SILICON LABS

Whitepaper

How to do Failure Analysis on Locked IoT Devices

How to do Failure Analysis on Locked IoT Devices

Product returns for IoT devices can be challenging. A device should be securely locked down as much as possible before being deployed to the field. However, when a device fails in the field, you want the ability to open the device up in order to understand the root cause and remove the issue from future devices. Through new device security features, the best of both worlds can now be achieved.

Device makers focus a lot of their attention on making the best IoT product possible - making it attractive and valuable to customers, selecting the right materials, ensuring reliable manufacturing, and making sure the hardware and software are of high quality. Device makers also take care of the security of their product, making sure the software can be updated in the field so they can fix issues that arise.

As an increasing amount of their products are deployed, eventually issues will occur, and the success of their product can often depend greatly on how they address these issues. Addressing them poorly can result in angry customers, bad reviews and potentially a failed product. Addressing issues properly can make customers feel cared for and result in years of friction-free sales at large volumes.

This paper reviews how to do failure analysis on locked IoT devices, and how it can be simplified through proper observability into the product without compromising product security and intellectual property (IP) protection.

The Product Return

A device can be returned to the device maker for several reasons, including but not limited to:

- The device was ‘dead on arrival’
- The device stopped working after some time
- The device exhibits incorrect behavior

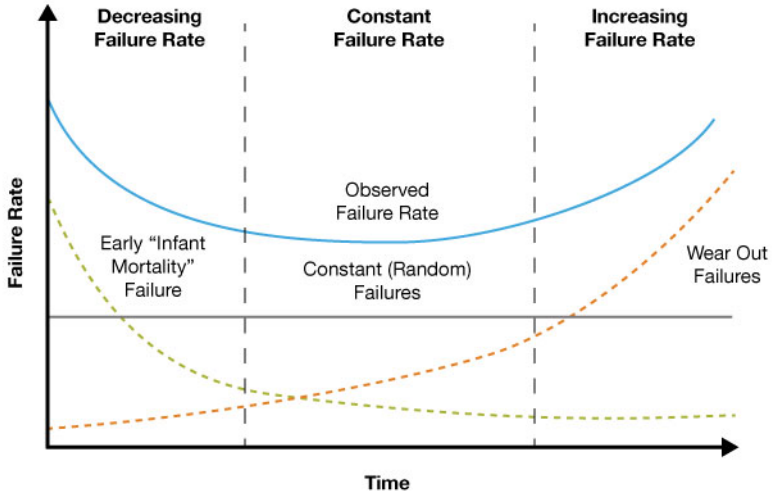
When a product is returned, large customers or distributors will want to know the impact of the issue and how to resolve it.

- What percentage of products are affected?
- What is the failure rate (once a day, once a week, once a month)?
- What is their exposure?
- What is the root cause of the failure, and your plan to fix it? How can you be sure you have found it?
- What do manufacturers need to do on their end to contain the problem for their operations or customers?
- How can they be compensated for the issue?

A device maker must be able to answer all these questions and more.

The “bathtub curve” shown below is a simple model for the failure rate of a product through its lifecycle. Through proper handling of product returns, early issues are dealt with and improved on quickly, while random and wear-out failures are efficiently identified and communicated properly. Note that all the failure components can be improved through engineering and cost tradeoffs, and the acceptable failure rate differs between product categories.

A potential challenge with product returns is that not all failing devices are returned. Often a customer will expect a certain failure rate, and frequently the device maker is not notified until the failure rate becomes significant. For mature products, this is not an issue as failures are often not due to product issues, but for new and ramping products, this can be problematic.



The "bathtub curve" is widely used in reliability engineering, and describes a typical view of device failures.

Image owned by Silicon Labs.

The Internal Process

Once a product return is received, the device maker needs to determine what priority to put on the investigation. Some products are returned as a courtesy with no real pressure to get to a resolution quickly, while others might have significant demands behind them. It is up to the company to determine what level of investigation is necessary based on the circumstance.

One way to approach a product return is with Eight Disciplines Problem Solving, also known as 8D, a method developed by the Ford Motor Company to resolve problems. Steps include:

- D0 – Problem Summary – including assessment of whether full 8D approach is necessary
- D1 – Team Formation
- D2 – Problem Definition
- D3 – Containment Plan – controlling impact while root cause and final solution are being found
- D4 – Root Cause Analysis – understanding the cause of the issue
- D5 – Corrective Action Plan
- D6 – Implement and Validate Corrective Actions
- D7 – Prevent Recurrence
- D8 – Sign Off / Commitment

Remedies to the problem come in two phases: Containment is the quick fix to minimize the impact of the issue and can be critical to resolving the problem in a timely manner. It can include but is not limited to halting firmware updates to devices, pulling affected inventory back from customers, implementing additional testing in production to catch the issue, and potentially stopping production altogether until the problem is solved. Being able to efficiently and accurately detect what products are affected can limit the negative impact of containment. In either case, communication with customers is key, as they are held responsible by their own customers and business relationships might be at stake.

The second phase of remedy is Corrective Action, applying the final fix to the problem once the root cause is understood. Finding the root cause of a problem can often be time-consuming and expensive, and there are cases where companies are unsuccessful in finding the root cause.

Note that an 8D is not just triggered based on product returns. It could also be triggered based on sudden increases in yield losses or other internally-discovered problems.

Failure Analysis

Failure Analysis or Root Cause Analysis starts at the places that are a combination of easy to test and, based on experience, likely to reveal the root cause of many issues. This would often include visual inspection and running a product through production test to see if the device still passes or not. If not, that could indicate performance degradation in the field or intermittent product or test issues that were not caught during the first run-through.

Failure analysis goes on to other simple things, including looking for shorts, loose connections and using probing tools to look for unexpected behavior. If a chip is suspected to have a problem, the chip could be replaced with a new chip or reprogrammed to see if the issue persists.

The most complex chip in a device is often the microcontroller or application processor. Unexpected behavior can be the result of several different causes, including but not limited to the following issues:

1. Damage to the chip, either physical or electrical
    - Soldering at wrong temperatures could lead to delamination issues.
    - Tensions in the PCB could crack the package and leads.
    - Submersion in liquids could cause formation of metal whiskers leading to shorts.
    - Improper handling could lead to ESD damage.
2. Changed physical behavior due to wear-out
    - Sustained operation of IOs outside current / voltage limits over time could alter their characteristics.
    - Significant transient voltage overshoot on pads could lead to damage.
    - Excessive writes to the same flash regions could wear out specific flash bit-cells.
    - Accelerated wear-out due to operation at high temperatures.
3. Improper application behavior
    - Incorrect or improperly programmed flash image.
    - Incorrect application configuration or state in flash.
    - Application exposed to unexpected environmental inputs; sensor readings out of range.
    - Application bugs that were not discovered during testing.
4. A problem with the chip itself
    - Chips have a failure rate measured in parts-per-million. Some failures are expected.
    - Excessive failure rates could be the result of a manufacturing issue.
    - Intermittent errors that only occur during certain conditions.
    - Consistent failures, i.e. a function that does not work as advertised.

The above causes can result in a wide range of failure modes or behaviors. Issues (1)-(3) are the most common, but issues of type (4) do occur, and identified problems are reported as chip errata so developers can work around them. The remaining sections in this paper primarily focus on ensuring enough visibility into the device to deal with issues of type (3) – improper application behavior. This also applies to certain issues of type (2), such as characterizing wear-out related issues and uncovering any issues related to the chip itself (3).

Chip Locking

To analyze an application to get to the root cause of an issue, it is often very useful to be able to see what is going on inside the chip: to read the flash contents of the chip to see if the program, configuration information, and state variables are correctly written, and observe the program while it is running. There is, however, a challenge with this approach.

There are many reasons why a device maker does not want to allow anyone to dig into the chip and observe the application. The firmware on the device could be company intellectual property, the device could contain secret information such network keys, end-customer personal information or credentials, or the application could monitor quantities the user would be charged for, such as a water meter monitoring how much water the user is consuming.

For these reasons, a chip is usually locked down when it leaves production to prevent adversaries from being able to gain access to the chip, often including internal company personnel who want to perform failure analysis.

The mechanism to lock down a chip comes down to a clear tradeoff between security, ease of failure analysis and operational complexity. Table 1 shows seven methods of chip locking options and tradeoffs.

- Operational Difficulty describes how difficult the method is to implement by engineering and manufacturing.
- In-Field Risk describes how easy it would be for a hacker to gain access to the device in the field.
- Failure Analysis describes how easy it would be for device maker staff to gain access to the device for failure analysis.

Table 1 includes a new method introduced by Silicon Labs in its new Series 2 portfolio; using asymmetric keys to enable a good combination of security and ability to do failure analysis, which is also known as “Secure Debug.”

Secure Debug provides several benefits. Other locking schemes force the device-maker to make a trade-off between operational complexity, security and the ability to perform efficient failure analysis. The new method makes the process easy by not requiring device-independent secrets to be programmed, has a high level of security, and allows full access to the chip during failure analysis, which is key to determining device problems quickly.

Method	Operational Difficulty	In-Field Risk	Failure Analysis
Don’t lock device	**Easy**. Nothing special to be done	**Significant**. Any hobbyist hacker can break into the device and alter it or steal IP	**Easy**. Full access to device
Remove access to debug pins	**Medium**. Not possible on all devices. Must use e.g. BGA with debug pins close to center	**Medium**. Sophisticated hackers can alter the PCB or remove the package to get to the debug pins	**Hard**. No access to debug pins means no access to chip. Must alter PCB or remove package to get to debug pins
Debug-Erase is a locking method available on some chips where a command run on the debug pins first erases the device, then unlocks it	**Easy**. Typically set with a configuration bit	**High**. Hacker cannot easily get to secrets stored on device, but can erase device and install their own firmware, altering device behavior, and worst-case putting the device back on the store-shelves for an unlucky customer to pick up	**Medium**. Failure analysis can debug the physical chip, but cannot see the state of the program configuration, code or other data, as the device must be erased before access is granted
Total Lockout leaves the debug pins in place but prevents any access to chip internals via debug pins. Irreversible lockout	**Easy**. Typically set with a configuration bit	**Low**. Hacker cannot easily get to secrets stored on device	**Very Hard**. No easy way to access chip

Lock with global code allows debug interface to be unlocked, using the same key for every device	**Easy**	**High**. For higher value assets, the code is very likely to be discovered, and entire fleet of devices is vulnerable	**Easy**. Same code for every device makes it easy to perform failure analysis
Lock with device-unique code	**Medium**. Must program unique code into every device at production and keep codes secure	**Medium**. Code may be intercepted during manufacturing and used later to unlock device in-field. Sophisticated attackers can potentially probe device to get access to code	**Easy**. Must identify device and retrieve code
Lock with asymmetric key – Secure Debug	**Easy**. Same public key is programmed into every device. Private key must be kept safe	**Low**. Unlock key is not stored on device	**Easy**. Identify device and generate unlock token using private key. Execute command against device to invalidate token and re-lock device

Table 1. *Methods of locking down a device and associated trade-offs.*

Secure Debug

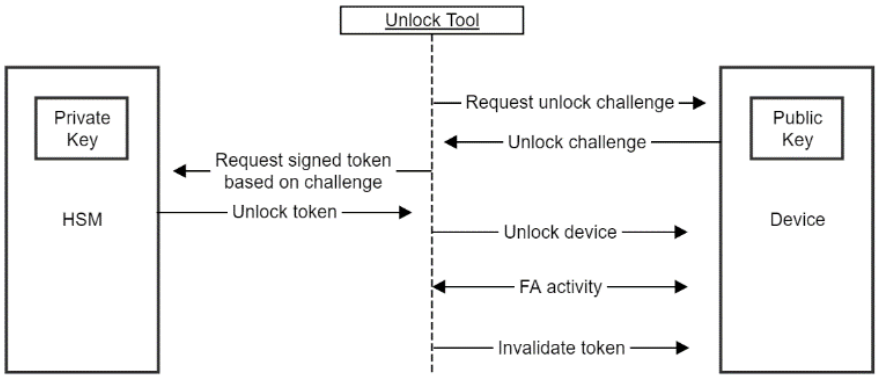
To enable Secure Debug for a series of devices, the device maker creates an asymmetric key-pair. The public key is programmed into every device at manufacturing, and the private key is kept safe and secure in a way deemed sufficient by the device maker. A Hardware Security Module (HSM) might be a good option.

To unlock a locked device, e.g. for failure analysis, the device maker queries the device for a debug unlock challenge, as well as the unique device ID. Using the challenge, a debug unlock token is created and signed with the private part of the debug unlock key pair by the device maker. This debug unlock token can now be used by failure analysis engineers to unlock the part to get the ability to debug and continue with their root cause analysis. An overview of this process is shown below.

Once the failure analysis is completed, a command can be run on the device to invalidate the unlock token, preventing it from ever being used again.

With the Secure Debug approach, the device maker holds the key to the device. If the device maker believes there is something wrong with the chip itself and sends it to failure analysis by the chip vendor, the device maker would have to generate an access token for the chip vendor for the chip vendor to have access to the device. This puts control fully in the hands of the device maker while still providing full flexibility for performing proper failure analysis.

Secure Debug can be a key enabler for secure, quick and efficient failure analysis for situations when the error cannot be found through inspection and the recommendation is to enable it wherever possible.



Secure Debug: Unlock Sequence

Conclusion

Dealing with field failures and other product issues are a natural part of building and selling embedded devices, and by dealing promptly and efficiently with issues that come up, a device maker can significantly improve the customer experience, product success and business outcome. A solution must be chosen that provides full access to devices for failure analysis but does not impose a security risk and does not add operational complexity. Locking a device using an asymmetric key, a technique that Silicon Labs calls Secure Debug, is a method that maintains all the benefits of ensuring secure devices without adding complexity and risk.