information about schemes used and associated configurations. Metadata such as name, location or time of use can be used to draw further conclusions about use and relevance. Finally, an identification of insecure cryptographic primitives can also be achieved by static **source code** analysis. As described in [Pa22], existing tools can be extended to include cryptographic aspects.

The subsequent sections detail three prototypes created as a proof-of-concept for a certain category. The **Package Manager Analyzer**, Section 4, examines the software applications used, the **Network Analyzer**, Section 5, can be used to analyze the system's communications and, as a third and final exemplary tool, the **SSH Key Analyzer**, Section 6, examines the stored data in the form of SSH keys.

4 Package Manager Analyzer

Notion A package manager is a software application that is installed on a Linux operating system and is responsible for managing the installation, removal, and upgrading of programs and libraries. While the implementation of package managers might be different across various Linux distributions, the fundamental concept remains the same. To perform operations, a package manager communicates with one or more central repositories to retrieve software packages. The remote repositories contain the software packages and also the associated metadata, such as the package name, package version and a list of its dependencies. The package manager also has to maintain a local database of the available packages and the corresponding metadata on the system, which is synchronized with the upstream repository. When the user issues a request to update a package, the package manager performs an initial dependency check and installs any necessary dependencies. Following this, the package manager downloads and installs the package itself. If required, a package manager can also perform post-installation tasks, such as removing artifacts generated during the update process. However, the provided features vary depending on the underlying Linux distribution. The most popular Linux distribution is Ubuntu, which comes with the apt package manager. Regular updates are crucial to patch security vulnerabilities in installed software. In the context of post quantum cryptography, this is especially important, because updates can also make changes to the implemented cryptographic behavior of packages, either by deprecating older algorithms, or adding newer ones. One such example that comes in mind is OpenSSH, which added experimental support for PQC XMSS keys in OpenSSH 7.6[11]. It is safe to assume, that with the increasing adoption of PQC, more software will add

[11] `https://www.openssh.com/releasenotes.html` (11.10.2023)

PQC functionality in the upcoming years. With package managers being the primary way to obtain updates, they also make a good component to analyze a system for PQ-resilience.

Implementation Upgrading a system to a state that is fully PQ-resilient is a process that can only be partially automated, for several reasons. Modern systems are very intricate and heterogeneous, and can vary in terms of format, configuration, platform, and other aspects. Therefore, the prototype presented in this paper is supposed to assist a system administrator in the upgrading process by identifying and reporting the components on the system that need to be upgraded. The upgrading process still needs to be performed manually by the administrator. Currently the prototype supports the package manager apt[12], which is typically used on Debian and Ubuntu based Linux distributions. Support for other package managers could be added in a similar manner by implementing parsers for their command-line tools to query installed packages or by implementing support to read their databases. To identify the relevant packages, the prototype performs several operations described in the following section.

1. **Aggregation of installed packages** Initially, the tool has to obtain information about the packages that are currently installed on the system. This is achieved by parsing the list of packages, provided directly by the package manager, including the package name and the package version.
2. **Identification of Insecure packages** In the next step, every package needs to be evaluated to determine if it is secure in accordance with the required security standards or not. Currently, this step is performed by comparing each package with its exact version, or a range of versions against a knowledge base. These knowledge bases have to be compiled manually, with additional entries of vulnerable packages being added over time. If the evaluation deems a package insecure, it is flagged as such.
3. **Identification of affected dependencies** Any packages that identified as PQ-insecure in the previous stage, will have their incoming dependencies (i.e., other packages that depend on them) included as potentially insecure as well. This step is necessary, as a package that depends on another package that is deemed PQ-insecure may leverage some of its functionality, thereby causing the PQ-insecure functionality to propagate through the dependency chain.
4. **Export of findings** The findings are then exported as JSON file, containing the package name, the package version, the vulnerable dependency etc.

5 Network Analyzer

Notion Network-traffic is a security-critical component, as it could pass third-party-components like firewalls or routers or insecure networks alike the Internet. Network-traffic could potentially be recorded and later cryptographically analyzed by an attacker. As there

[12] `https://ubuntu.com/server/docs/package-management` (11.10.2023)

is a broad variety of software using encrypted communication, it might not be clear for every application which cryptographic scheme is used during communication. A rather often used protocol to secure network traffic is TLS (Transport-Layer-Security), which allows one to establish a secure communication channel between two endpoints, ensuring the security objectives of confidentiality, integrity and authenticity [MI04]. It is the most widely used application data security protocol in the world and was used in more than 90% of all Internet connections in 2020 [LKK21]. The TLS Handshake is the central protocol when establishing a new TLS connection. It is responsible for negotiating the cryptographic schemes used, authenticating the communication partners and exchanging keys. That means that, for the purpose of inventorying the cryptographic primitives used, the TLS handshake messages are particularly important. TLS uses asymmetric cryptographic methods to exchange a transport key which is then used for a symmetric encryption of payload data. The Open-quantum-safe project already works on the integration of PQC into the TLS protocol [SM17]. Also, newer versions of the chrome browser added support for the PQC-scheme Kyber[13]. The Network-Analyzer captures the in- and outgoing traffic and analyzes the cryptographic methods used in secured protocols. These are then reported together with IP-addresses, port numbers and the application name on the local machine.

Implementation The network-analyzer is written in C and uses the pcap[14] interface to capture network-traffic on a specified network-interface. As TLS was used in more than 90 percent of the Internet traffic [LKK21] in 2020, the prototype supports TLS as a proof of concept. Support for other security protocols, e.g., SSH can be implemented in a similar manner. Extension and rework of data models of the prototype is required in order to do so. The prototype is currently only compatible with TLS 1.2 as the handshake design in version 1.3 is different in some respects. Nevertheless, an extension for version 1.3 analysis is planned for the future. To determine the cryptographic schemes used for a certain connection, the prototype performs the following steps:

1. **Filter for IPv4 and TCP packages** As a first step, packages are filtered for beeing IPv4 and for being tcp-traffic. When the package evaluated to be IPv4 and TCP, the IP-Address and TCP-Port are extracted from there headers.
2. **Evaluate TLS record header** As a next step, the fields of the TLS record header are evaluated to check if the package contains a TLS handshake. This evaluation might result in false positives, as other data transfered with other protocols might accidently look like a valid handshake header.
3. **Evaluate TLS handshake header** In the following step, the handshake header is evaluated to find out if the package contains a Client Hello or Server Hello Message.
4. **Resolve Ciphersuite-IDs** In Case of a Client Hello Message, the offered Cipher Suites are given by there id and are resolved to their name using a lookup-table. In case of a Server Hello, the selected Cipher Suite is resolved.

[13] `https://blog.chromium.org/2023/08/protecting-chrome-traffic-with-hybrid.html` (11.10.2023)
[14] `https://www.tcpdump.org/manpages/pcap.3pcap.html` (31.10.2023)

5. **Resolve applicationname by local port** The application name is resolved by calling lsof [15] with the following options `lsof -n -i :PORT` with PORT replaced by the local port-number to filter for an application with a tcp connection with a local port number matching the local port number in the filtered package.
6. **Write entry into json-file** When the parsing was successful, an entry is written into an array in a json file.

6 SSH Key Analyzer

Notion Another security-critical component that is worth analayzing, is locally stored cryptographic material as e.g. SSH Key files, certificates or password files.
SSH is based on the transport layer and provides confidentiality, authenticity and integrity for network communications. It was designed specifically for remote login and operation of network services. There are two versions of SSH, with SSH-2 being the most widely used today [LY06]. SSH is designed to work with almost any public key format, cipher and algorithm (signature and/or encryption). In general, a public key type is defined by the key format, the signature and/or encryption algorithm, and the encryption of the signatures and/or encrypted data. Authentication is usually explicit, i.e., key exchange messages include a signature or other proof of the authenticity of the server. Implicit authentication is possible, but not usually provided.
Currently, the authentication mechanisms employed rely on conventional cryptographic algorithms, such as RSA, ECDSA etc. Ongoing efforts are made, to integrate PQ-resistant algorithms into OpenSSH [OQS]. Work for implementing PQ-resistant algorithms in the SSH protocol are being done. In a Linux-based environment, individual users possess their own set of SSH keys, typically stored under `/home/user/.ssh`. It is possible to identify SSH keys present on a system in an automated manner, as well as, determining the algorithms and their key sizes. Extracting this information can give insight into the security level of the SSH Key file, based on the algorithms in use.

Implementation In order to detect the stored SSH-Keys on the system and to analyze their algorithm and key length, the prototype performs the folowing steps:

1. **Aggregate SSH Key files** In order to analyze the SSH-Key files, the prototype has to locate these. Therefore the prototype iterates over the systems user-entries, which contain an entry with the users home-directory. If applicable, the prototype iterates over the key files located in each users `.ssh` directory.
2. **Create fingerprints from SSH Key files** An SSH-Key-fingerprint is created for each key file using the `ssh-keygen` [16] command in the following way:

[15] `https://man7.org/linux/man-pages/man8/lsof.8.html` (11.10.2023)
[16] `https://man.openbsd.org/ssh-keygen.1` (11.10.2023)

`ssh-keygen -l -f FILENAME`. *-l* creates a fingerprint, *-f* defines that the key file name follows.

3. **Extract information from SSH Key fingerprints** As next step, the algorithm-name and key length are extracted from the fingerprint.
4. **Determine SSH Key file owner** As next step, the owner of the file is read from the file meta-information. This is done using the Linux stat[17] system call.
5. **Writing json entry** As a last step, the collected information are written to a json file.

The analysis of other cryptographic material can be implemented in a similar manner using the appropriate toolsets to the corresponding file formats.

7 Conclusion and Future-Work

In this paper, we presented key requirements for a crypto-inventory, based on a literature review. From this, we identified five categories of tools for analyzing a system and its infrastructure. Furthermore, we presented the crypto-detection toolbox, a prototypical proof-of-concept for a crypto-inventory compilation software. It consists of three main components, each corresponding to a particular category. Although our initial evaluation of the toolbox was restricted to a theoretical approach and to functional testing on some small-scale applications (e.g., Linux clients and internal webservers), our work meets the following important requirements defined in Section 3. This is because the toolbox supports automated discovery that covers network traffic, local key storage and package manager dependencies, fully meeting the requirements of automation, cryptography in use, categorization, key management and location. However, it remains to test whether these requirements can be quantified and qualified via suitable metrics.
The introduced prototype is planned to be extended by several other modules to support the detection of a wider range of cryptography in use. For instance, the package-manager analyzer can be extended to support further package-managers (e.g., pacman, rpm) or container systems (e.g., snap or docker). Moreover, it may also be extended to analyze and report the availability of HSM and TPM modules in a given system. A code scanning tool may also be developed to cover the remaining categories of connected hardware and source code scanning. On another note, the prototype currently creates reports in a json based format using a custom data model. It is planned to make it compatible with the CBOM format, since its predecessor is already an established standard. As the need for establishing crypto-agility and enabling cryptographic migrations is becoming greater, compiling a crypto-inventory is a must. We consider this work a step in that direction, and extend an invitation to researchers and to the industry to test and evaluate our toolbox on large-scale IT-systems and platforms. Therefore, we provide our current implementation as an open-source code on GitLab[18].

[17] `https://man7.org/linux/man-pages/man2/stat.2.html` (11.10.2023)
[18] `https://gitlab.com/pqc-cdt` (08.01.2024)

Bibliography

[Al22] Alagic, Gorjan; Apon, Daniel; Cooper, David et al.: NIST IR 8413-upd1: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process. Technical report, July 2022. https://doi.org/10.6028/NIST.IR.8413-upd1 (last accessed 2023-10-11).

[Al23] Alnahawi, Nouri; Schmitt, Nicolai; Wiesmaier, Alexander; Heinemann, Andreas; Grasmeyer, Tobias: On the State of Crypto-Agility. Technical report, 2023. https://eprint.iacr.org/2023/487.

[Be09] Bernstein, Daniel J.: Introduction to post-quantum cryptography. In: Post-Quantum Cryptography. Springer, 2009.

[BL17] Bernstein, Daniel J.; Lange, Tanja: Post-quantum cryptography. Nature, 549(7671):188–194, 2017.

[BPS21] Barker, William; Polk, William; Souppaya, Murugiah: Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms. Technical report, National Institute of Standards and Technology, April 2021.

[BS23] BSI: Technical Guideline TR-03116-TS. Technical report, BSI, May 2023.

[BSN21] Barker, William; Souppaya, Murugiah; Newhouse, William: Migration to Post-Quantum Cryptography. Technical report, NIST, April 2021.

[Ch16] Chen, Lily; Jordan, Stephen; Liu, Yi-Kai; Moody, Dustin; Peralta, Rene; Perlner, Ray; Smith-Tone, Daniel: Report on Post-Quantum Cryptography. Technical Report NIST IR 8105, National Institute of Standards and Technology, April 2016.

[CLO21] Campagna, Matt; LaMacchia, Brian; Ott, David: Post Quantum Cryptography: Readiness Challenges and the Approaching Storm. Technical report, January 2021. arXiv:2101.01269 [cs].

[Cr20] Cryptosense: CRYPTOGRAPHY INVENTORY. white paper, Cryptosense, February 2020.

[ET20] ETSI: Migration strategies and recommendations to Quantum Safe schemes. Technical report, 2020.

[GHP18] Giacon, Federico; Heuer, Felix; Poettering, Bertram: KEM Combiners. In (Abdalla, Michel; Dahab, Ricardo, eds): Public-Key Cryptography – PKC 2018. Springer International Publishing, Cham, pp. 190–218, 2018.

[Gi23] Giron, Alexandre Augusto: Migrating Applications to Post-Quantum Cryptography: Beyond Algorithm Replacement. Technical report, 2023. https://eprint.iacr.org/2023/709.

[Ha23] Hasan, Khondokar Fida; Simpson, Leonie; Baee, Mir Ali Rezazadeh; Islam, Chadni; Rahman, Ziaur; Armstrong, Warren; Gauravaram, Praveen; McKague, Matthew: Migrating to Post-Quantum Cryptography: a Framework Using Security Dependency Analysis. Technical report, July 2023. arXiv:2307.06520 [cs].

[HHW23] Hohm, Julian; Heinemann, Andreas; Wiesmaier, Alexander: Towards a Maturity Model for Crypto-Agility Assessment. In (Jourdan, Guy-Vincent; Mounier, Laurent; Adams, Carlisle; Sèdes, Florence; Garcia-Alfaro, Joaquin, eds): Foundations and Practice of Security. Springer Nature Switzerland, Cham, pp. 104–119, 2023.

[HM21] Harishanker, Ray; Mays, Edward: IBM bringing organizations along quantum-safe journey. Technical report, February 2021.

[IS20] ISARA: Managing Cryptographic and Quantum Risk. Technical report, ISARA, June 2020.

[KNW18] Kreutzer, Michael; Niederhagen, Ruben; Waidner, Michael: Eberbacher Gespräch: Next Generation Crypto. Technical report, 2018.

[LKK21] Lee, Hyunwoo; Kim, Doowon; Kwon, Yonghwi: TLS 1.3 in Practice: How TLS 1.3 Contributes to the Internet. In: Proceedings of the Web Conference 2021. pp. 70–79, 2021.

[LY06] Lonvick, Chris M.; Ylonen, Tatu: The Secure Shell (SSH) Transport Layer Protocol. Technical report, 2006.

[Ma21] Ma, Chujiao; Colon, Luis; Dera, Joe; Rashidi, Bahman; Garg, Vaibhav: CARAF: Crypto Agility Risk Assessment Framework. Journal of Cybersecurity, 7, 02 2021.

[MdiMvH20] Muller, Ir. F.; drs. ir. M.P.P. van Heesch: Position Paper, MIGRATION TO QUANTUM-SAFE CRYPTOGRAPHY. Technical report, 2020.

[MI04] Mahboob, Athar; Ikram, Nassar: Transport Layer Security (TLS)–A Network Security Protocol for E-commerce. Technocrat PNEC Research Journal, 1:2004, 2004.

[OPp19] Ott, David; Peikert, Christopher; participants, other workshop: Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility. Technical report, 2019.

[OQS] Open-Quantum-Safe: Open-quantum-safe/openssh: Fork of openssh that includes prototype quantum-resistant key exchange and authentication in SSH based on liboqs. Technical report.

[Pa22] Paul, Sebastian: On the Transition to Post-Quantum Cryptography in the Industrial Internet of Things. PhD thesis, Technische Universität Darmstadt, Darmstadt, 2022.

[SM17] Stebila, Douglas; Mosca, Michele: Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In (Avanzi, Roberto; Heys, Howard, eds): Selected Areas in Cryptography – SAC 2016, volume 10532, pp. 14–37. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[TCA23] TNO; CWI; AIVD: The PQC Migration Handbook, Guidelines for migrating to post-quantum cryptography. Technical report, March 2023.

[U.16] U.S. National Institute of Standards and Technology (NIST): Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. Technical report, 2016.

A Appendix

Scheme	**Algorithm**	**Type**	**Quantum-Resistance**
Symmetric Key Encryption	AES	Block Ciphers	Requires Larger Keys
Hash Function	SHA-2	One Way Functions	Requires Larger Output
	SHA-3	One Way Functions	Requires Larger Output
Key Exchange	DH	Discrete Logarithm	Not Secure
	ECDH	Discrete Logarithm	Not Secure
Public Key Encryption	RSA	Integer Factorizing	Not Secure
	ECC	Discrete Logarithm	Not Secure
Digital Signatures	RSA	Integer Factorizing	Not Secure
	DSA	Discrete Logarithm	Not Secure
	ECDSA	Discrete Logarithm	Not Secure

Tab. 1: Quantum-Resistance of Classical Cryptography - Adapted from [Ch16, KNW18]

Primitive	**Hardness Assumption(s)**
Lattices	Learning with Errors (or Rounding) / Shortest (Closest) Vector / Shortest Integer Solution / NTRU
Code-based	Syndrome Decoding (Error Correcting Codes)
Multivariate	Hidden Field Equations / Random Multi-Quadratic Systems / Extended Isomorphism of Polynomials
Isogenies	Supersingular Elliptic Curve Isogenies
Hash-based	Hash Resistance (One-Way Functions)

Tab. 2: Overview of PQC Primitive Families and Hardness Assumptions

No.	Name	Sources	Description
1	Automated	[CLO21] [Ha23]	The crypto-inventory is compiled using automated tools.
2	Cryptography in use and active usage knowledge	[Cr20], [Ma21] [ET20] [BPS21] [Pa22] [BPS21]	The crypto-inventory contains information about the cryptographic scheme/algorithm itself, as well as associated configurations and the owning application/protocol.
3	Up-to-date	[Cr20] [TCA23] [Pa22]	The crypto-inventory contains up-to-date information.
4	Protected data knowledge	[Cr20] [ET20] [BPS21]	The crypto-inventory contains information about what data is protected.
5	Key management knowledge	[Cr20] [Ma21] [ET20]	The crypto-inventory contains information about how keys and certificates are managed and stored
6	Categorizable	[Ma21]	The crypto-inventory provides the ability to arrange entries into different scopes that help organizing the migration later on.
7	Location Knowledge	[Ma21]	The crypto-inventory identifies the location within the infrastructure an entry belongs to.
8	Dependencies	[Ma21] [ET20] [BPS21]	The crypto-inventory indicates whether the object under consideration has dependencies on other objects inside or outside the system, and lists them if applicable.
9	Query-ability	[Cr20] [Ha23]	The crypto-inventory is query-able in the sense that it provides the necessary interfaces to access the inventory data in a usable way.
10	Obstacle detection	[Cr20] [BPS21]	The crypto-inventory identifies obstacles to crypto-agility.
11	Emergency plan		The crypto-inventory should be able to indicate whether alternative technologies can be used in the event of a threat and, if so, provide detailed information on adaption.
12	Hybrid usage		The crypto-inventory provides information on whether a hybrid implementation of the cryptographic primitive under consideration is possible in principle, or is even used. If so, all information necessary for the implementation is described.

Tab. 3: Requirements on a cryptographic inventory.

No.	Name	Metric	Note
1	Automated	Coverage rate of system under consideration that can be analyzed automatically out of all application scenarios and technologies in scope.	Depending on the (sub)system under consideration and the technologies involved, different tools may be required to automatically analyze the cryptography used.
2	Cryptography in use and active usage knowledge	Coverage rate of analyzed system, for which the mentioned information is known and listed in the inventory.	It should be described whether and how cryptographic material (e.g. keys) is actually used, and whether the use is cryptographic or non-cryptographic in nature. A separate percentage for the nature of the application (i.e. cryptographic vs. non-cryptographic) could also be considered. Additionally, the version number should be indicated if present.
3	Up-to-date	Coverage rate in terms of up-to-date entries in the crypto-inventory as a metric for an up-to-date inventory.	In case keys are managed by a software system, an integration into the crypto-inventory is recommended. For a software-product in development, this could be achieved by an integration of generating/updating the crypto-inventory into the build-pipeline. It is always advisable to adapt or recompile the inventory when a new version of a software package is released, as the applications may have inter dependencies.
4	Protected data knowledge	Coverage rate of cryptographic material where, as a minimum, the category (e.g., login credentials, customer data, intellectual property, ...) of data is known.	
5	Key management knowledge	Coverage rate of entries for which the above information is provided.	It should state whether key management software is used and whether the data is hard-coded. Furthermore, the validity of key material (creation and expiration data) should be taken into account in order to estimate effort of changing keys and warn in case of expired keys. Entries that do not include cryptographic keys or certificates were not considered while calculating the coverage.
6	Categorizable	Coverage rate in terms of entries assigned to a category.	The different categories should be defined in advance.
7	Location Knowledge	Coverage rate of entries assigned to a location.	Possible locations, such as on-premises or in the cloud, should be determined in advance.
8	Dependencies	Coverage rate of entries where provider/owner is specified	Dependencies exist whenever other components would also be affected by a change in the cryptographic material and relevant functionalities can only be maintained if all parties involved are informed and agree. Dependencies concern vendors of a crypto-solution, the owner of cryptographic material in use, all types of supply chains or equal communication partners. Depending on the implementation, the listing may be done by cross-referencing to other inventory entries.
9	Query-ability	As meaningful evaluation of this requirements would require the effective use of the crypto-inventory, no metric is given.	The demands on query-ability highly depends on the infrastructure to analyze. Anyhow in order to aggregate data on demand in case of complicated questions, data might be stored in a SQL-capable relational database.
10	Obstacle detection	Coverage rate of inventory entries, which are clearly assigned whether they contain "hard-to-migrateäspects.	Examples of obstacles to crypto-agility include hard-coded information and key length constraints. Additional information, such as data size limits in applications or protocols, is required to fulfill this requirement. In the case of build integration, it is recommended that sources are scanned for static keys and that key length limits in critical applications are known. It is also recommended that long-lived keys are taken into account.
11	Emergency plan	Coverage rate of inventory entries for which it is indicated whether an alternative proceeding is applicable.	This requirement describes measures that can be applied immediately, not changes that are theoretically possible but not yet available. Detailed information for the transition may be, e.g., alternative cryptographic schemes. Where appropriate, these may take the form of references to other inventory entries. If no direct alternatives are applicable, it should be described how to proceed instead or where further information is available.
12	Hybrid usage	Coverage rate of inventory entries for which it is indicated whether hybrid executions are possible or in use.	Examples of implementation information include the number and type of primitives, the key derivation function used, and the corresponding configurations.

Tab. 4: Metrics and Notes on the Requirements