Schlüssel und Zertifikate in der Architektur

Kryptografie ist ein wesentlicher Baustein sicherer Software. Zertifikate und Schlüssel korrekt einzusetzen, gehört zu den Aufgaben der Softwarearchitektur.

Von Gerald Richter

In Zeiten, in denen es für alles eine App gibt und sich alles aus dem Homeoffice fernsteuern lässt, ist es undenkbar, Systeme nur physisch abzusichern und die Kommunikation mit der Außenwelt zu verhindern. Unternehmen sammeln Daten zentral oder lagern gleich die gesamte Software in die Cloud aus. Das gilt sowohl in der klassischen IT als auch im OT- (Operational Technology) und IoT-Umfeld. Viele Geräte und sogar Gegenstände haben eine Verbindung ins Internet, Produktionsanlagen stellen Daten zentral beispielsweise als digitalen Zwilling zur Verfügung, und selbst die totgeglaubte Industrie 4.0 zur Vernetzung von Lieferanten, Produzenten und Kunden erfährt mit Manufacturing-X (die Beschreibung dazu und weitere Quellen finden sich unter ix.de/zyue) neue Aufmerksamkeit. Die Europäische Union hat mit dem EU Cyber Resilience Act eine Rechtsvorschrift geschaffen, die Sicherheit für Software innerhalb der Mitgliedsländer vorschreibt und Security by Design fordert.

Security muss am Anfang des Designprozesses von Software stehen. Angriffe, die Buffer Overflows, inkorrekte Parametervalidierung oder Schwachstellen in der Nebenläufigkeit ausnutzen, sind nach wie vor an der Tagesordnung. Teams müssen solche potenziellen Schwachstellen bereits beim Design der Software berücksichtigen, sowohl bei der Auswahl der Tools und Programmiersprachen als auch bei der Architektur.

Dieser Artikel beleuchtet vor allem einen Teilaspekt der Architektur, der für die Security wichtig ist: den richtigen Einsatz von Kryptografie. Sie verhindert keine Schwachstellen im Code, schützt aber sowohl die Daten, mit denen die Software arbeitet, als auch das Programm. Damit zielt sie auf die in der IT-Security üblichen Schutzziele ab: Vertraulichkeit, Integrität, Authentizität.

Zur Vertraulichkeit gehört, dass Daten nur der- oder diejenige zu Gesicht bekommt, der sie auch sehen soll. Vertraulichkeit entsteht klassisch durch Verschlüsselung. Das zweite Schutzziel Integrität stellt sicher, dass sich die Daten nicht manipulieren lassen: Wer die Daten empfängt, muss sicherstellen können, dass diese auf dem Weg vom Sender nicht verändert wurden. Hier helfen digitale Signaturen. Die Authentizität sorgt dafür, dass die Gegenstelle der Kommunikation diejenige ist, die sie zu sein vorgibt. Das verhindert, dass sich jemand mittels eines Man-in-the-Middle-Angriffs dazwischengeschaltet hat. Dieses Schutzziel lässt sich mit kryptografisch verifizierbaren Identitäten umsetzen.

Codesigning gegen Manipulation

Kryptografie hilft in vielen Bereichen der Softwareentwicklung, die genannten Schutzziele umzusetzen. Sie kann zunächst die Anwendung selbst schützen, damit Angreifer die Integrität nicht angreifen können, indem sie Funktionen umprogrammieren oder Schadcode einbringen und damit beispielsweise Daten direkt an der Quelle anzapfen oder manipulieren.

Digitale Signaturen sollen sicherstellen, dass niemand die Software manipuliert hat. Dazu erhält das Binary eine Signatur mit einem privaten Schlüssel, der mit dem öffentlichen Schlüssel als Pendant überprüfbar ist. Wesentlich ist dabei, dass auch die Integrität der Software geschützt ist, die die Signatur prüft. Dazu dient in der Regel eine Kette von Prüfungen. Dabei wird der erste Teil der Software, die Firmware, das BIOS beziehungsweise UEFI, durch die Hardware selbst auf Integrität geprüft.

Die Firmware kann dann die Signatur des Bootloaders prüfen, der die Signatur des Kernels untersucht, der wiederum sicherstellt, dass die Anwendung korrekt signiert ist. Die Prüfkette darf keine Lücke aufweisen, da diese ein Einfallstor für einen Angriff wäre. In Systemen, die man vollständig kontrollieren kann, wie bei Embedded-Systemen, kann und muss man auch selbst sicherstellen, dass die Kette vollständig implementiert ist. Die einzige Herausforderung dort besteht in den begrenzten Hardwareressourcen.

Auf Systemen, die man nicht komplett unter Kontrolle hat, kommen oft Zertifikate für das Codesigning zum Einsatz. Hierbei muss das Betriebssystem den öffentlichen Schlüssel als Gegenstück des zur Signatur verwendeten privaten Schlüssels nicht kennen. Es muss lediglich in der Lage sein, die Echtheit des Zertifikats zu verifizieren, da der öffentliche Schlüssel Teil des Zertifikats ist. Eine solche Verifikation erfolgt mittels einer Zertifizierungsstelle (Certificate Authority, CA). Diese fungiert als vertrauenswürdiger Dritter, der die Zertifikate signiert hat, sodass man wiederum mit dem öffentlichen Schlüssel der CA die Gültigkeit der Zertifikate prüfen kann.

Bei Windows, macOS, iOS und Android lassen sich unsignierte Binaries ohne explizite Bestätigung nicht ausführen. Auch stehen hier die entsprechenden Tools für Entwickler zur Verfügung, um Codesigning einfach in den Entwicklungsworkflow zu integrieren. An anderen Stellen, beispielsweise im Embedded-Bereich, aber auch unter Linux sieht es jedoch anders aus. Hier kommen oft noch unsignierte Binaries zur Ausführung, da unter Umständen das Ökosystem im Betriebssystem oder in der Entwicklungsumgebung fehlt. Oft hat die Hardware nicht die notwendige Rechenleistung oder keine Möglichkeit, die Firmware zu prüfen. Das bedeutet, dass bei der Softwarearchitektur auch die Architektur der Hardware entscheidend ist, um sichere Software zu entwickeln.

Sichere Updates

Wer beim Softwaredesign alle Punkte berücksichtigt, kann sich trotzdem nicht in Sicherheit wiegen. Bugs werden oft nachträglich entdeckt oder kryptografische Verfahren gebrochen. Auch kommt es vor, dass jemand zu einem späteren Zeitpunkt Schwächen in zuvor als sicher angesehenen Protokollen entdeckt. Für eine sichere Software ist es deswegen unumgänglich, Updates durchführen zu können. Der EU Cyber Resiliance Act (CRA) fordert für alle in der Europäischen Union in Handel gebrachten Produkte lebenslange Softwareupdates – mindestens für fünf Jahre, wenn die Lebenszeit des Produktes nicht nachgewiesen kürzer ist.

X-TRACT

- Richtig angewandt, tragen Zertifikate und die dazugehörigen Schlüssel zu einer sicheren Softwarearchitektur bei.
- Sie stellen das Erreichen der Sicherheitsziele Vertraulichkeit, Integrität und Authentizität sicher.
- Codesigning, sichere Softwareupdates, Maschinenidentitäten, Verschlüsselung und Krypto-Agilität müssen umgesetzt werden.
- Korrektes Zertifikats- und Schlüsselmanagement ist kritischer Bestandteil der Softwaresicherheit.
- Der Cyber Resiliance Act der EU fordert diese Maßnahmen in Zukunft für alle Waren mit digitalen Elementen.

EU Cyber Resiliance Act

Gemäß Artikel 10 (1) müssen Hersteller gewährleisten, dass die Eigenschaften ihrer Hard- und Softwareprodukte Mindestsicherheitsanforderungen bei Konzeption, Entwicklung und Herstellung erfüllen:

Pflichten der Hersteller

Schutzziele	Schutz der Vertraulichkeit von Daten durch Verschlüsselung mit modernsten Mechanismen, Schutz der Integrität von Daten, Befehlen, Programmen und Konfigurationen vor Manipulation, Schutz vor unbefugtem Zugriff durch Authentifizierungs-, Identitäts- oder Zugangsverwaltungssysteme, Verfügbarkeit wesentlicher Funktionen, einschließlich Abwehrfähigkeit und Eindämmung von Denial-of-Service-Angriffen.
Security by Default	Auslieferung mit einer sicheren Standardkonfiguration mit Option zum Reset auf diese Konfiguration, Die Verarbeitung ist auf die für den Zweck angemessenen und relevanten Daten zu beschränken.
Security by Design	Minimierung der eigenen negativen Auswirkungen auf die Verfügbarkeit der Dienste anderer Geräte oder Netze, Minimierung der Angriffsflächen inklusive externer Schnittstellen, Implementierung geeigneter Mechanismen zur Minimierung der möglichen Auswirkungen eines Vorfalls.
Monitoring	Loggen sicherheitsbezogener Informationen sowie einschlägiger interner Vorgänge (beispielsweise Zugang zu Daten, Diensten oder Funktionen und Änderungen daran).
Sicherheitsupdates	Regelmäßige Sicherheitsupdates zur Behebung von Schwachstellen, gegebenenfalls auch durch automatische Aktualisierungen.

Während Mobile-Apps via App Store eine Updatemöglichkeit ebenso mitbringen wie Linux-Distributionen, müssen sich Softwarehersteller an anderen Stellen eigenhändig darum kümmern. Sie müssen einen Prozess bieten, um Softwareupdates (automatisch) zu verteilen, und dafür sorgen, dass dieser Prozess sicher ist. Ansonsten sind Aktualisierungen das perfekte Angriffsziel. Hier gilt es vor allem die Integrität sicherzustellen, indem der Hersteller die Updates signiert und diese Signatur beim Updateprozess überprüft wird.

In der Softwareentwicklung kommen zudem oft viele Bibliotheken zum Einsatz. Das Entwicklungsteam ist somit auch Konsument von Inhalten und muss für eine sichere Einbindung sorgen. Es darf nicht einfach einen Container von Docker Hub oder ein Modul aus einer beliebigen Registry einbinden, sondern muss sicherstellen, dass die Authentizität (Woher stammt die Library?) und die Integrität (Ist sie nicht manipuliert?) sichergestellt ist.

Jenseits der Kryptografie fordert der Cyber Resilience Act die für sicheres Design unumgänglichen Software Bills of Material (SBOM), die angeben, welche Komponenten in eine Software eingeflossen sind, um sie gegen bekannte Schwachstellen zu prüfen. Letztlich ist der Softwarehersteller dafür verantwortlich, einen sicheren Updateweg zur Verfügung zu stellen. Er kann sich nicht auf öffentliche Repositories verlassen, über die er keine Kontrolle hat.

Aufgabe der Softwarearchitektur ist es, Binaries, Container und Module, soweit sie nicht in signierter Form in öffentlichen vertrauenswürdigen Repositories zur Verfügung stehen, ausnahmslos auf Sicherheitslücken zu prüfen und zu signieren. Vor allem muss sichergestellt sein, dass das System nur Komponenten ausführt, deren Signatur vorher geprüft wurde.

Identitäten für Systeme und Geräte

Moderne Applikationen bestehen meist nicht mehr aus einem monolithischen Ganzen, sondern teilen sich auf, etwa in Microservices, Mobile-Apps, Webanwendungen oder Anwendungen für IoT-Devices. Das Zusammenspiel erfordert viel Kommunikation, die oft nicht nur im sicheren Rechenzentrum erfolgt. Dadurch lässt sich Sicherheit nicht mehr durch physische Kontrolle sicherstellen, sondern verlangt andere Methoden.

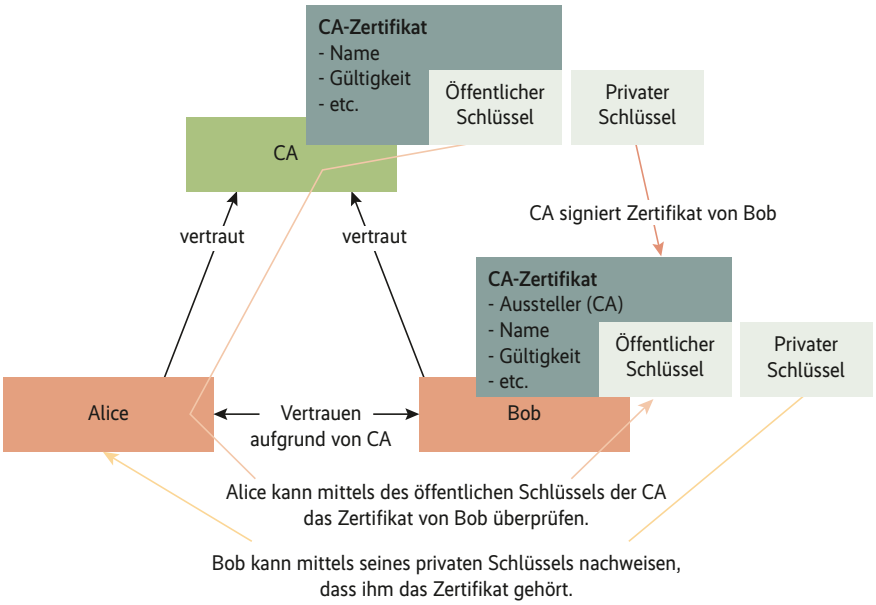
Hier kommt Zero Trust ins Spiel: Zero Trust bedeutet nicht, dass es kein Vertrauen gibt oder das es gar ohne Vertrauen geht, sondern das Vertrauen erst erworben werden muss. Das Konzept besagt, dass jede Kommunikationsbeziehung, ohne Ausnahme, erst authentisiert und autorisiert werden muss, bevor eine Kommunikation stattfinden darf. Für die Authentisierung bedarf es einer sicher feststellbaren Identität. Für Menschen bedeutet das meist Username/Passwort (hoffentlich) in Kombination mit einem zweiten Faktor oder neuerdings auch WebAuthn. Für Geräte, Systeme und (Micro-)Services kommen typischerweise X.509-Zertifikate zum Einsatz.

Blick auf X.509-Zertifikate

Zu einem Zertifikat gehört immer ein Paar aus öffentlichem und privatem Schlüssel. Das Zertifikat fügt diesem Schlüsselpaar Metadaten wie Name, Gültigkeit und Verwendungszweck hinzu. Der öffentliche Schlüssel ist Teil des Zertifikats, während der private Schlüssel geheim gehalten und sicher aufbewahrt werden muss. Ein Zertifikat wird mit dem privaten Schlüssel der CA (des Ausstellers) signiert.

Mit dem öffentlichen Schlüssel der Zertifizierungsstelle kann das Gegenüber prüfen, dass die CA das Zertifikat signiert hat. Wenn man also der Zertifizierungsstelle vertraut, kann man jedes Zertifikat, das diese ausgestellt hat, auf Gültigkeit prüfen. Der öffentliche Schlüssel eines Zertifikats ermöglicht es, Signaturen und verschlüsselte Daten zu verifizieren oder zu entschlüsseln, die mit dem zugehörigen privaten Schlüssel eines Zertifikats erstellt wurden. Das Zertifikat lässt sich problemlos übertragen, da es ausschließlich öffentliche Daten enthält.

Um sicherzustellen, dass die Verifikation beziehungsweise Entschlüsselung auch mit dem korrekten öffentlichen Schlüssel erfolgt, ist lediglich der öffentliche Schlüssel des CA-Zertifikats erforderlich. Um alle Zertifikate zu verifizieren, die eine CA ausgestellt hat und in Zukunft noch ausstellen wird, genügt es, den öffentlichen Schlüssel der Zertifizierungsstelle auf sicherem Weg zu verteilen. Das macht die Verteilung wesentlich einfacher als zu jedem Zertifikat oder Schlüsselpaar den öffentlichen Schlüssel auf gesichertem Wege zu übertragen. Das Verfahren erlaubt eine Vertrauensbeziehung zwischen Unbekannten, solange



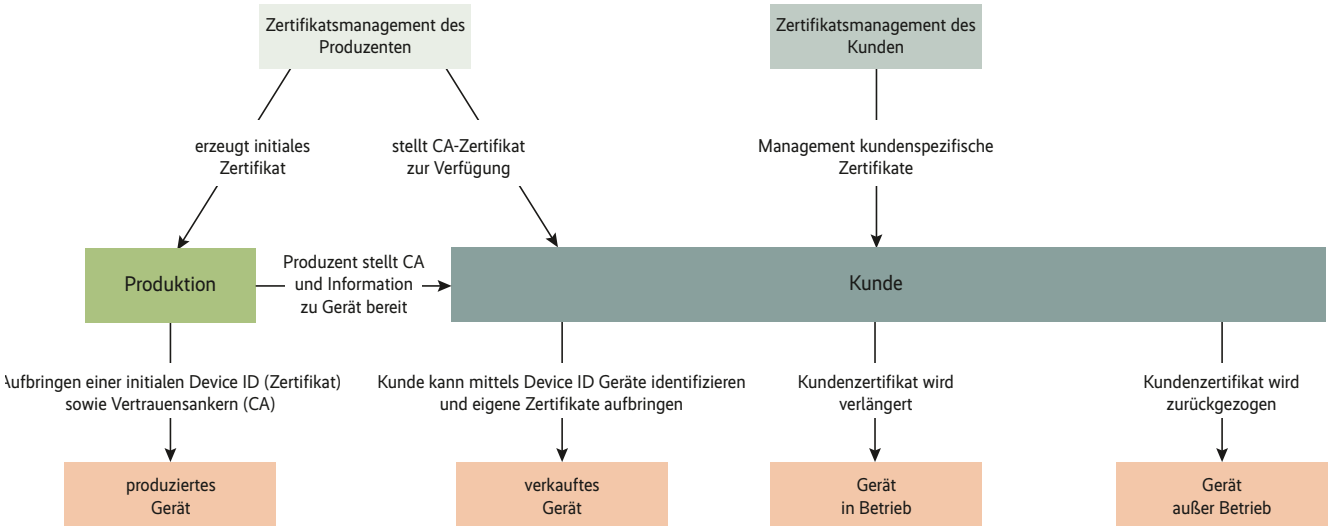
Funktionsweise der Vertrauensbeziehung zwischen Unbekannten mittels X.509-Zertifikaten (Abb. 1)

beide derselben CA vertrauen (siehe Abbildung 1).

Damit sich also ein Gerät oder Service sicher ausweisen kann, benötigt es ein Zertifikat und den zugehörigen privaten Schlüssel. Dadurch erhält es eine Maschinenidentität. Außerdem sind die CAs erforderlich, um sicherzustellen, dass die Gegenstellen der Kommunikation die erwarteten Zertifikate vorweisen können. Beides muss initial auf sicherem Weg auf das Gerät beziehungsweise System kommen.

Systeme, die ausreichende Entropie zur Verfügung stellen können, können den privaten Schlüssel direkt auf dem Gerät erzeugen. Gute Zufallswerte sind dabei essenziell, da ansonsten die Schlüssel unter Umständen vorhersehbar sind. Das kann vor allem auf Embedded-Systemen zum Problem werden. In diesem Fall sollte das Schlüsselpaar außerhalb erzeugt werden. Dafür, wie auch für den sicheren initialen Transport des CA-Zertifikats und den Transport des Zertifikats für die Identifikation ist ein Konzept erforderlich.

Unternehmen, die den Schlüssel direkt auf dem System generieren, können das zugehörige Zertifikat beispielsweise mit einem Standard-Enrollment-Protokoll wie SCEP, EST oder ACME erzeugen. In der Kubernetes-Welt hat sich Cert-Manager als Standardwerkzeug etabliert. Es kann auf der einen Seite Pods mit CAs und Zertifikaten versorgen und hat auf



Zertifikats-Lifecycle bei IoT-Geräten (Abb. 2)

der anderen Seite eine Anbindung an diverse Zertifizierungsstellen.

Verschlüsselung und TLS

Der Austausch von Daten, die über nichtvertrauenswürdige Netze gehen, erfolgt inzwischen nahezu ausschließlich verschlüsselt. Aber auch die Kommunikation über interne Verbindungen beispielsweise zwischen Microservices oder Systemen im selben Rechenzentrum sollte immer verschlüsselt erfolgen. Hat es ein Angreifer erstmal auf ein internes System geschafft, selbst auf ein unwichtiges, sind unverschlüsselte Verbindungen eine perfekte Datenquelle. Angreifer können alle Daten für ihre Zwecke missbrauchen.

Während bei der Vernetzung von Standorten meist ein VPN zum Einsatz kommt, greift man zwischen Applikationen oder Microservices meist auf TLS-verschlüsselte Verbindungen unter beispielsweise HTTPS zurück. Der Vorteil von VPN ist, dass es eine Schale um alle Inhalte legt, sodass sich die einzelne Applikation nicht mehr um die Verschlüsselung kümmern muss. Dem Vorteil von TLS, dass jede einzelne Verbindung verschlüsselt ist, steht entgegen, dass dadurch Entwicklerinnen und Entwickler den korrekten Einsatz von TLS in der Software sicherstellen müssen.

Für TLS in einer modernen Variante (TLS 1.2 oder 1.3) bieten die meisten aktuellen Libraries sinnvolle Defaults für die verwendeten Verschlüsselungsalgorithmen (Ciphers). Es schadet aber trotzdem nicht, das Vorgehen zu verifizieren und es beispielsweise gegen die Technische Richtline des BSI (TR-02102-2, siehe ix.de/zyue) zu prüfen. Da Ciphers beim Verbindungsaufbau ausgehandelt werden, ist es nicht nur wichtig zu prüfen, was tatsächlich zur Anwendung kommt. Die Anwendung muss auch die Verfahren überprüfen, die der Client oder Server anbieten, damit die Gegenstelle nicht versuchen kann, einen unsicheren Cipher auszuhandeln.

TLS setzt auf Zertifikate, die sich auch zur Authentisierung heranziehen lassen. Bei einer TLS-Verbindung benötigt der Server normalerweise ein Zertifikat, mit dem er sich identifiziert. Dieses sollte der Client unbedingt nutzen, um dieses mittels der CA auf Gültigkeit zu prüfen und sicherzustellen, dass es die erwarteten Daten enthält. Im Minimalfall ist zu prüfen, dass der Hostname, zu dem der Client die Verbindung aufgebaut hat, im Zertifikat enthalten ist. Diese Prüfung bedeutet zusätzlichen Aufwand. Wenn man sie aber unterlässt, können Angreifer sich als Proxy in die Verbindung einschleichen und mittels eines Man-in-the-Middle-Angriffs an die Daten kommen oder diese gar manipulieren. mTLS (mutual-TLS) geht noch einen Schritt weiter: Dabei hat nicht nur der Server ein Zertifikat, sondern auch der Client, der sich damit authentisieren kann. Auf die Weise kann sowohl der Server den Client identifizieren als auch der Client den Server. Das ist die Voraussetzung für eine beidseitig sichere Kommunikation.

Lifecycle der Zertifikate

Leider ist es nicht damit getan, Zertifikate einmal auf das System auszurollen, sie bedürfen auch der Pflege. Da Zertifikate ein Ablaufdatum haben, müssen sie regelmäßig erneuert werden. Das ist bei einem Cloud-Service dank der dauerhaften Verbindung einfach umsetzbar, stellt aber bei IoT-Devices oft eine Herausforderung dar (siehe Abbildung 2). Sie sind häufig nicht immer mit dem Internet verbunden und liegen unter Umständen vor der Auslieferung lange im Lager. Ein abgelaufenes Zertifikat darf daher nicht dazu führen, dass keine Kommunikation und damit auch kein Zertifikatsupdate mehr möglich ist. Wenn Geräte und Systeme ihr Lebensende erreicht haben, gilt es zudem, die zugehörigen Zertifikate zurückzuziehen, um Missbrauch zu vermeiden. Zurückgezogene Zertifikate finden sich in einer Zertifikatssperrliste (Certificate Revocation List, CRL) oder können per Online Certificate Status Protocol (OCSP) abgefragt werden. Server und Clients müssen diesen Status beim Verwenden von Zertifikaten prüfen.

Die Welt der Kryptografie wandelt und entwickelt sich ständig. Algorithmen, die vor einiger Zeit noch als sicher angesehen wurden, wie die Hashverfahren MD5 und SHA1, gelten heute als gebrochen und sollten nicht verwendet werden. Daher ist es notwendig, bereits im Designprozess einzuplanen, dass in Zukunft andere Krypto-Verfahren erforderlich sein könnten, um das Sicherheitsniveau zu halten. Das unterstreicht erneut die Notwendigkeit für sichere Softwareupdates.

Diese als Krypto-Agilität bezeichnete Möglichkeit der Anpassung wird im Zusammenspiel mit Quantencomputern umso wichtiger. Die Schätzungen, wann diese produktionsreif sein werden, gehen weit auseinander, aber sie könnten in den nächsten zehn bis zwanzig Jahren durchaus Realität werden. Quantencomputer können die asymmetrischen Verfahren brechen, die aktuell Zertifikaten zu Grunde liegen.

Die Arbeit an quantenresistenten Algorithmen hat bereits begonnen. Ähnlich wie bei früheren Verschlüsselungsalgorithmen findet ein Wettbewerb statt, um sichere quantenresistente Algorithmen zu finden. Es gibt bereits eine Reihe von als sicher angesehenen Kandidaten. Allen gemeinsam ist, dass sie aufwendiger zu implementieren sind sowie größere Schlüssellängen und mehr Rechenleistung erfordern.

Schlüssel in der Architektur

Zertifikate und die dazugehörigen Schlüssel ermöglichen – richtig angewandt – eine sichere Softwarearchitektur. Damit erreichen Projekte die Sicherheitsziele Vertraulichkeit, Integrität und Authentizität. Teams müssen Aspekte wie Codesigning, sichere Softwareupdates, Maschinenidentitäten, Verschlüsselung und Krypto-Agilität umsetzen.

Außerdem ist es wichtig, Systeme zu verwenden, die Schlüssel und Zertifikate sicher erstellen, verteilen und verwalten. Das ist nicht nur ein Wunsch der Sicherheitsverantwortlichen im Unternehmen, sondern das fordert zwingend auch der CRA der EU für alle in Handel gebrachten Waren mit digitalen Elementen.

Es reicht bei Softwareprojekten nicht aus, eben mal mittels openssl-Kommandozeile ein paar Zertifikate zu erzeugen, sondern die Sicherheit der gesamten Architektur fällt und steigt mit dem sicheren und zuverlässigen Erzeugen und Verwalten der Zertifikate und Schlüssel. Zertifizierungsstellen und Zertifikatsmanagementsysteme werden damit zu einem kritischen Teil der Sicherheitsinfrastruktur. (rme@ix.de)

Quellen

Die Links zu Manufacturing-X, dem CRA und der Technischen Richtlinie des BSI finden sich unter ix.de/zyue.

GERALD RICHTER

ist CTO und Gründer der ECOS Technology GmbH, die sich seit 25 Jahren mit IT-Sicherheit im Bereich PKI und Remote Access beschäftigt. Dabei hat er reichhaltige Erfahrung in der Entwicklung von IT-Sicherheitssoftware in verschiedensten Programmiersprachen gesammelt und liebt auch heute noch den direkten Kontakt mit dem Code.