Keylabs

https://keylabs.io

Keystone3 Wallet Audit: Final Report

Nov 22, 2023

Executive Summary

The Keystone3 is a hardware wallet with many security firsts for hardware wallets in general and more specifically for audits performed by Keylabs. The Keystone3 is the first wallet we've audited with a point-of-sale grade secure microcontroller. The first wallet we've audited with a fingerprint sensor. The first hardware wallet we've audited that implements Physically Unclonable Function-based keys on the secure elements. The first hardware wallet we've audited with three secure elements. The first hardware wallet we've audited with an effective tamper circuit. More importantly, it's also the first wallet we've audited where the tamper rendered the device unusable, which is the recommended action when the device is tampered. The wallet also featured things like firmware upgrades over SD cards and even an intuitive setup process - which is something we notice during the setup of a wallet for testing. Most importantly, the Keystone3 offers plenty of headroom for further improving security and adding additional security primitives over time in later firmware updates. Overall the Keylabs team was able to identify 1 High severity and two Low severity issues in the firmware and 3 Low severity issues in the hardware.

Hardware Wallet Threat Model

This is an outline of the threat model the Keystone3 was evaluated against. Many of these points are common to all hardware wallets, but some of these points were adjusted to reflect specifics of the Keystone3 wallet.

Adversary Profiles

Remote Attackers: Actors capable of remotely comprising the host or mobile phone that is used to connect to the wallet and leveraging access to it.

Local Attackers: Actors with permanent or temporary access to the wallet. These actors generally have expertise in hardware hacking, knowledge of hardware

vulnerabilities of the wallet and/or vulnerabilities in the current firmware version on the wallet.

Insider threats: Actors with authorized physical and/or remote access to the wallets who may misuse their privileges, for example during device manufacturing or provisioning or by shipping malicious software or firmware.

Threats

Physical Access/Evil Maid Attacks: Temporary or permanent physical access to a user's wallet. Sufficient for a malicious actor to interact with the device.

Supply Chain Attacks: Software, firmware or hardware tampered with during manufacturing, development, distribution, or delivery. Generally in conjunction with Internal threats.

Remote Attacks, Malware and Software Attacks: Malicious code on the host or mobile phone capable of interacting with the hardware wallet.

Man-in-the-Middle (MitM) or Replay Attacks: Intercepting or manipulating the communications between the wallet and the host software as it's being sent over the wire (USB) or via wireless protocols, such as NFC or Bluetooth.

Side-Channel Attacks: Extracting information from unintended channels like power consumption or electromagnetic emissions.

Firmware Vulnerabilities: Exploiting vulnerabilities in the firmware, including the upgrade routines and bootloaders to recover or compromise the wallet and/or seed.

Key Extraction: Techniques to recover the seed, private key or seed phrase at rest. This may include having to brute force the PIN space.

Downgrade attacks: Techniques to downgrade the software an/or firmware to a previous vulnerable version. For hardware wallets this is generally enforced by a bootloader.

Attack Vectors

Malware: Compromised hosts and mobile phones used to interact with the wallet stealing the user's pin and/or preventing transactions or directing them to different addresses.

Evil Maid Attacks: Temporary physical access to the device, sufficient to swap the device, reflash the device, but not necessarily to physically modify the device.

Loss, Theft or Destruction: physically losing access to the hardware wallet and the cryptographic seed by physically losing, having the device stolen or destroyed.

Supply Chain: Compromised hardware introduced during manufacturing or distribution and/or compromised firmware and/or software delivered via updates.

Social Engineering: Tricking the user into entering the pin incorrectly or sending funds to the wrong address.

Authentication Bypass: Insufficient or poorly implemented mitigation of brute forcing or bypassing the authentication scheme entirely.

Firmware Exploitation: Identifying and exploiting vulnerabilities in the firmware allowing for the seed to be recovered.

Insecure Communications: Intercepting data during communication between the wallet and the host as well as between different ICs on the wallet.

Fault Injection and Physical Attacks: Several forms of physical attacks against electronic components can cause them to operate abnormally, potentially bypassing authentication and enabling features that compromise device security, such as debugging.

Secrets in Non-Volatile Memory: Physical attacks against the device often result in the full extraction of the Non-Volatile Memory (NVM) contents which may include the seed.

Secrets in Volatile Memory: It is common to be able to recover the volatile memory contents of devices.

Mitigation Strategies

Strong Physical Security: Wallets should implement tamper detection and should have a case that provides sufficient tamper-evidence for the user to see a wallet that has been tampered with.

Device Verification: Implement mechanisms for users to verify the authenticity of their own wallet, such as nicknames and device pairing. Additionally users should be able to rely on firmware signatures to check the authenticity of their device.

Secure Boot and Firmware: Employ secure boot to verify the firmware prior to execution, regularly update firmware to address vulnerabilities. Since wallets may be stored, for example in a drawer, it's important to offer firmware updates whenever they're plugged in.

Multi-Factor Authentication: Require multiple forms of authentication to access the wallet. All interaction with the wallet should require the user PIN or Passphrase or at least a button press to confirm.

Encryption: Communications between the host or mobile phone and the wallet should be encrypted to prevent malware from MitM the communications. The seed should also be encrypted at rest on the device.

Regular Security Audits: Conduct regular security assessments of changes to the device firmware to ensure that the firmware mitigates all known attacks.

Documentation Recommendations

Create detailed documentation outlining security practices, threat mitigation strategies, and incident response procedures. The threat models should be specific to the particular context and product, and they should be reviewed and updated regularly as the threat landscape evolves.

Architecture Audit

Malware

The main advantage of a hardware wallet is to protect the user from malware infecting their machine or mobile phone. The physical isolation and separate computing environment of a a hardware wallet is generally enough to satisfy the requirement for protection against malware on the host. Malicious code running on the hardware wallet itself can be eliminated with secure update mechanisms and user interaction to trigger the update process. The Keystone3's MH1903 primary microcontroller in conjunction with the secure elements is sufficient for this task, assuming that the software update mechanism is secure and requires confirmation by the user. Additionally, the Keystone3 utilizes multiple secure elements that can also provide additional functionality for verifying firmware origin. Malicious firmware updates can completely undermine the security of the hardware wallet. The hardware wallet can't verify the correctness and lack of vulnerabilities in a firmware update. It can however verify that the hardware update is signed by the manufacturer. Therefore, the supply chain security of the build process for the firmware update is of utmost security importance.

Evil Maid Attack

The Evil Maid Attack is where an unattended wallet is accessed by an malicious actor. In an evil maid attack the malicious actor may swap the device or modify the hardware and/or firmware of the device. In either case, the goal is usually to recover the PIN of the device. To prevent this attack, the device must authenticate itself to the user and detect and report tampering. The Keystone3 wallet has the hardware to support these features, but it must display to the user some unique pattern, serial number, or similar such that the user can identify their own wallet. The device must always report the tamper status prior to the user entering the PIN. The software and physical tampering must be evaluated to determine mitigation against this threat.

Loss, Theft or Destruction

The user must be safe from the hardware wallet being lost, stolen or destroyed. The wallet must provide a recovery mechanism, ideally from an offline backup. Typically this is done by recording the seed phrase generated by the hardware wallet during the wallet creation that can be recorded, backed up and subsequently imported. As the wallet is a physical device and that can be lost, stolen or destroyed there must be a robust recovery mechanism. Specifically for Keystone3 the fingerprint sensor should be used to protect any secret

information such as the private keys or seed phrases, but not necessarily used to derive the wallet seed as this may be reproducible on another device. This is primarily a firmware feature and will be evaluated during the assessment.

Supply Chain Attacks

In a supply chain attack, a malicious actor is generally able to get physical access to the device prior to the user receiving and setting up the device. The most common attack is pre-initializing the wallet with a seed phrase that is known to the attacker. Because the seed phrase is known to the attacker, the attacker can steal any funds that are stored on any of the addresses derived from that seed phrase. To an unsuspecting user, it appears the device is ready for use and they are not aware of the attack.

More sophisticated supply chain attacks may physically modify components on the device, replacing components entirely with malicious ones. For example, if the fingerprint sensor is replaced with a malicious one, it may be possible for an attacker to bypass the fingerprint entirely. It may also be possible to replace the microcontroller with a less secure and/or counterfeit one. In such a case it may be possible for the attacker to utilize a backdoor to retrieve private data from the device. It is therefore important for the user to understand they have authentic hardware.

Another kind of supply chain attack is a denial of service on some of the security ICs responsible for storing the cryptographic state. Such ICs often use internal counters to track pin attempts and also provide functions such as generating entropy. Such attacks can increase internal counter values to the point where the device can be rendered non-functional. Careful consideration should be taken when implementing such mechanisms as part of device production, device setup as well as whenever the device is reset. The hardware wallet should clearly define which operations are permitted and what can be verified by the user.

One of the biggest goals in providing a secure supply chain is verifiability of the manufactured device. A manufactured device should be tracked through manufacturing and recorded as verifiably working. This will prevent malfunctioning devices from making it to users and provide an audit trail of how many units are actually manufactured. Additionally it will allow manufacturers to only manufacture the exact amount of units they plan on manufacturing and supporting. This will also make it difficult to produce any unauthorized or cloned devices as this can be detected when verifying the device.

Lastly, the Keystone3 bill of materials is quite complex including SRAM ICs, Flash ICs, multiple Secure Elements etc. The bill of materials should be evaluated in terms of which components can end up being a bottleneck during production. As the most recent

world-wide chip shortage has shown, many chips had become unavailable, forcing manufacturers to make ad-hoc changes to their Bill of Materials and PCBs and necessitating multiple product revisions with differences in hardware and software.

User Approval via Trusted Display

Without a trusted display, a hardware wallet is susceptible to convincing a user to erroneously sign a malicious transaction. It should be noted that when dealing with complex smart contract calls in particular, there might not be enough information displayed on the device for the user to fully visually validate the transaction. Even with a hardware wallet, there is the possibility of losing a NFT or authorizing a smart contract with more privileges than intended. As interacting with other wallets and extensions is a key feature of the Keystone3, this should be carefully considered.

In simpler cases where funds are transferred between two addresses. Here users just need to verify the network and the address of the recipient of the transaction. However, the display user experience is important to aid the user in properly performing the verification. If the display and interaction is cumbersome, it will reduce the overall security of the device.

Authentication Bypass

Authentication bypass on a hardware wallet typically involves cracking or brute forcing the PIN. The Keystone3 also includes a fingerprint scanner and it has not yet been evaluated if the authentication logic is a PIN and fingerprint or a PIN or a fingerprint. In any case, bypassing the authentication to the wallet will allow access to the private keys and/or seed phrase.

The issue with hardware wallets, and embedded devices more generally, is that they are less powerful than the attacker CPU. Therefore, common cloud and desktop security measures of password based key derivation functions generally don't apply to embedded systems, as they can be brute forced on a more powerful machine. Instead they should rely on the number of failed attempts.

There are several methods to prevent or thwart repeated attempts and this will be evaluated on the Keystone3. Most importantly both secure elements provide primitives for building this kind of authentication. Additionally, how the wallet maintains the state of attempts is critical to the security as well.

Bus/Protocol Sniffing

Bus sniffing is the technique of recovering plaintext data on internal buses of the device. There are numerous peripherals on the Keystone3 and it has not yet been evaluated if

security critical data is transmitted in plaintext over these buses, but this remains a threat to most hardware wallets with numerous components. Especially of interest is the interaction with the secure elements as most secure elements can support encrypted interaction, but often, default to plaintext secrets.

Bus sniffing can also be mitigated with a tamper response system, which it appears the Keystone3 has the hardware to support but must be tested. Of interest, is how permanent the tamper detection is and if it can be reset or not which might allow for interesting attack vectors.

Similar to transmissions in plaintext on buses, data can be transmitted over the air in plaintext utilizing protocols such as Bluetooth Low Energy. In general the same level of care should be applied to any wireless transmissions.

Fault Injection

Fault injection is the technique of introducing faults into the normal operation of the system with the intention of bypassing or temporarily (transient faults) disabling security features. Of note, are the numerous Electromagnetic Fault Injection (EMFI) and Voltage Glitching attacks that have been demonstrated on hardware wallets. The attack can either focus on the application code or the embedded immutable ROM. Attacking the ROM or BootROM generally means the code cannot be patched in the field. The Keystone3 selected a microcontroller that appears to be security focused with some marketing language indicating defense against such attacks, however it is not clear to what standard it has been tested.

Protection of secrets at rest

In a stateful design such as the Keystone3, the hardware wallet is expected to protect the seed as it is stored in the device. If the wallet is lost or stolen, an attacker will attempt to recover the seed because if the seed is recovered, then funds can be transferred with a software wallet.

Besides the seed, there may be other secrets on the device such as authentication parameters. As there are at least 5 chips with some memory (2 secure elements, 2 NOR flashes, and 1 MCU) there is a significant area of potentially exposed memory. How these components interact to protect the secrets must be evaluated to ensure there is no easily retrievable sensitive data.

Protection of secrets in use

Similarly to the at rest scenario, there are two volatile memories with the external SRAM and the MCU's SRAM. Of particular importance is how the seed is managed in the MCU itself. The device has both USB and Bluetooth interfaces. USB is especially known to have complicated software interaction. If the seed is accessible in the same memory space as the USB stack, then an USB exploit could simply copy the seed out over the USB interface for example.

Therefore, internal software protection measures on the MCU are critical to ensure memory separation of the seed and other software, which must be evaluated on the Keystone3.

Firmware Audit

The fingerprint sensor code runs on the Maxim MAX32520 secure element/secure microcontroller. However, this code was not audited as part of the firmware/software audit and was explicitly left out of scope and will be re-evaluated at a later date. Keylabs was provided adequate mitigations by the Keystone team for all the findings outlined in this part of the report.

Tamper Response Inadequate (Severity: High)

When a tamper is detected the tamper response is to call TamperEraseInfo in anti_tamper.c. This function writes tamper detection information in the two external (to the MCU) secure elements. However, it does not appear to record any indication to internal memory. Nor does it verify that the tamper data is successfully written apart from physical layer commands.

This is a weakness because it is known to the system that a tamper occurred, but the attacker can stall the communications to the external secure elements, for example by driving the lines low or high, and the tamper event will not be recorded. Possibly then, the attacker can modify the tamper detection signal, reboot, and not have access to the system. This has not been dynamically tested, only observed through static analysis.

To perform the encrypted write to the ATECC608, after a tamper is detected the following occurs:

1. The MCU reads 20 bytes of random from the trng. Internal trng may stall, which could cause a delay.
2. Wake up the ATECC608.
3. The MCU and the ATECC608 perform a checkMac operation. The command, once received over i2c is at minimum 5ms, max of 13ms (according to the ATECC508 public datasheet).
4. The MCU must derive the encryption key for the ATECC608 (which is sha256 based) then send the encrypted write command which is at minimum 32 bytes and that commands takes 7ms to 26ms to complete on the ATECC608.

A conservative estimate is that the operation should take anywhere from 50ms - 200ms of time. If the battery is disabled (or the i2c bus shorted) before this entire operation completes, a tamper event will not be recorded.

For example, to determine if a tamper occurred, here is the logic. Note that all of the page data must return the tamper mark. Because the return code of the encrypted read is not checked (which is another problem), this tamper logic can be aborted by disrupting the i2c bus convincing the device that a tamper has not occurred. Obviously, if the tamper mark was written but the tamper read back failed, it's not clear the state of the device but it's concerning that the device can proceed in this unknown state.

```
/// @brief Check the device whether be tampered.
/// @return True if the device has been tampered.
bool Tampered(void)
{
    uint8_t pageData[32];
    static bool tampered = false;
    static bool checked = false;
    if (checked) {
        return tampered;
    }
    Atecc608bEncryptRead(15, 0, pageData);
    PrintArray("pageData", pageData, 32);
    for (uint32_t i = 0; i < 32; i++) {
        if (pageData[i] != TAMPER_MARK) {
            printf("pageData[%d]=%d\n", i, pageData[i]);
            tampered = false;
            checked = true;
            return tampered;
        }
    }
    tampered = true;
    checked = true;
    return tampered;
}
```

In conclusion, the tamper response is not adequate for a prepared attacker if this sequence is known. This can be fixed by _first_ writing to an internal non volatile memory, then writing to external memory, then possibly updating internal memory.

Physical Access may result in Entropy Injection (Severity: Low)

On the Keystone3, randomness from several sources is combined to generate entropy. An attacker with physical access to the device may be able to inject low entropy, downgrading the overall entropy of the GenerateEntropy function.

```
/// @brief Generate 32 byte entropy from SE and mcu TRNG.
/// @param[out] entropy
/// @param[in] entropyLen
/// @param[in] password Password hash is part of the entropy sources.
/// @return err code.
int32_t GenerateEntropy(uint8_t *entropy, uint8_t entropyLen, const char
*password)
{
    uint8_t readBuffer[ENTROPY_MAX_LEN], tempEntropy[ENTROPY_MAX_LEN];
    int32_t ret, i;

    do {
        HashWithSalt(tempEntropy, (uint8_t *)password, strlen(password),
"generate entropy");
        TrngGet(readBuffer, ENTROPY_MAX_LEN);
        KEYSTORE_PRINT_ARRAY("trng", readBuffer, ENTROPY_MAX_LEN);
        for (i = 0; i < ENTROPY_MAX_LEN; i++) {
            tempEntropy[i] ^= readBuffer[i];
        }
        KEYSTORE_PRINT_ARRAY("tempEntropy", tempEntropy, ENTROPY_MAX_LEN);
        ret = DS28S60_GetRng(readBuffer, ENTROPY_MAX_LEN);
        CHECK_ERRCODE_BREAK("get ds28s60 trng", ret);
        KEYSTORE_PRINT_ARRAY("ds28s60 rng", readBuffer, ENTROPY_MAX_LEN);
        for (i = 0; i < ENTROPY_MAX_LEN; i++) {
            tempEntropy[i] ^= readBuffer[i];
        }
        KEYSTORE_PRINT_ARRAY("tempEntropy", tempEntropy, ENTROPY_MAX_LEN);
        ret = Atecc608bGetRng(readBuffer, ENTROPY_MAX_LEN);
        CHECK_ERRCODE_BREAK("get 608b trng", ret);
        KEYSTORE_PRINT_ARRAY("608b rng", readBuffer, ENTROPY_MAX_LEN);
        for (i = 0; i < ENTROPY_MAX_LEN; i++) {
            tempEntropy[i] ^= readBuffer[i];
        }
        KEYSTORE_PRINT_ARRAY("tempEntropy", tempEntropy, ENTROPY_MAX_LEN);
        memcpy(entropy, tempEntropy, entropyLen);
    } while (0);
    CLEAR_ARRAY(readBuffer);
    CLEAR_ARRAY(tempEntropy);
    return ret;
}
```

There are two issues with this code. The first is the use of the XOR operation in general and the second is that the ATECC608 only provides an unauthenticated getrandom command. The problem is that if any of the devices fails to produce actual random

data an XOR of the results will only exacerbate the low entropy. For example, XOR with all 0's doesn't change the value at all, XOR with 1 flips the value, and the commutative property of xor implies you can XOR with a previous operand to reveal the previous result.

Since the ATECC608 is the last source of entropy and the ATECC608 getrandom command is not authenticated, but only checksummed, this data can be injected by a malicious attacker. For example, the attacker can simply return all 0's or all 1's to decouple the entropy from the ATECC608 completely, while still generating a response that will appear valid to the microcontroller. Note, for this attack to succeed, it must happen when the entropy is being generated, which makes the attack far less feasible in practice and requiring either supply or physical access to the device. Moreover, this only affects part of the entropy generation.

However, it is still the case that a silent failure of the entropy generation in any of the three random generators can eliminate them as a source of entropy completely, but especially in the case of the ATECC608. The resulting entropy should thus only be used in conjunction with other unique pseudo-random sources, such as device unique IDs. In practice, it's better to consider that any hardware random number generator source may fail subtly, generating insufficient entropy and instead combine several entropy sources. Additionally, to eliminate low entropy results from a hardware entropy source a key derivation can be applied to the results, where a KDF function is chained to each new input from the hardware or another model based on cryptographic secure pseudo random generators.

Security Functions Optimized Out (Severity: Low)

There are numerous cases of code such as the following, which occurs in bip39.c lines 129, but also bip39.c lines 202 where before a return statement, a buffer is memset to 0. The issue with this code is it may be entirely optimized out by the compiler as logically, it serves no purpose.

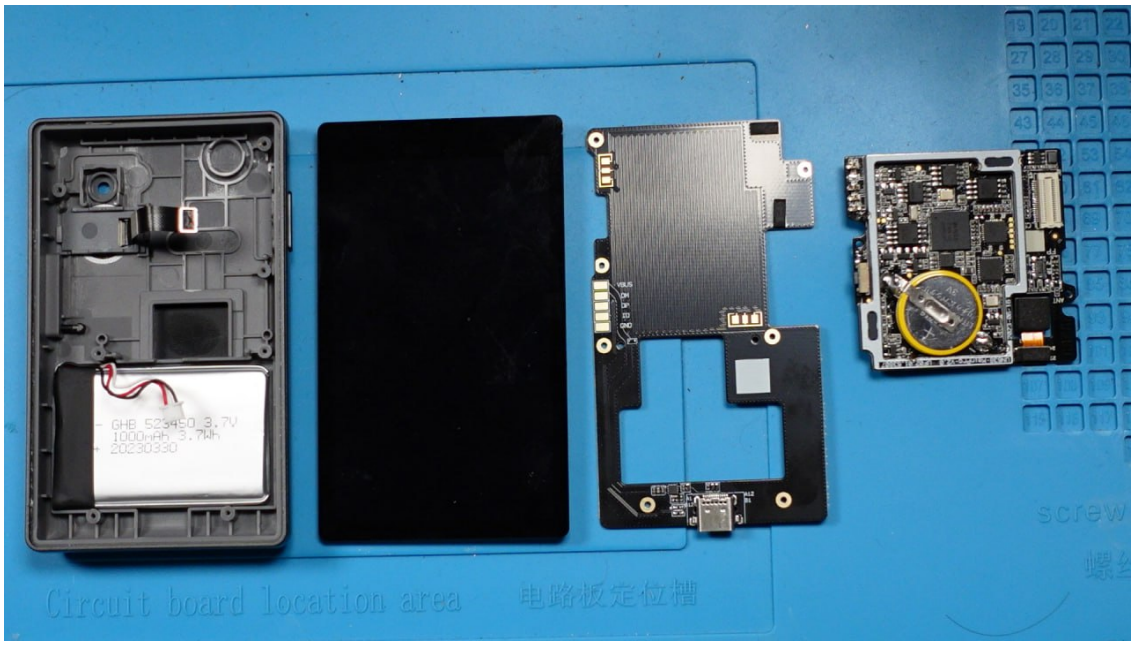
```
*output = mnemonic_from_bytes(w, tmp_bytes, bytes_len + 1);
 memset(tmp_bytes, 0, sizeof(tmp_bytes));
 return *output ? SUCCESS_CODE : -1;
```

The presence of this can be confirmed by inspecting the object code, which is not available to us at the moment, however the preferred solution is to write a version of memset that is guaranteed to work. This is typically called memset_s or similar and the trick is to loop through the buffer with a keyword volatile index, which prevents the compiler from optimizing what is otherwise known as a useless (to the compiler) operation.

Additionally, there is a MACRO called CLEAR_ARRY in user_utils.h and CLEAR_OBJECT both of which just called memset to 0. These appear to be used in security critical functions like TamperEraseInfo in anti_tamper.c which is clearing memory on a security event however, the same doubts about the actual performance of the zero remain.

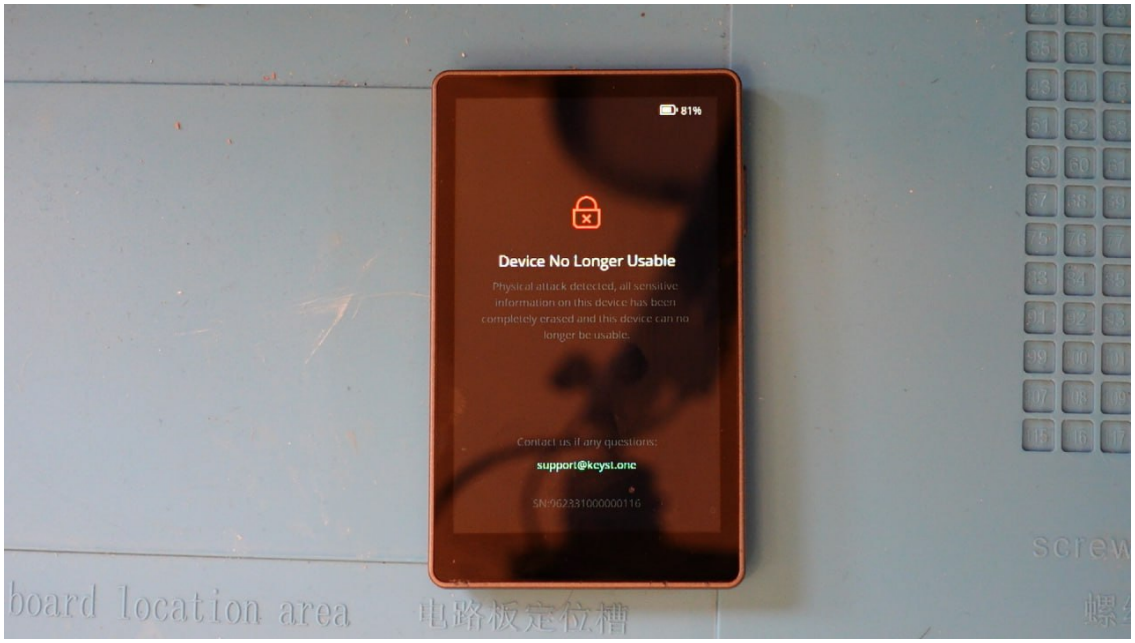
Hardware Audit

As part of the hardware audit the overall hardware design of the Keystone3 was considered and carefully evaluated. It is worth, noting that the use of a total of three secure elements, one of which executes some of the security code in conjunction with the primary MCU, which has relevant security features, such as OTP and tamper detection. Additionally, the physical design and layout of the PCB, the design of the case including a protective mesh for the circuit, a standby coin cell battery to ensure protection even when the primary battery is discharged. All of these design decisions mitigated many common attack vectors for which Keylabs tests as part of the hardware wallet audit.



From left to right, Keystone3 case, screen, mesh PCB and main PCB.

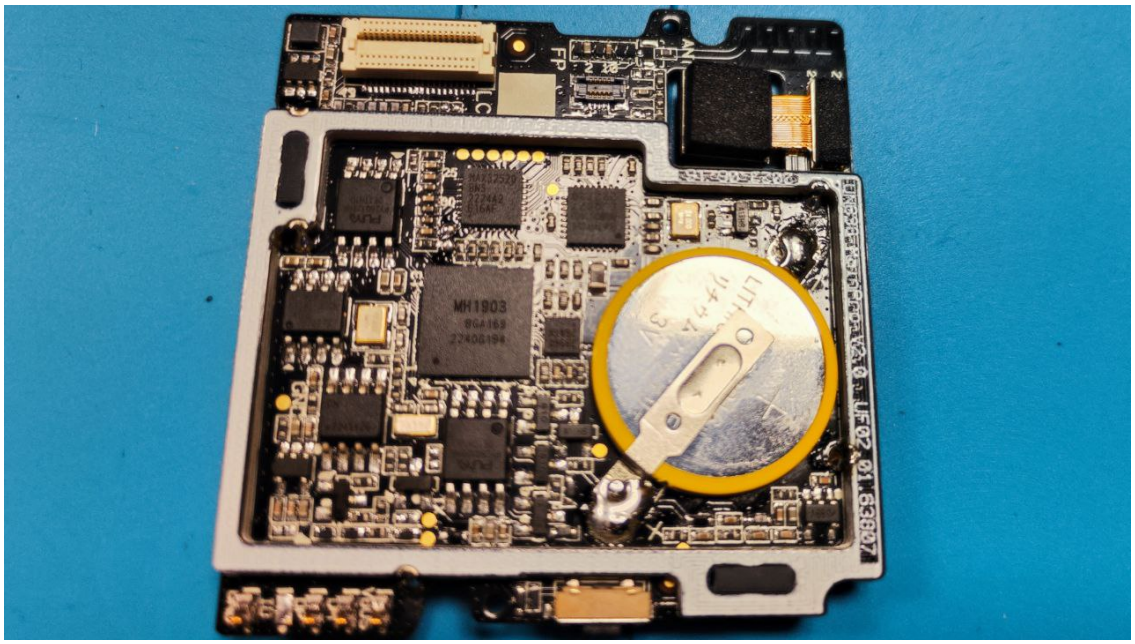
The construction of the Keystone3 requires the protective mesh PCB to be removed to access all of the sensitive signals and components. Removing the protective mesh PCB triggers a tamper event on the device. Keystone3 is the first wallet tested by Keylabs that utilizes a fully functional tamper circuit that renders the device non-usable after a tamper event and for which the team was not able to discover a trivial bypass. Because the tamper circuit is powered by the coincell battery it is capable of detecting a tamper event even if the primary lithium-polymer battery is disconnected or drained.



Keystone3 device rendered no longer usable after a tamper event was detected.

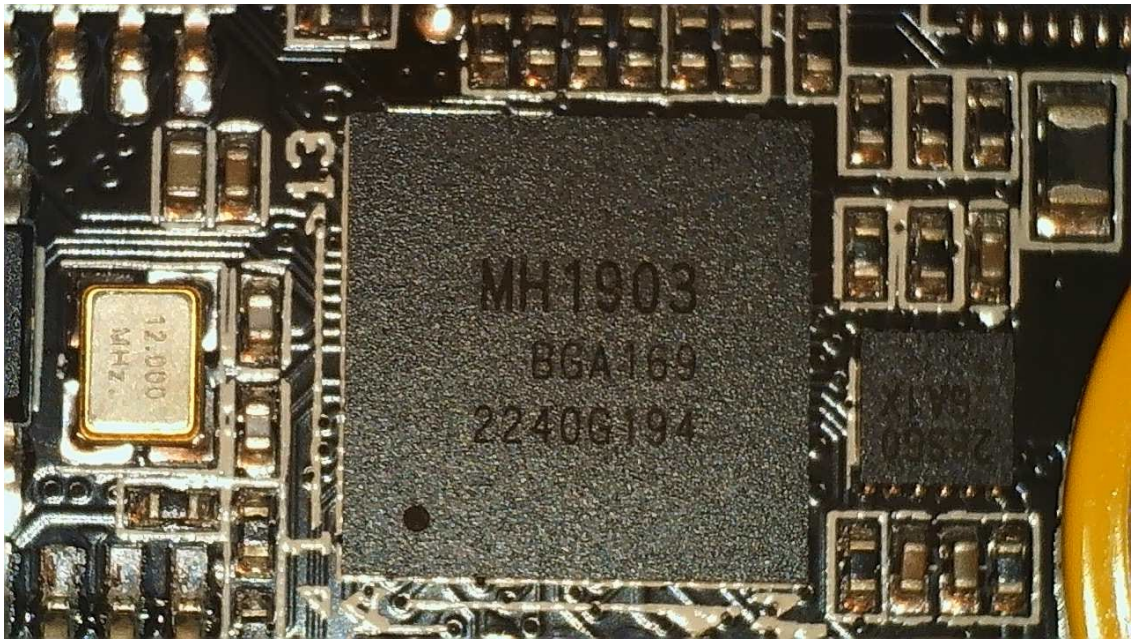
Security Component Analysis

All sensitive components on the Keystone3 are located underneath the mesh PCB. These include the MCU, the three secure elements as well as external memories for the various components. As these memories are not readily accessible and the tamper response was deemed to be sufficient, the security of these memories was not further evaluated.



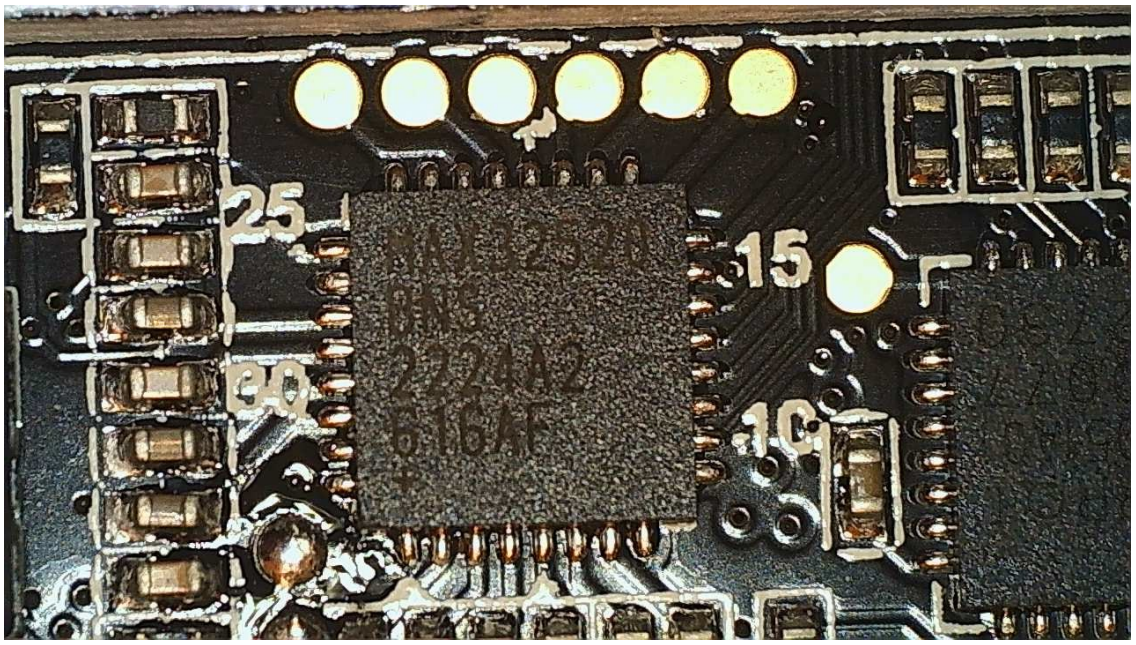
Keystone3 Primary PCB with coin cell backup battery visible

The primary microcontroller is the Megahunt MH1903. This is a point-of-sale grade microcontroller with several tamper inputs available for detecting device tampering.



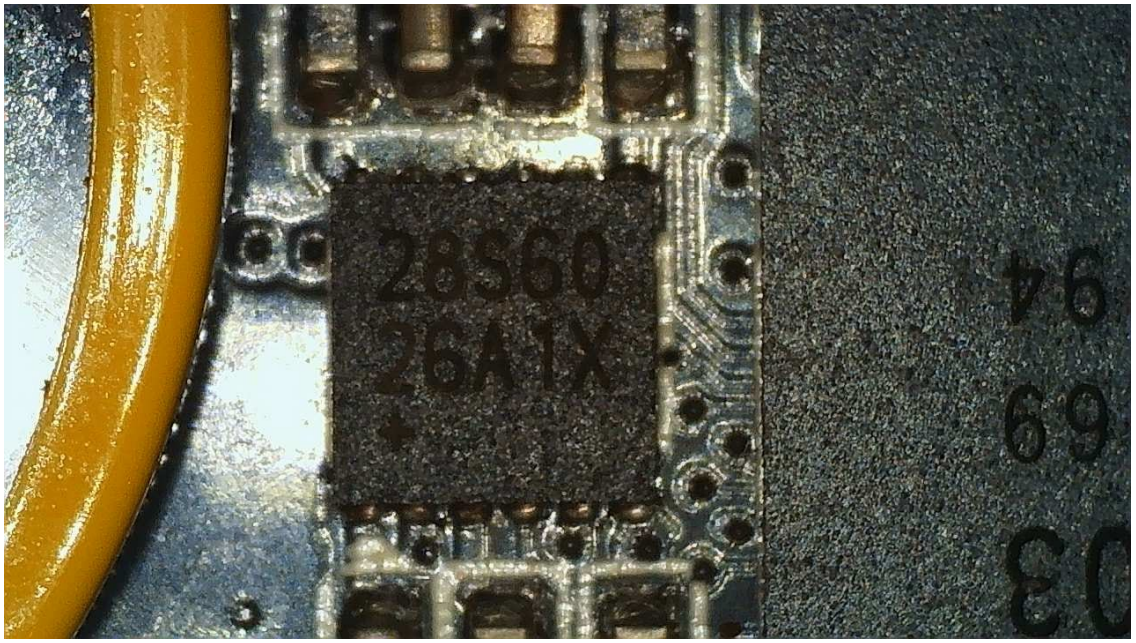
Primary Microcontroller: Megahunt MH1903

The fingerprint verification code is executed on the Maxim MAX32520 secure element ARM Cortex-M4. Notably this chip utilizes Maxim's ChipDNA technology, which implements Physically Unclonable Functions for keys to protect the internal memories of the device from physical attacks.



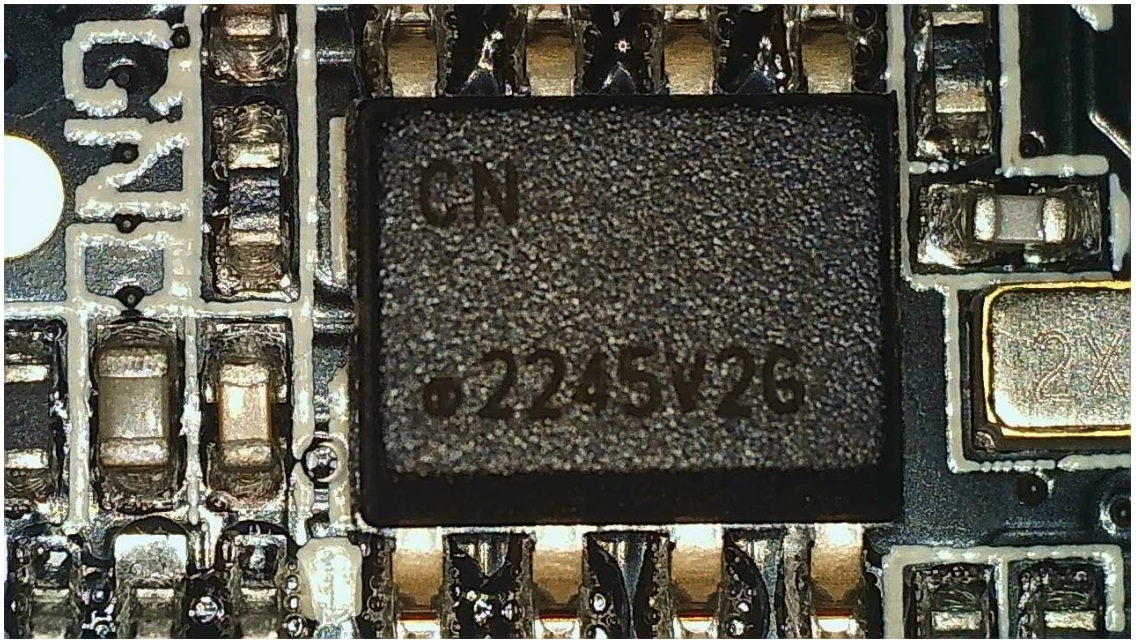
Maxim MAX32520 secure ARM Cortex-M4 with PUF

Next, there are two secure elements. First, the Maxim DS28S60Q, which also uses Maxim ChipDNA.



Maxim DS28S60Q secure co-processor/secure element with PUF

Finally a Microchip ATECC608B secure element. This chip adds additional features that are unavailable in the other secure elements, most notably a hardware monotonic counter that is incremented when used.



Microchip ATECC608B secure co-processor/secure element

Hardware Audit Findings and Recommendations

All the findings in this section are classified as Low and are not critical to fix before the product launch. Keylabs encourages Keystone to consider fixing these issues in future device batches and device revisions.

Tamper Response Unaffected by Screen Removal (Severity: Low)

The wallet itself implements a security boundary with a PCB mesh covering sensitive components. Removing the screen does not trigger a tamper response as the screen is outside the security boundary. Screen removal is relatively simple with heat and scalpel. Moreover, the gap between the case edge and the screen makes it possible to remove the screen without leaving obvious marks on the display or the surrounding case. As opening the device by removing the screen does not bypass the security boundary, i.e. the mesh still needs to be removed to access sensitive signals, this finding is of Low severity. Keystone should consider adding a better glue or seal around the display that would cause the display to break as part of being removed. Moreover a future hardware revision could integrate a light sensor to detect this kind of tamper event.

Tamper Response Detected by Static Logic Level Signal (Severity: Low)

The tamper circuitry on the Keystone3 is implemented as a static logic level signal on a GPIO pin. This means that an attacker may be able to drive this signal by contacting one of the traces carrying this signal by drilling either through the security boundary or directly exposing and connecting to the tamper GPIO BGA pad of the microcontroller. Subsequently the tamper circuitry could be completely disabled. As this attack would not immediately compromise the device and would require a high degree of accuracy to execute, this finding is of Low Severity. Keystone should consider implementing an active tamper signal instead of a logic high or logic low signal in future revisions to further improve the tamper response.

No Tamper Response for Case (Severity: Low)

The wallet itself implements a security boundary with a PCB mesh covering sensitive components. The PCB with sensitive components and the protective mesh form effectively a “sandwich” in which sensitive surface mounted components are in between the PCB and the protective mesh PCB. This however does not include the case and in particular the back, which includes several components and test points. Similarly to the screen an event in which the case is opened or removed from the

backside should ideally trigger a tamper response. This could in theory be implemented with light sensors in a later revision. As opening the back of the case does not necessarily result in compromising the device and the security boundary is still in tact, i.e. the mesh still needs to be removed to access sensitive signals, this finding is of Low severity.

Test Points on Back of the PCB (Severity: Informational)

The way the wallet implements the security boundary makes it theoretically possible to remove the device PCB without triggering a tamper response. Because several test points remain accessible on the backside of the device, it should be verified that there are no sensitive test points on this side of the PCB. The primary programming test points are protected by the mesh and are within the security boundary.

Mitigations and Retest

Tamper Response Inadequate (Severity: High)

Fixed, Verified.

Physical Access may result in Entropy Injection (Severity: Low)

Fixed, verified.

Security Functions Optimized Out (Severity: Low)

Fixed, verified.

Tamper Response Unaffected by Screen Removal (Severity: Low)

No fix needed. Low severity issue.

Tamper Response Detected by Static Logic Level Signal (Severity: Low)

No fix needed. Low severity issue.

No Tamper Response for Case (Severity: Low)

No fix needed. Low severity issue.

Test Points on Back of the PCB (Severity: Informational)

No fix needed. Informational only.