Permission rights secure code signing service

Introduction

The CommScope PKI Center™ provides a variety of security-related services spanning several different markets. Our security services are provided to network and service operators as well as manufacturers and are utilized in numerous ecosystems. The main security offerings include:

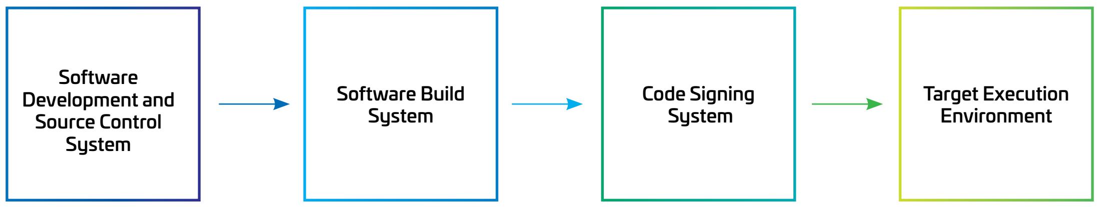
- Software platform security services, including secure code signing and debugging services for product software platforms and application code.
- Operation of certificate authorities (CAs) that issue digital certificates for high-volume manufactured products (such as cable modems, routers, set-top boxes and mobile phones), as well as for high-value infrastructure applications and server machines.
- Generating digital certificates and cryptographic keys or obtaining secure data from external licensing authorities such as CableLabs®, Digital Transmission Licensing Administrator (DTLA), Digital Content Protection (DCP), Google®, Netflix®, etc., as well as packaging, secure delivery and installation of these data into products in factories (including OEM/ODM) and repair centers.
- On-line Personalization Update System for creation and secure delivery of renewable security and personalization data to fielded devices on a mass scale.
- Security consulting to system engineering, product management and development teams about network, application, and platform security for both secure and non-secure products.

This white paper describes the secure code signing system **PRiSM** (Permission Rights Signing Manager) operated by the CommScope PKI Center. Code signing could be very complicated to someone who is just trying to protect the code he/she develops. There are different types of signing; various formats of the code image, signing keys and verification keys involved; and many other signing parameters. Even for the same product, different levels of code are likely to have totally different settings in each of these aspects.

Our WebTrust-audited ability to host CAs and our secure process standards allows software and device vendors to outsource this function with confidence. This allows these vendors to focus their energy and resources on their customers and product development. We have been providing secure code signing services since 2002, supporting a wide range of products and various use cases.

Software security life cycle

Integrity-protected software typically goes through four stages (from creation to execution) as illustrated in the following figure. It includes software development, software build, code signing, and execution environment. The software development system includes the software source control system. It allows developers to design and implement software based on functional requirements. The software build system compiles and combines the developed source code to generate a software package ready for signing. The software development system and build system may be combined as a single system, but there are also cases where they are maintained and hosted separately. The resulting software package is then submitted to the code signing system for signing. Either the code signing system or the software build system generates the final signed package, which is then deployed to the target execution environment. In the target execution environment, the signature of the software package is verified before it is accepted for execution.


Software Development and Source Control System
Software Build System
Code Signing System
Target Execution Environment

All four stages described must be secured to prevent attackers from compromising the integrity of the software. Adversaries typically attack the weakest link in the system. In this paper we are going to discuss a secure code signing system in more detail; however, the security of the other stages must also be adequate. In other words, having a secure code signing system without protecting the software build process or target execution environment will only provide a false sense of security.

In December 2020, a group of suspected nation-state cyber attackers managed to compromise the SolarWinds software development platform and injected SUNBURST malware into properly signed Orion products, allowing unauthorized access to networks of a wide range of government and private sectors [1]. It is interesting to note that the infected software package was legitimately signed. This incident demonstrates that securing a software development and build system is as important as securing a code signing system.

Shortly after the discovery of the SolarWinds SUNBURST attack, another malware named SUPERNOVA was also reported[2]. In this case, however, the malicious DLL module was not part of a signed SolarWinds package. Instead, the SUPERNOVA malware was designed to masquerade as a legitimate SolarWinds web service. Since the DLL module had not been signed, it would be allowed to execute only if the target environment did not enforce signature verification. Such unsigned malware attacks are blocked when code signature verification is enforced. It is therefore important to ensure that only properly signed software is accepted in the target platform.

Need for code signing

Software running on any electronic devices or platform is designed and developed to serve its intended purposes. A mobile banking application, for example, allows a user to manage his/her banking account online—communicating securely with the online banking service provided by the bank. A video streaming application communicates with the streaming servers to get video content and playback on the client device. A 5G virtual network function includes code developed to manage the bandwidths or other services of the mobile clients. It is crucial that these applications remain intact as they were released by their publishers—not modified, replaced or corrupted by anyone intentionally or unintentionally. To ensure that only legitimate software or scripts are executed on a device or platform, a digital signature is generated over an executable using a private signing key. The signature and the executable are packaged together for delivery to the target platform. To ensure the code is not modified, this digital signature is verified on the target platform using the corresponding verification key.

In general, there are two main approaches where signed code can be utilized: secure code download and secure boot. With secure code download, when a code image is downloaded to a device, its signature is checked before the device will accept the download and save it locally into persistent storage. This prevents any code download that is not signed properly with the correct signing key. When the device reboots, however, the code signature may not be checked again. For devices with silicon that do not support secure boot, limiting physical access to the device and hardening against remote installation of unauthenticated software remotely (including, of course, implementing secure code download) may be the best option available. Some older cable modem products, for instance, fall into this category.

However, sophisticated attackers can (and do) often find a way to bypass code authentication and install malicious attack software. A few of the many examples that allow malicious software to be installed into IoT (internet of things) and mobile devices without physical access are described in [3], [4] and [5]. An example where physical access or close proximity is initially required for hacking a smart TV device is described in [6].

The limitation of secure code download is the lack of authentication of software that has already been installed on a device. If an attacker gains access to the device and manages to modify the code responsible for verifying the code download signature, then any code can be downloaded and accepted. Alternatively, if a code signature is checked only after a download—and not every time software is loaded and executed from internal storage—an attacker with access to the device may be able to overwrite persistent storage with a malicious unsigned code image. Secure boot, on the other hand, enforces that a device is booted from trusted software all the way from boot code, to platform, OS, and applications.

In an ideal scenario, a device with secure boot will start up from hardware-protected boot code. For instance, the boot code may be programmed into read-only memory that is not modifiable unless someone physically swaps out the memory module. The boot code includes verification code that will load and verify the next boot stage, using an embedded verification key that cannot be modified. From then on, each boot stage is responsible for verifying the next software layer before executing it. This forms a strong chain of trust for all the software running on the device.

Code signing normally utilizes asymmetric cryptography, such as RSA or elliptic curve algorithms. There is a private signing key and a public verification key. The signer responsible for signing the code owns the signing key, while the verification key is to be used by the verification software. The verification key needs to be integrity-protected on the target platform, so it cannot be modified or replaced with another key. The signing key needs to be kept confidential. If it is compromised, the attacker can use it to sign any code—defeating the purpose of code signing altogether.

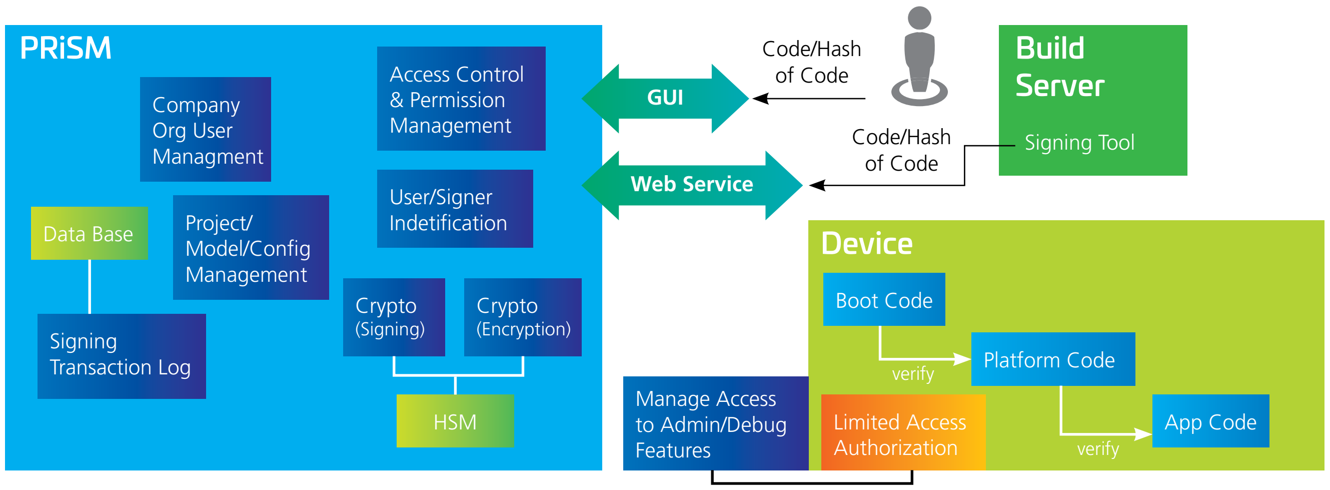
Secure code signing system

As explained above, protecting the code signing key is of utmost importance. If the code signing key is stored as a data file in a developer's laptop, it is susceptible to exposure and cloning. Multiple copies of the signing key may be created, and it will be impossible to determine who actually used it to sign a given version of the code. Even if the signing keys are well protected, code signing infrastructure with inadequate physical, network or system security may be susceptible to attacks. If an attacker manages to hack into the code signing system, even though they may not be able to extract the keys, they may still be able to direct the system to sign malicious code as they wish.

PRiSM's cloud-based code signing system architecture addresses these security issues. The system utilizes a cluster of centralized servers with a set of best-in-class hardware security modules (HSMs) and multiple layers of physical and network security. All code signing and encryption keys are protected by HSMs. All network devices and physical hosts are hardened according to the latest security guidelines in the industry. Regular and extensive network scanning and penetration testing are in place to minimize any remaining vulnerabilities for an attacker.

A conceptual view of the PRiSM code signing system is shown in the figure below. The system consists of components that manage the organizations, users, and signing configurations (as explained further in this paper). Crypto operations are performed in HSMs, which host and protect the code signing and encryption keys. For ease of use, PRiSM provides both a human GUI (graphical user interface) as well as machine interfaces. It also maintains transaction logs for all signing operations performed, which provides traceability and accountability.

On the target device or platform, signed code images of various stages are verified in sequence to ensure authenticity. For example, boot code is verified using hardware-protected firmware, which verifies the platform code. The platform code in turns verifies the applications. Apart from code integrity protection, secured access to debugging capabilities may be needed during development and troubleshooting, to allow features to be turned on or off. PRiSM provides a way to control and manage secure debugging access as described in more details in this paper.



PRiSM is designed with reliability in mind and includes failover and high availability features—ensuring that the code signing system is available whenever it is needed. PKI Center's geographically-diverse disaster recovery and business continuity plan further ensure system availability in case of natural disasters and other emergency scenarios.

Nowadays, the need for code signing and the importance of protecting the signing keys are well understood. However, it may not be cost effective for every business to invest the resources to build its own secure code signing infrastructure just to protect a handful of signing keys. PKI Center's PRiSM provides a turnkey solution for hardware and software partners to manage their code signing needs. Our WebTrust-audited ability to host CAs—and our adherence to the highest levels of security—allow software and device vendors to outsource this function with confidence. This allows our partners to focus their energy and resources on their customers and product development.

Versatile crypto support

PRiSM supports a wide variety of standard cryptographic algorithms. Currently (last revised 2020) supported code signing algorithms and signature formats include RSA PKCS #1 v1.5, RSA-PSS, ECDSA, PKCS #7, HMAC-SHA1 and HMAC-SHA256. Supported encryption/decryption algorithms include RSA PKCS #1 v1.5, RSA-OAEP and AES. Supported hash algorithms include SHA-1, SHA-2, and SHA-3 families. PRiSM also supports many industry-standard code signing formats such as Android APK signing, JAR signatures and Microsoft Authenticode. Support for new code signing standards is constantly being added as needs arise. For example, UEFI (Unified Extensible Firmware Interface) secure boot and Docker container signing are on PRiSM's roadmap.

In addition to evolutions to the above algorithms and formats, PRiSM can and does perform code encryption and signing according to customized or proprietary formats, which are often required by secure SoCs and microcontrollers. We have in-depth understanding in the security architecture and code signing formats of many vendors, including Broadcom, Infineon, Intel, MTK, STMicro, TI, and Xilinx. With the expertise and experience, we are ready to work with new code signing requirements our customers may need.

Secure debugging access

In addition to signing and encrypting code images, PRiSM provides support for secure debugging access by developers, testers and integrators. During the debugging or development phase of a product, some features may need to be temporarily turned on or off. For example, verification of applications may need to be turned off in order to debug an issue without having to sign the code every time it is modified/tweaked during the process. As another example, some physical ports (e.g., Ethernet port) may be disabled on a device by default. These ports may need to be turned on during debugging but not for production use by the end user. PRiSM provides a way to deliver a signed object, called the access token, that allows certain features to be turned on or off temporarily. The access token is tied to the device and has a configured lifetime and/or reboot count. Access token configurations are enabled by a PRiSM administrator at the direction of the product manager (PRiSM customer) to allow developers to perform debugging without compromising the security of production devices.

In some cases, a device may have a JTAG port that facilitates debugging. The JTAG port is typically disabled in production and requires a password for unlocking. PRiSM can also deliver the unique JTAG password via the access token in an encrypted form. The JTAG password does not appear in the clear during transmission or to the developer. This keeps the JTAG password confidential and therefore prevents possible compromise.

Easy management, Better control

PRiSM is a cloud-based code signing system that can control and manage many crucial security parameters. For example, there could be a security version number (or a timestamp) associated with the code for rollback prevention. When there is a major security vulnerability discovered in a version of the code, a new version is developed to address the issue. Anti-rollback control allows a software security version to be incremented such that code with a lower version would not be allowed to execute. One way to sign code with a security version may be for a developer to include security version into the code header and then submit to a code signing server for signing. Alternatively, security version number may be inserted or otherwise verified by the code signing system, making sure that only the expected value is used before signing. This ensures tight and traceable control over security-sensitive parameters.

Another example of a security parameter that can be controlled this way is a model identifier or market identifier. Each different product model or market is assigned a different value. Code signed with a certain model or market identifier can only be accepted on a target with the matching value. This provides a way to segregate code developed for different product models, markets or customers.

PRiSM organizes different code signing parameters using a hierarchical structure of signing configurations. Customers have the freedom to use this flexible structure to manage different projects, product lines, product models, market segments, different layers of code, and so on. At the bottom of the hierarchy are the actual signing configurations, each set up for a specific target use case. A configuration defines the type and format of signing, the code signing and/or encryption keys to be used, and any other parameters needed in the specific code signing process.

User and role management

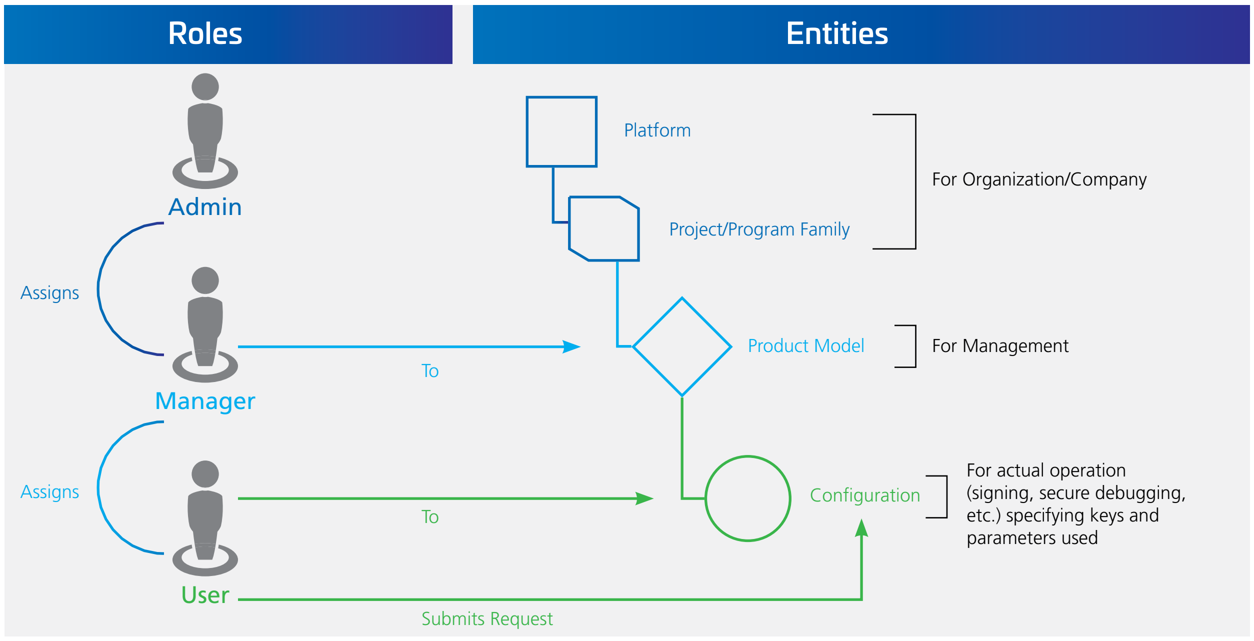
Three user roles are defined within the PRiSM code signing system: administrators, managers, and users.

PRiSM administrators are responsible for the code signing format, keys used, and any associated parameters. Code signing configurations are typically defined at the beginning of a project. Occasionally, there may be a need to update some of the configuration parameters, e.g., incrementing the secure version number. These changes are handled by PRiSM administrators and are not editable by other users. This prevents any intentional or unintentional modifications of the signing parameters and enforces strict control of the signing configurations.

PRiSM managers (CommScope or customer employees designated by the customer) assign or revoke user rights signing privileges to PRiSM configurations for users within their organization. Typically, a team lead or manager is the most familiar with the software development tasks assigned to each team member and his or her corresponding code signing needs.

PRiSM users are the actual developers who submit code for signing to the PRiSM system under the signing configuration they are authorized to use. The parameters needed for a signing operation are already defined within a PRiSM configuration. The user only needs to select the pre-defined configuration and submit the code that needs to be signed. A user will only be shown the signing configurations for which he/she is authorized. This not only enforces the separation of different vendors, groups, products, and projects, but also makes it simple and straightforward for users to sign code.

The PRiSM role separation allows for simple and effective user and signing configuration management.



Privacy protection

PRiSM provides secure code signing for multiple vendors while maintaining isolation and segregation between different parties. PRiSM managers and users are only able to access signing configurations and reports belonging to their organizations. CommScope personnel who are not providing PRiSM support, including CommScope's top management, are not permitted to access PRiSM customer information without explicit approval from the customer involved. Signing keys, configurations, and user activities are all protected and restricted to authorized parties.

In most cases, a code hash is computed locally inside a browser or a PRiSM client application and then submitted to PRiSM in an automated manner for code signature generation; the original code image is not uploaded to PRiSM. Even where the full code image needs to be uploaded, PRiSM will not store it beyond the code signing operation. Once a code signature is generated, PRiSM will erase the original image. Only a hash of the original image is archived for auditing purposes.

Different interfaces for different needs

PRiSM supports multiple interfaces for code signing, including an interactive browser-based interface and two automated machine-to-machine (M2M) interfaces. With the browser-based interface, registered users are issued secure hardware tokens for generating one-time passcodes (OTPs) for login. A user can log into PRiSM with any major browser, using a PIN and an OTP generated by the secure hardware token as credentials. PRiSM provides a simple GUI for users to submit code for signing or encryption. Users will only see the signing configurations for which they are authorized. The same browser-based interface to PRiSM provides administrative functions. PRiSM managers use the GUI to grant or remove signing privileges to users for each signing configuration they manage.

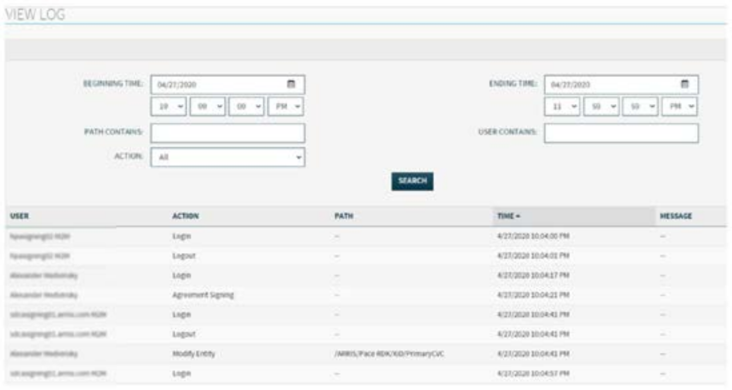
To support scenarios such as nightly software builds, PRiSM provides two different automated M2M interfaces—eliminating the need for human intervention. An M2M interface uses a USB-based hardware crypto token issued to a machine client for authentication. A Java-based code signing client interacts with the crypto token and handles all communications with the server. Simple command-line interface of the code signing client allows easy integration with build scripts for full automation. This provides a turnkey solution for code signing automation. Use of hardware crypto tokens provides a high level of trust for client identity.

In some cases, the use of hardware crypto tokens may not be feasible or required, such as for a cloud-based code signing client. For such cases, PRiSM also provides a REST API for code signing. A digital certificate is issued to the client, which will be used for client authentication to the PRiSM server over the REST API. The API is simple to use and straightforward to implement.

Audit trail and reporting

One of the major advantages of using a code signing system is the ability to maintain an audit trail for all code signing activities—ensuring that every act of code signing is accounted for. For every code signing operation performed, PRiSM records information including who or which client machine (in the case of M2M code signing) submitted the code, when the signing took place, what type of code signing and/or encryption (as indicated by the configuration) and which key and parameters were used. PRiSM records a message digest of the code submitted for each PRiSM transaction. If, later, a piece of signed code turns out to contain security vulnerabilities or other critical software bugs, the code's signature can be traced back to the original signer.

PRiSM also provides extensive reporting options, including user/signer activity reports and usage reports by configuration or by company. Such reports are valuable to project leads and managers in identifying any abnormal or suspicious PRiSM usage that may require further investigation. A couple of sample reports are shown below. The first is an activity log that shows actions performed within a time period. The second is a summary usage report that shows number of transactions per operation type within a time period.



BEGINNING TIME: 04/27/2020 10 00 00 PM

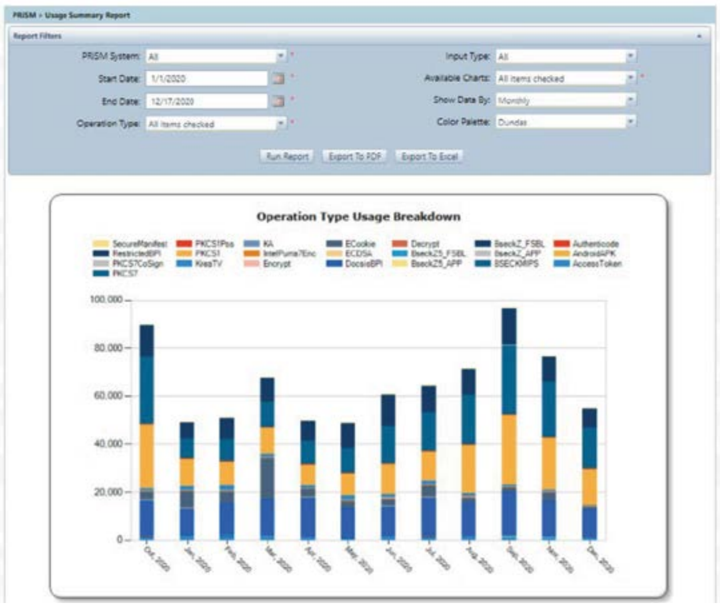
ENDING TIME: 04/27/2020 11 59 59 PM

PATH CONTAINS:

USER CONTAINS:

ACTION: All

USER	ACTION	PATH	TIME ▴	MESSAGE
[illegible]	Login	–	4/27/2020 10:04:00 PM	–
[illegible]	Logout	–	4/27/2020 10:04:01 PM	–
[illegible]	Login	–	4/27/2020 10:04:17 PM	–
[illegible]	Agreement Signing	–	4/27/2020 10:04:21 PM	–
[illegible]	Login	–	4/27/2020 10:04:41 PM	–
[illegible]	Logout	–	4/27/2020 10:04:41 PM	–
[illegible]	Modify Entity	/ARRIS/Pace RDK/XiD/PrimaryCVC	4/27/2020 10:04:41 PM	–
[illegible]	Login	–	4/27/2020 10:04:57 PM	–



PRiSM System: All

Start Date: 1/1/2020

End Date: 12/17/2020

Operation Type: All items checked

Input Type: All

Available Charts: All items checked

Show Data By: Monthly

Color Palette: Dundas

Summary

CommScope PKI Center's PRiSM is a unique world-class code signing system—providing code signing services to many companies for different kinds of devices, platforms and chips. We support a wide range of code signing and encryption standards and custom formats. We have flexibility to support any new customized code encryption and signing formats that may be required for a particular hardware or software platform, or for an ecosystem. The PRiSM team is ready to work with our customers to recommend and design a format that best suits their needs.

We provide easy and reliable 24×7 access with both an interactive web portal as well as automated interfaces—anytime, anywhere—via the internet. Its centralized and secure management and protection of signing and encryption keys ensure that our customers' valuable assets are very well protected.

References

[1] Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor

[2] SUPERNOVA: A Novel .NET Webshell

[3] Malicious Apps Bypass Security Tools to Steal Data

[4] Hacking Team's evil Android app had code to bypass Google Play screening

[5] Unsecured IoT: 8 Ways Hackers Exploit Firmware Vulnerabilities

[6] Hacking an Android TV in 2 minutes

WP-115523-EN (03/21)