Secure Design
Best Practice Guides
Release 2 November 2019

Disclaimer, Terms of Use, Copyright and Trade Marks and Licensing

Notices

Documents published by the IoT Security Foundation ("IoTSF") are subject to regular review and maybe updated or subject to change at any time. The current status of IoTSF publications, including this document, can be seen on the public website at: https://iotsecurityfoundation.org.

Terms of Use

The role of IoTSF in providing this document is to promote contemporary best practices in IoT security for the benefit of society. In providing this document, IoTSF does not certify, endorse or affirm any third parties based upon using content provided by those third parties and does not verify any declarations made by users.

In making this document available, no provision of service is constituted or rendered by IoTSF to any recipient or user of this document or to any third party.

Disclaimer

IoT security (like any aspect of information security) is not absolute and can never be guaranteed. New vulnerabilities are constantly being discovered, which means there is a need to monitor, maintain and review both policy and practice as they relate to specific use cases and operating environments on a regular basis.

IoTSF is a non-profit organisation which publishes IoT security best practice guidance materials. Materials published by IoTSF include contributions from security practitioners, researchers, industrially experienced staff and other relevant sources from IoTSF membership and partners. IoTSF has a multi-stage process designed to develop contemporary best practice with a quality assurance peer review prior to publication. While IoTSF provides information in good faith and makes every effort to supply correct, current and high quality guidance, IoTSF provides all materials (including this document) solely on an 'as is' basis without any express or implied warranties, undertakings or guarantees.

The contents of this document are provided for general information only and do not purport to be comprehensive. No representation, warranty, assurance or undertaking (whether express or implied) is or will be made, and no responsibility or liability to a recipient or user of this document or to any third party is or will be accepted by IoTSF or any of its members (or any of their respective officers, employees or agents), in connection with this document or any use of it, including in relation to the adequacy, accuracy, completeness or timeliness of this document or its contents. Any such responsibility or liability is expressly disclaimed.

Nothing in this document excludes any liability for: (i) death or personal injury caused by negligence; or (ii) fraud or fraudulent misrepresentation.

By accepting or using this document, the recipient or user agrees to be bound by this disclaimer. This disclaimer is governed by English law.

Copyright, Trade Marks and Licensing

All product names are trademarks, registered trademarks, or service marks of their respective owners.

Acknowledgements

We wish to acknowledge significant contributions from IoTSF members for this latest release of the Best Practice Guides.

Contributors

- Jeff Day, BT plc
- Roger Shepherd, Chipless Ltd
- Paul Kearney, Birmingham City University
- Karim Tobich
- Richard Marshall, Xitex Ltd
- Pieter Willems, Silex Inside
- Eric Vetillard, NXP
- Rüştü Irklı, Arçelik A.Ş.
- Trevor Hall, DisplayLink
- Chris Torr, MAOSCO Ltd
- Peter Bennett, MathEmbedded
- Michael Richardson, Sandelman Software Works

Reviewers

- IoTSF Members

Contents

The following lists the security topic areas in the IoTSF Secure Design Best Practice Guidelines (BPGs).

Each topic area has an assigned letter for easy reference. It also indicates they collectively form a set of Guidelines that should be acted on as a whole. However no specific order of reading or action is intended.

Each topic area must be studied, understood and every security item implemented where possible. These security items should be documented to form part of the overall security design of the product. Any item that cannot be implemented for good technical or business reasons must also be documented in the design.

More details about every Guideline topic area can be found on the IoTSF website.

Executive Summary

The general principles of this release of the Best Practice Guidelines apply in all market areas.

'Internet of Things' (IoT) devices fall into three main categories:

- Sensors, which gather data
- Actuators, which effect actions
- Gateways, which act as communication hubs and may also implement some automation logic.

All these device types may stand alone or be embedded in a larger product. They may also be complemented by a web application or mobile device app and cloud-based service.

IoT devices, services and software, and the communication channels that connect them, are at risk of attack by a variety of malicious parties, from bedroom hackers to professional criminals or even state actors. Possible consequences to consumers of such an attack could include:

- Inconvenience and irritation
- Infringement of privacy
- Loss of life, money, time, property, health, relationships, etc.

For vendors, operators and suppliers, potential consequences may include loss of trust, damage to reputation, compromised intellectual property, financial loss and possible prosecution.

Malicious intent commonly takes advantage of poor design, but even unintentional leakage of data due to ineffective security controls can also bring dire consequences to consumers and vendors. Thus it is vital that IoT devices and services have security designed in from the outset.

The IoT Security Foundation's Best Practice Guides provide concise essential advice on 'things to do' to help secure IoT products and systems (but we leave the designer to decide how best to do it). Links are provided to further information and discussion. The Guides are intended to be used alongside the Foundation's Security Compliance Framework.

IoT device types cover a vast range from simple one chip, low power, basic functionality devices, up to complex mains powered multi-function units. Devices at the low end of the range in particular may have security features constrained by cost, available processing power and performance, size, type of power source etc. These Guidelines aim to highlight basic principles of good practice, while recognising that many devices are not able to satisfy every requirement due to their real-world constraints. In such cases, designers must consider the trade-off between the constraints and the risks, and document the implications for where and how the device may be used if a downgrade in security results.

Although primarily aimed at IoT designers and developers, there is information for consumers as well. The IoTSF hopes the Guidelines will also empower other parties in the supply chain to ask the right questions.

A: Classification of Data

The term "data" is broad and can include functional data, data about people, collections of data and vendors' intellectual property. The degree of protection required against unauthorised viewing, changing or deletion of data depends on the sensitivity of that data. A "data classification scheme" defines a number of classes or levels of sensitivity for data and is key to its protection. Classifying data according to the scheme means the right level of security can be identified and applied, commensurate with the nature of the data being processed. Data classification will also help ensure compliance with legal regulations.

Be aware that different countries have differing laws around what constitutes 'sensitive data', where data can be held and how it must be protected in storage and transit.

1. Define a data classification scheme and document it.
2. Assess every item of data stored, processed, transmitted or received by a device and apply a data classification rating to it. Take into account that collections of data may be more sensitive than individual items and so may be classified differently.
3. Ensure the security design protects every data item and collections of items against unauthorised viewing, changing or deletion, to at least its classification rating or higher.
4. When documenting the security design, also document the data items, their classification and the security design features that protect them.

Further discussion on classification of data can be found here.

Resources on how to classify & handle data are listed below:

- Government Security Classifications
- SANS Information Classification - Who, Why and How

B: Physical Security

IoT devices are often deployed in locations that can be accessed easily for extended periods of time. This makes them liable to physical damage, tampering with switches and making connections to management, debugging and test ports. Side-channel attacks may allow the extraction of encryption keys or other data by monitoring power consumption, temperature fluctuations or electromagnetic emissions etc. Devices in the supply chain are also at risk.

Production devices can be protected against physical access to data and intellectual property by physically barring access and removing all means of unwanted connection.

1. Any interface used for administration or test purposes during development should be removed from a production device, disabled or made physically inaccessible.
2. All test access points on production units must be disabled or locked, for example by blowing on-chip fuses to disable JTAG.
3. If a production device must have an administration port, ensure it has effective access controls, e.g. strong credential management, restricted ports, secure protocols etc.
4. Make the device circuitry physically inaccessible to tampering, e.g. epoxy chips to circuit board, resin encapsulation, hiding data and address lines under these components etc.
5. Provide secure protective casing and mounting options for deployment of devices in exposed locations.
6. To identify and deter access within the supply chain, consider making the device and packaging “tamper evident”.
7. For high-security deployments, consider design measures such as active masking or shielding to protect against side-channel attacks.

Further discussion on physical protection can be found here.

Resources on how to apply physical security are listed below:

- Securing Hardware for Embedded Systems (1)
- Securing Hardware for Embedded Systems (2)

C: Device Secure Boot

The integrity of a device depends critically on executing a trusted boot sequence. A staged boot sequence, where every stage is checked for validity before initialising, minimises the risk of rogue code being run at boot time. Having a fully assured first boot stage is vital to ensuring the subsequent stages can be trusted.

1. Make sure the ROM-based secure boot function is always used. Use a multi-stage bootloader initiated by a minimal amount of read-only code (typically stored in one-time programmable memory).
2. Use a hardware-based tamper-resistant capability (e.g. a microcontroller security subsystem, Secure Access Module (SAM) or Trusted Platform Module (TPM)) to store crucial data items and run the trusted authentication/cryptographic functions required for the boot process. Its limited secure storage capacity must hold the read-only first stage of the bootloader and all other data required to verify the authenticity of firmware.
3. Check each stage of boot code is valid and trusted immediately before running that code. Validating code immediately before its use can reduce the risk of TOCTOU attacks (Time of Check to Time of Use).
4. At each stage of the boot sequence, wherever possible, check that only the expected hardware is present and matches the stage's configuration parameters.
5. Do not boot the next stage of device functionality until the previous stage has been successfully booted.
6. Ensure failures at any stage of the boot sequence fail gracefully into a secure state, to ensure no unauthorised access is gained to underlying systems, code or data (for example, via a U-Boot prompt). Any code run must have been previously authenticated.

Further discussion on secure booting can be found here.

Resources on how to boot securely are listed below:

- Securing the IoT: Part 1
- Securing the IoT: Part 2
- A Tour of TOCTOUs

D: Secure Operating System

There are many ways in which a threat agent can infiltrate an operating system. 'Hardening' the operating system helps protect against this by using the latest software, removing all unnecessary access rights and functions, and limiting visibility of the system.

1. Include in the operating system (OS) **only** those components (libraries, modules, packages etc.) that are required to support the functions of the device.
2. Shipment should include the latest stable OS component versions available.
3. Devices should be designed and shipped with the most secure configuration in place. A decision to reduce security must be a justified and documented decision made downstream from shipment if absolutely necessary.
4. Ensure the OS is securely booted.
5. Continue to update (thoroughly tested) OS components to the latest stable versions throughout the lifetime of a deployed device.
6. Disable all ports, protocols and services that are not used.
7. Set permissions so users/applications cannot write to the root file system.
8. If required, accounts for ordinary users/applications must have minimum access rights to perform the necessary functions. Separate administrator accounts (if required) will have greater rights of access. Do not run anything as root unless genuinely unavoidable.
9. Ensure all files and directories are given the minimum access rights to perform the required functions.
10. Consider implementing an encrypted file system.
11. Document the security configuration of the OS.
12. Use proper Change Control methods to manage changes to the OS.

Further discussion on securing operating systems can be found here.

Resources on how to secure operating systems are listed below:

- NIST Guide to General Server Security
- OWASP Internet of Things Project

E: Application Security

Whether using software developed in-house or 3rd party applications, good software design practices must be followed. Security must be designed in from the outset and not added on as an afterthought. A documented security design ensures subsequent issues can be more readily addressed.

1. Document the security design of applications.
2. Applications must be operated at the lowest privilege level possible, not as root. Applications must only have access to those resources they need.
3. Applications should be isolated from each other. For example, use sandboxing techniques such as virtual machines, containerisation, Secure Computing Mode (seccomp), etc.
4. Ensure compliance with in-country data processing regulations.
5. Incorporate security into all stages of the software development lifecycle, including software design, secure source code storage and traceability, code reviews, code analysis tools etc.
6. Use secure design and coding techniques. For example, sanitise and validate all data input before processing, prevent buffer overruns, use secure protocols and remove weak encryption ciphers.
7. Ensure all errors are handled gracefully and any messages produced do not reveal any sensitive information.
8. Never hard-code credentials into an application. Credentials must be stored separately in secure trusted storage and must be updateable in a way that ensures security is maintained.
9. Remove all default user accounts and passwords.
10. Ensure 3rd party application software and libraries, whether off-the-shelf or specifically developed, follow these security guidelines wherever possible.
11. Use the most recent stable version of the operating system and libraries.
12. Never deploy debug versions of code. The distribution should not include compilers, files containing developer comments, sample code, or other superfluous files.
13. Consider the impact on the application/system if network connectivity is lost. Aim to maintain normal functionality and security wherever possible.

Further discussion on securing applications can be found here.

Resources on how to secure applications are listed below:

- NIST Guide to General Server Security
- OWASP Security Testing Cheat Sheet
- SANS Web Application Security Checklist
- OWASP Data Validation
- CMU SEI secure coding standards
- OWASP Secure Coding Practices

F: Credential Management

'Credentials' are evidence of the identities of people or other entities. They can take many forms and are used to control access to data or enable secure communications. Compromised credentials are the easiest way to gain unauthorised access to data or services. Passwords, encryption keys, digital certificates and other credential data must always be handled securely and updated periodically, otherwise they can become ineffective.

1. A device should be uniquely identifiable by means of a factory-set tamper resistant hardware identifier if possible.
2. Use good password management techniques, for example no blank or simple passwords allowed, permit non-alphanumerics (e.g. + or *) as well as letters and digits, never send passwords across a network (wired or wireless) in clear text, and employ a secure password reset process.
3. Each password stored for authenticating credentials must use an industry standard hash function, along with a unique salt value that is not obvious (for example, not a username).
4. Passwords stored for use as credentials must be strongly encrypted, using an industry standard algorithm.
5. Store credentials or encryption keys in a Secure Access Module (SAM), Trusted Platform Module (TPM), Hardware Security Module (HSM) or trusted key store if possible.
6. Aim to use 2-factor authentication for accessing sensitive data if possible.
7. Ensure a trusted & reliable time source is available where authentication methods require this, e.g. for digital certificates.
8. Digital certificates require careful management as part of an effective credential solution. A certificate has a defined lifetime (and may need to be revoked) so should not be just deployed and then forgotten. Further discussion on certificates and their management is available at the link below*.
9. There must be a secure reliable means to update a digital certificate and its certificate chain on a device before it expires.
10. A certificate used to identify a device must be unique and only used to identify that one device. Do not reuse the certificate across multiple devices.
11. A 'factory reset' function must fully remove all user data/credentials stored on a device.

* Further discussion on good credential management can be found here.

Resources on managing credentials are listed below:

- How to store passwords safely
- Key Management Cheat Sheet
- An Overview of Digital Certificates

G: Encryption

Always use the strongest encryption algorithm available and only downgrade from that if absolutely necessary. Typically, any data attributable to an individual must be encrypted to ensure privacy and comply with data protection regulations. Any management data must be encrypted to protect the integrity and availability of the service.

Be aware that there are strict laws in some countries around the use and export of encryption.

1. Apply the appropriate level of encryption commensurate with the classification of data being processed.
2. Use industry-standard cypher suites, use the strongest algorithms and always use the most recent version of an encryption protocol.
3. When configuring a secure connection, if an encryption protocol offers a negotiable selection of algorithms, remove weaker options so they cannot be selected for use in a downgrade attack.
4. Store encryption keys in a Secure Access Module (SAM), Trusted Platform Module (TPM), Hardware Security Module (HSM) or trusted key store if possible.
5. Do not use insecure protocols, e.g. FTP, Telnet.
6. It should be possible to securely replace encryption keys remotely.
7. If implementing public/private key cryptography, use unique keys per device and avoid using global keys. A device's private key should be generated by that device or supplied by an associated secure credential solution, e.g. smart card. It should remain on that device (or associated solution) and never be shared/visible to elsewhere. Conversely, a device's public key may be shared elsewhere to support encryption with this device.

Further discussion on encryption can be found here.

Resources on encryption are listed below:

- OWASP Guide to Cryptography
- An Overview of Cryptography
- Understanding the 3 Main Types of Encryption
- Cryptography Just For Beginners

H: Network Connections

A device connects with the rest of the world through network connections made over one or more network interfaces. It is vital to protect these points of access, limiting possible routes into the device to the bare minimum. It is also best practice to ensure the device only makes connections it specifically requires to function and that any sensitive data (e.g. keys, personal data, passwords) exchanged over those connections are kept secret.

1. Activate only those network interfaces that are required (wired, wireless - including Bluetooth etc.).
2. Run only those services on the network that are required.
3. Open up only those network ports that are required.
4. Run a correctly configured software firewall on the device if possible.
5. Always use secure protocols, e.g. HTTPS, SFTP.
6. Never exchange credentials in clear text or over weak solutions such as HTTP Basic Authentication.
7. Authenticate every incoming connection to ensure it comes from a legitimate source.
8. Authenticate the destination before sending sensitive data.

Further discussion on securing network interfaces can be found here.

Resources on securing network connections are listed below:

- Security-Enhanced Linux
- Certificate and Public Key Pinning
- Transport Layer Protection Cheat Sheet

J: Securing Software Updates

Devices contain a variety of programmable entities, including “software”, “firmware”, “FPGA configuration”, “FLASH data”, etc. A software update (sometimes referred to as ‘patch’) provides a means to fix functional bugs and security vulnerabilities in a device.

Ideally, every device should be able to have software updates remotely managed and installed by the vendor. An alternative approach is for the vendor to publish software updates for users (device owners/operators) to download and install themselves. Users should only obtain updates from a trusted source such as the vendor’s product website.

1. Encrypt update packages to hinder reverse engineering.
2. The update routine must cryptographically validate the integrity and authenticity of a software update package before installation begins. Updates performed during a device re-boot must include these checks as part of the secure boot process - *see Best Practice Guide C: Device Secure Boot*.
3. Ensure that the package cannot be modified or replaced by an attacker between being validated and installed - a TOCTOU (Time of Check to Time of Use) attack.
4. To ensure a complete set of security fixes is applied during an update, the installation routine must automatically determine all required dependencies and install any previous versions of the software as required. If unable to resolve all dependencies, the device must be left in a safe, functioning state and details made available to the installer.
5. A ‘fail safe’ mechanism is required that will leave a device in a known safe state in the event an update fails.
6. Implement a trusted anti-rollback function, to prevent unauthorised reversion to earlier software versions with known security vulnerabilities.

Further discussion on updating software can be found here.

Resources on securing software updates are listed below:

- Sensor Network Software Update Management
- Introduction to Code Signing
- What is Code Signing?
- Firmware Update Architecture for IoT Devices

K: Logging

Event logging is vital for aiding fault and security management, and must be reliable, accessible and most likely confidential too. The integrity of logs also needs to be protected, e.g. against attackers seeking to cover their tracks. Simple battery-powered IoT devices have limited resources and may have to send events to a local hub for logging there, or forward to a central log management facility. Logs are only of value if the information they contain is examined and acted upon. They should be monitored and analysed regularly to detect potential and actual faults, security breaches and to investigate incidents retrospectively.

1. Ensure all logged data comply with prevailing data protection regulations.
2. Run the logging function in its own operating system process, separate from other functions.
3. Store log files in their own partition, separate from other system files.
4. Set log file maximum size and rotate logs.
5. Where logging capacity is limited, just log start-up and shutdown parameters, login/access attempts and anything unexpected.
6. Restrict access rights to log files to the minimum required to function.
7. If logging to a central repository, send log data over a secure channel if the logs carry sensitive data and/or protection against tampering of logs must be assured.
8. Implement log 'levels' so that lightweight logging can be the standard approach, but with the option to run more detailed logging when required.
9. Monitor and analyse logs regularly to extract valuable information and insight.
10. Synchronise to an accurate time source, where possible, so log file time stamps can be easily correlated.
11. Passwords and other secret information should not ever be displayed in logs.

Further discussion on logging can be found here.

Resources on logging are listed below:

- NIST Guide to Computer Security Log Management
- OWASP Logging Cheat Sheet
- Creating a Secure Linux Logging System

L: Software Update Policy

It is essential that processes and mechanisms for updating software are robust, reliable and secure. If devices are not updated, their security is likely to decrease over time as new attacks & vulnerabilities emerge. However, IoT devices are often constrained by limited resources, for example, energy (e.g. button battery), computation (e.g. cryptography), or communication (e.g. low bandwidth). This can limit the timeliness or size of software updates, or can even prevent being able to apply updates at all. In the latter case, very careful consideration must be given to the security implications of deploying such devices.

Both system builders and users must have a policy about updating software on devices in the field, addressing:

1. Management of all connected devices over their complete device lifecycle, including:
    a. Maintaining an active manifest of devices integrated into the user's organisation.
    b. Actively maintaining version information about software deployed on devices.
    c. Processes for planned device updating and rapid deployment of critical updates.
    d. Identification of unfixable or non-updateable devices that have known attack vectors, and processes to ensure that such devices are prevented from compromising the security of the system, for example through device revocation or some other reliable method.
    e. Securely managing change of device ownership.
    f. Securely managing devices at their end of life.

2. A clear, publicised, process for managing software errata. This process must enable developers, users and security researchers to report security vulnerabilities and other issues, and must enable rapid communication to users. It should also:
    a. Define a process for identifying affected configurations.
    b. Define the circumstances that require a software update to be developed and released.
    c. Define the urgency of releasing an update, based on the potential impact of the threat to the user, vendor and other users of the network, and impact to the user of deploying the update.
    d. Define the procedure for updating software on devices.
    e. Identify clear ownership and escalation processes within the organisation.

3. Mechanisms for software updates must be clearly defined within the software architecture.

4. The policy must recognise that existing standards for software patching, such as NIST SP800-40, may well need to be adapted for updating software on IoT systems.

Further discussion on software update policies can be found here.

Resources on software update policy are listed below:

- Management of Constrained Devices (1)
- Management of Constrained Devices (2)
- Patch Management Methodology
- Guide to Enterprise Patch Management Technologies
- Online Trust Alliance IoT Upgradability and Patching

M Assessing a Secure Boot Process

A boot process designed to be secure, complemented by software created using secure development techniques, should be resistant to malicious attack, both for normal operation, and for debug, development or analysis. A draft design for a secure boot process must be assessed to ensure it addresses the following key requirements:

1. The secure boot cannot be bypassed.
2. All code loaded as part of the boot process, unless it runs directly from ROM, is verified to ensure:
    a. The code was created by the expected, authorised source
    b. The code has not been modified since it was created
    c. The code is intended for the device type on which it is to be run
3. Verify code after it has been loaded into RAM (this is preferable to verifying it 'at rest' in storage).
4. The boot sequence starts running from ROM, using an immutable root key to verify the first code to be loaded. (Consider multiple immutable root keys for verifying different boot stages, for generating derived keys, or even for redundancy in case of subsequent compromise).
5. Modules of code are loaded progressively, but only after each previous stage has been successfully verified and booted.
6. Any existing data currently installed on the device that will be used as part of the boot configuration is checked for length, type, range etc. prior to use within the boot process.
7. At each stage of the boot process, wherever possible, the boot software checks that the hardware configuration matches the expected configuration parameters for that stage.
8. The boot process ensures that if an error occurs during any stage of the process, the device "fails gracefully" into a secure state in which RAM has been cleared of residual code. Graceful failure must also ensure the device is not 'bricked' and no unauthorised access can be made to underlying systems, code or data (e.g. via a U-Boot prompt).
9. The manufacturer implements a secure process for generating keys and certificates. The provisioning, storage and usage of keys and certificates on devices is secure. Device end-of-life management ensures the security of keys and certificates is maintained.
10. Bootloader code is updated to address vulnerabilities (although this may not be possible in the initial ROM stage).

Further discussion on securing booting can be found here.

Resources on how to boot securely are listed below:

- Securing the IoT: Part 1
- Securing the IoT: Part 2

N: Software Image and Update

One of the critical aspects of ensuring the integrity of an IoT device is ensuring that any software installed on the device is from a trusted source and has not been altered maliciously or accidentally since the software was created.

To establish the authenticity and integrity of a software update, a cryptographic signature is attached to the software package. The signature takes the form of a cryptographic one-way hash of the package, encrypted using a key available to both the signing entity (typically the Original Equipment Manufacturer - OEM) and the target device. The device verifies the package before it can be trusted and then installed.

The device confirms the authenticity and integrity of the software package by calculating for itself the cryptographic one-way hash of the software package, decrypts the encrypted hash from the signature and compares the two hashes. If they are the same, the device has verified the software package to be from the expected signing entity (e.g. OEM) and to be unaltered from when first created. If the hashes do not match, the device should discard the package as untrusted.

The key used to create the signature hash can be either symmetric (e.g. AES) or asymmetric (e.g. RSA). An advantage of symmetric cryptography is that it generally requires less memory and CPU power than is needed for asymmetric cryptography. However, secure handling and distribution of symmetric keys at scale is onerous.

Conversely, asymmetric cryptography has the advantage that the device only stores the public part of the key pair, which does not need to be kept secret. For this reason, and the easier distribution at scale, asymmetric cryptography is usually recommended for software signing.

A system builder should ensure the following:

1. The signature's cryptographic key size and hash algorithms have sufficient cryptographic strength for the intended service life of the product.
2. That the signature method chosen has a key provisioning method suitable for the intended manufacturing supply chain (bearing in mind there may be cause to replace keys in-life).
3. The system makes use of any hardware cryptography support available on the device, such as hardware key store, accelerated hashing and decryption operations.
4. The copies of the symmetric or asymmetric keys used to create the software component signatures are stored in sufficiently secure storage (e.g. Hardware Security Module - HSM), ideally compliant with international standards such as FIPS-140.

Further discussion on software signing can be found here.

Resources on software signing are listed below:

- NIST Software Considerations for Code Signing
- Code Signing
- Firmware Update Architecture for IoT Devices
- Best Practice for Code Signing

P: Side Channel Attacks

A Side Channel is an unintended/unanticipated capability to observe changes in the state of a system, where 'system' could be at the chip, board, application, device or network level.

Side Channel Attacks deduce information based on these changes and then use that information to exploit the system. For example monitoring variations in temperature, or the timing of certain events, changes in power consumption, emissions of sound or electromagnetic radiation at any frequency etc. can leak information about the system from which data may be inferred or deduced.

Monitoring of a system's behavioural symptoms may be further augmented by using Fault Injection. This technique deliberately runs a system under conditions outside those for which it was designed, and can establish side channels not available under normal operation. Such changes may be induced by variations in power supply conditions, clock timing, light exposure, temperature, vibration, electromagnetic exposure etc.

The risk of side channel attacks should not be ignored, especially when dealing with high-risk scenarios or those in harsh environments. However, anticipating and mitigating these factors can be difficult, sometimes subtle, and possibly complex to resolve. Thus, work in this area must consider the operating risks and conditions, and of course cost.

The following actions are a starting point in the kind of thinking that needs to be applied. There can be many variables in a given scenario, so not all of the following may be relevant, nor should they be considered a comprehensive list.

1. Evaluate the risks to determine the level of protection needed and the impacted modules.
2. Obfuscate signals by varying amplitude and/or time domain. Randomise jitter and delay.
3. Obfuscate hardware and software-based functions with randomised performance (or constant performance) regardless of the inputs.
4. Insert dummy operations.
5. Mask cryptographic functions and/or employ dedicated cryptographic modules.
6. Design circuitry to fail gracefully and reliably as power supply rails reach designed limits.
7. Design circuitry to fail gracefully and reliably as temperature reaches designed limits.
8. Implement error checking to combat bit flipping.
9. Include fault injection countermeasures in the design.
10. Employ appropriate mitigating physical construction (mountings, housing, EMF shields etc.).

Further discussion on side channel attacks can be found here.

Resources on side channel attacks are listed below:

- Provably secure masking of AES
- Masking against Side-Channel Attacks
- Mask Generation Functions
- Side Channel Attacks and Their Mitigation
- Side-Channel Attacks

www.iotsecurityfoundation.org