Better Keep Cash in Your Boots - Hardware Wallets Are the New Single Point of Failure

Adrian Dabrowski
University of California, Irvine
United States
a.dabrowski@uci.edu

Katharina Pfeffer
SBA Research
Austria
kpfeffer@sba-research.org

Markus Reichel
Vienna University of Technology
Austria
mx.markus.rei@gmx.net

Alexandra Mai
SBA Research
Austria
amai@sba-research.org

Edgar R. Weippl
University of Vienna
Austria
edgar.weippl@univie.ac.at

Michael Franz
University of California, Irvine
United States
franz@uci.edu

ABSTRACT

Hardware wallets are currently considered the most secure way to manage cryptocurrency keys and sign transactions. However, previous publications show that such tokens can be replaced or manipulated in a number of hard-to-detect ways pre- or post-delivery to the user and that implemented (remote) attestation and authenticity checks fail their purpose for multiple reasons.

We analyzed the architecture of current products by examining their initialization procedure and attestation methods. Unlike previous publications, we found that tightened attestation and communications encryption will not solve the fundamental architectural flaws sustainably. We conclude that the architecture of current-generation cryptocurrency hardware wallets missed the opportunity for a resilient design by copying the PC's wallet architecture and thus merely shifting the single point of trust from the PC to the hardware wallet.

We advocate a mutually verified architecture through changes to BIP32/BIP44 wallet architectures to incorporate collaborative signatures and key generation. This way, neither a compromised wallet nor a compromised PC can meaningfully manipulate keys or transactions.

CCS CONCEPTS

• **Security and privacy → Hardware-based security protocols**; *Key management*.

KEYWORDS

cryptocurrencies; hardware wallets; collaborative protocols; decentralized finance

ACM Reference Format:
Adrian Dabrowski, Katharina Pfeffer, Markus Reichel, Alexandra Mai, Edgar R. Weippl, and Michael Franz. 2021. Better Keep Cash in Your Boots - Hardware Wallets Are the New Single Point of Failure. In *Proceedings of the 2021 ACM CCS Workshop on Decentralized Finance and Security (DeFi '21), November 19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3464967.3488588

DeFi '21, November 19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8540-4/21/11.
https://doi.org/10.1145/3464967.3488588

1 INTRODUCTION

Cryptocurrencies typically organize the ownership of funds using public and private key cryptography (or a derivate thereof). A cryptocurrency wallet software is managing those private keys and submitting signed transactions to the appropriate blockchain. Based on many incidents and the rampant spread of malware and botnets, general-purpose ("personal") computers (PC) and operating systems are increasingly suffering from deprivation of trust. Detachable dedicated hardware tokens promise a hardened single-purpose solution to keep private keys (or seeds thereof) in tamper-resistant storage safe from online or other theft in tamper-resistant storage. While the specialized token still needs the general-purpose PC to post the transaction, the assumption is that a compromised computer can no longer leak private keys nor alter transactions.

However, that unconditional shift of trust might be unjustified. Many current-generation hardware wallets do not operate under a fail-safe, mutually verifying architecture. Instead, our research suggests, they merely outsourced trust to the external hardware, which effectively only leads to a new single point of failure. Consequently, they switch the PC-related security risk profile for a hardware token one. Hence, they miss the opportunity to limit risks to an intersection of all involved mutually verifying components (i.e., the computer, the wallet software, and the hardware wallet token).

This paper analyzes the architecture and protocols of hardware wallets and discusses the shortcomings of the current generation of hardware wallets' architecture. Since wallets follow BIP32 [31] or BIP44 [23] standards, manufacturers have only a limited design space. After a protocol analysis, we created a proof-of-concept attack for demonstration purposes (see Appendix). Our attack's scenario builds upon Trezor's report [26] on the discovery of unauthorized Trezor One clones in the wild and stealthy implants in Ledger clones [1, 13]. For practical purposes, these clones and implants are indistinguishable from legitimate devices for users. Our proof-of-concept is a USB-clone (c.f. supply chain attacks [2]) of the widespread Trezor One hardware wallet, based on collected and replayed USB traces. It passes the setup, initialization, and genuineness test process of the original Trezor One client software. The clone will present attacker-controlled predetermined keys, giving a third party the ability to siphon off all funds transferred into that

wallet. The attack is surprisingly easy to perform and therefore showcases—along with previously reported vulnerabilities—the main points of our architectural critique.

Our contributions are:

- a **security analysis** of the current hardware wallet architecture (see Section 6).
- suggestions on how to improve the architecture's security and **remove a single point of trust** (see Section 7) with algorithmic trust based on mutually verifyable computations.
- comparison of algorithmic trust with other integrity-insuring techniques (i.e., attestation), see Section 7.2.

2 BACKGROUND

The background section focuses on Bitcoin as the predominant cryptocurrency, where most other currencies either borrowed basic concepts from, or are directly built upon them. Notable differences are pointed out explicitly.

2.1 Addresses and Transactions

On an abstract level, a typical blockchain (e.g., Bitcoin or Ethereum) is an account-based system where a transaction can withdraw funds (or post other changes) based on proving the possession of an eligible private key (via a signature). An *account*[1] consists of a public key (often transformed into a public address, e.g., via hashing) and a private key only known to that pair's creator. In the most simple form, pairs are generated based on random numbers. There is no explicit act of creating an address. Using an address as an output of a transaction, implicitly creates it (UTXO, Unspent Transaction Output). Opportunistic collision avoidance relies on the vastness of the address space. An address can be single-use (i.e., hold an amount for a certain period and then no more) or have changing amounts associated with it.

In Bitcoin, transactions have multiple inputs and outputs but have to use all funds available at an address. "Change" is given by directing one output to capture the remainder of the transaction and transferring it back to the original address or a new address of the same owner. Bitcoin never saves the actual amount associated with an address on the blockchain but keeps a public ledger of all transactions to recalculate the current amount when needed. This construct prevents many attacks such as double-spending by allowing every node to track all transfers made. However, this potentially leads to an explosion of addresses.

In Bitcoin, a scripting language governs access rights to funds. However, many clients and miners support only a standard set of predefined arrangements. In the most common *Pay-to-Pubkey-Hash* (P2PKH) transaction, a transaction draws funds by presenting the public key and a signature of the whole transaction with the private key for each input to unlock it. Smart contracts and scripting languages offer more possibilities to unlock funds but are out of scope for this paper.

2.2 Software Wallets

In order to simplify address and key management, most wallets are built on the BIP32 [18, 30, 31] hierarchical deterministic (HD) wallet concept using a (nested) child key derivation (CKD) function with a numerical index to create addresses on demand out of one secret master seed s. Thus, only the master seed s or the derived master key pair mk_p,mk_s (public key and private/secret key) needs to be stored encrypted (and backed up), simplifying key management. Child keys of child keys lead to a nested tree-like wallet structure where the leaf nodes are Bitcoin addresses (or their key pairs) with unique paths describing each key pair, e.g., *mk/0/2* in path-like BIP32 notation is $CKD\big(CKD(mk, 0), 2\big)$. The public key part of so-called *non-hardened keys* is verifiable (and producible) if at least one public key from higher tiers within the path is known. In contrast, so-called *hardened keys* lack that verification, but offer more privacy by not being linkable to any parent keys. In Bitcoin with BIP32, the CKD function is implemented as HMAC-SHA512 and the signature as Elliptic Curve Digital Signature Algorithm (ECDSA) respectively. The PubKeyHash is a series of RIPEMD160 and SHA256 hashes. Additionally, a full ASCII-printable Bitcoin address is Base58-encoded, includes a network prefix and a checksum. BIP44 [23] extends the HD wallet system to multiple blockchains (e.g., Ethereum). Figure 1 illustrates a P2PKH transaction from key *mk/i/j/k* to *mk/0/2*.

2.3 Transactions with Hardware wallets

While a software wallet stores the seed or master key pair encrypted, many users do not trust their Internet-connected general-purpose computer to protect them sufficiently from data theft and malware, including snooped passwords and falsified transactions or data.

A hardware wallet is typically an external hardware device promising hardened high-security storage of the key material ("Wallet Storage" in Figure 1). Since the key material is never meant to leave the device, all cryptographic core functionalities ("Wallet Functions" in Figure 1) are also moved from the client software to the hardware wallet device, i.e., a part of the wallet client software is outsourced to a dedicated device, but no architectural change is made (Figure 2). For certain operations, hardware wallets may require a passphrase or PIN entered on the PC client or the device itself or a confirmation on the device (e.g., a push button).

In this architecture, initializing and setting up a new bitcoin hardware wallet can happen in two ways: The PC client creates a seed and transfers it to the hardware wallet -or- the hardware wallet creates the random seed, optionally providing a one-time copy for backup purposes.

Transactions (or necessary data for it) typically have to pass between the hardware wallet and the PC client multiple times. For example, not all public keys might be available to the client software while assembling the transaction (Figure 1 provides an overview). The final two steps are signing the transaction using the private keys of all inputs by the hardware wallet (modulo possible external signatures) and posting the complete transaction by the PC client to the Bitcoin peer-to-peer network. This back-and-forth gives the two partners limited possibilities to verify each other. However, the PC client might not have the opportunity to verify addresses based on hardened keys or where the public key chain is unavailable.

2.4 Authenticity Checks and Attestation

Hardware wallets may implement various authenticity checks in order to detect supply-chain tampering of software and hardware [2,

[1] Herein used as an analogy to a bank account.

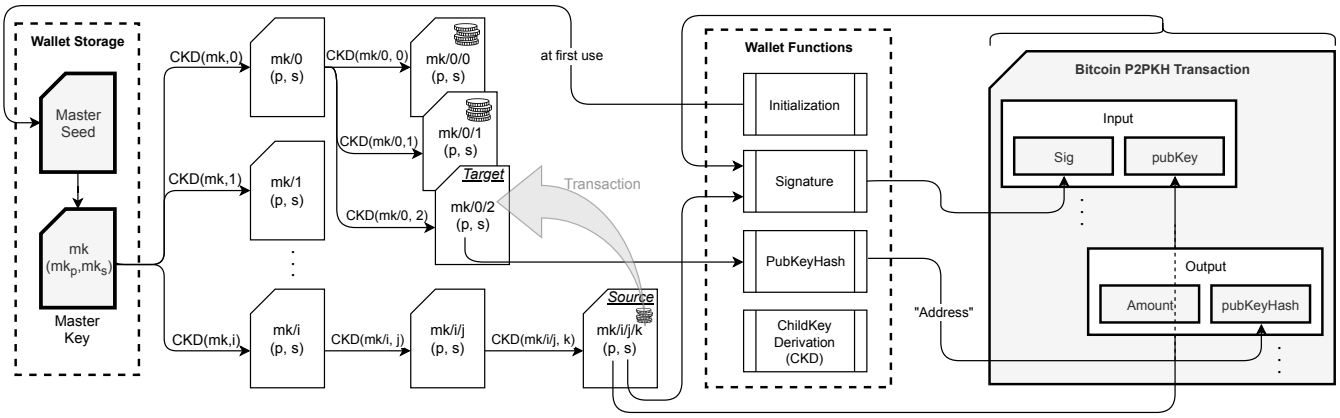

Figure 1: Simplified interplay between wallet and a Bitcoin P2PKH transaction. To simplify the figure, this example transaction transfers coins on the same wallet to a different address.

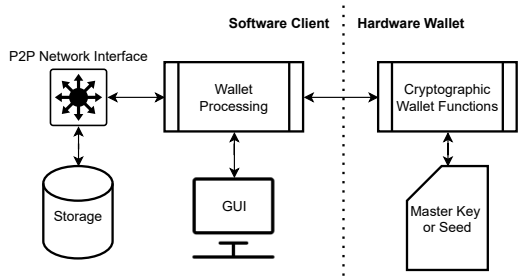

Figure 2: Hardware wallet architecture. Key and cryptographic management is moved to a dedicated hardware, while GUI and network remains on the PC software side.

6]. Pfeffer et al. [24] conducted a market review of the currently implemented authenticity checks of hardware security tokens, including hardware wallets from Ledger, Trezor, and Keepkey.

Implemented *software authenticity checks* include local and remote firmware attestation, as well as key attestation. In local firmware attestation, typically, the bootloader checks the firmware integrity, while in remote attestation, the token's internal status is checked by a trusted third party. In key attestation, the token gets shipped with an attestation key, which can later be verified (i.e., that the device was initialized in the factory, not somewhere else). Trezor also offers the possibility to manually load firmware, which is implemented in a way that mandatorily deletes all key material. Others ship without firmware and require the load. This method tries to mitigate pre-initialized tokens, firmware modifications, and partly token replicas.

Deployed *hardware authenticity checks* utilize secure CPUs or secure elements, which both shield electronic signals on the printed circuit board (PCB) or within an integrated circuit (IC), in order to prevent interception and manipulation. Thereby, the key material resides inside the CPU or secure element and never leaves it.

3 RELATED WORK

In recent years, several attacks on hardware wallets have been reported. Volotikin [29] and Rashid [25] exploited software vulnerabilities in the Ledger wallets to retrieve or modify secret keys. Nedospasov et al. [22] showed that hardware implants could be inserted into a hardware wallet, which can then remotely trigger the confirmation button, eliminating the need for victim interaction. Furthermore, they revealed that it is feasible to exploit a bug in the Ledger wallet's bootloader to flash custom firmware. Moreover, an incident has been reported where the attacker pre-initialized a hardware wallet and inserted a fake recovery sheet [1, 27]. Grover [13] demonstrated that bootloader protections can be bypassed by unsoldering the locked-down microcontroller and replacing it with a pristine one. Gkaniatsou et al. [12] performed an analysis of the low-level protocols of Ledger wallets. They showed that these protocols can be exploited to carry out attacks, such as gaining unauthorized access to the wallet and altering transactions. As a countermeasure, they suggest a scheme for encrypting specific security-relevant parts of the messages between hardware wallet and client software. However, their approach only defeats man-in-the-middle attacks, but not supply chain attacks or attacks where the client software or firmware is compromised. In contrast to their work, we assume a more comprehensive attack model (see Section 4). Pfeffer et al. [24] showed that currently implemented authenticity checks of hardware security tokens (including hardware wallets) are poorly understood (and performed) by users and do not cover all possible attacks.

Although some of the exploited vulnerabilities have already been addressed by the manufacturers, these attacks show that it is essential to re-design the current hardware wallet architecture to secure against supply-chain and other attacks. Therefore, we suggest critical fixes to the current architecture.

4 HARDWARE WALLET ATTACK MODEL

In the following section, we discuss a broad range of attacks against hardware wallets. This attack model is the basis for our evaluation of the current hardware wallet architecture (see Section 6) and for our proposed solution (see Section 7). Our proof-of-concept implementation (see Appendix A) falls into a subset of this attack model.

The attackers' overall aim is to weaken, pre-load, or exfiltrate the keys or cryptographic seeds stored on the hardware wallet to siphon off money, interrupt its availability, or ask for ransom [5]. To achieve this aim, the attacker can replace or modify the firmware

Table 1: Features and security engineering practices of hardware wallets

		Trezor One	Trezor T	Ledger Nano S	Ledger Blue	KeepKey
Packaging and HW	*Holographic sticker*	Two stickers (package)	One sticker (USB port)	No[1]	No[1]	One sticker (package)
	Tamper-evident box	Sealed with strong glue	No	No[2]	No[2]	No
	Device casing	Ultrason. welded plastic	Ultrason. welded plastic	Plastic	Plastic and aluminum	Anodized aluminum
	Openable device	No	No	Yes (reference pictures of PCB	Yes (no reference pictures of PCB)	No
	SE	No	No	Yes	Yes	No
Software	*BL Check*	By FW (hs)	By FW (hs)	By SE	By SE	By FW (hs)
	FW Check	By BL and SW (sig)	By BL and SW (sig)	- SW checks SE (sig) - SE checks MCU - Remote attestation	- SW checks SE (sig) - SE checks MCU - Remote attestation	By BL and SW (sig)
	Manual FW Load	Yes	Yes	No	No	No
	User Authentication	- Optional PIN - Optional passphrase	- Optional PIN - Optional passphrase	- Mandatory PIN - Optional passphrase	- Mandatory PIN - Optional passphrase	- Optional PIN - Optional passphrase
	Recovery	24-word phrase	24-word phrase	24-word phrase	24-word phrase	24-word phrase
Open Source	*FW*	Yes	Yes	No	No	Yes
	SW	Yes	Yes	Yes	Yes	Yes
	HW	Yes	Yes	No	No[3]	No

Hardware (HW), Client Software (SW), Firmware (FW), Secure Element (SE), Bootloader (BL), Signature check (sig), SHA256 hash (hs)

[1] Manufacturer claims to not use holographic stickers since they give users a false sense of security.
[2] Manufacturer claims that tamper-evident packaging is not needed due to strong device security.
[3] The development version is open-source.

and/or hardware of a **hardware wallet** anywhere and anytime between the token leaving the manufacturer and arriving at the end user. This includes selling fraudulent tokens to end users, inserting them into the re-seller hierarchy, or intercepting and replacing shipments in transit. An attacker might also buy a genuine token and return a tampered one to the vendor, who usually does not check the returned devices before redistribution [14].

The tempered or counterfeit device can leak secrets (i) in-protocol via the signature [10] or other parts of the transaction [4], or (ii) out-of-band using Bluetooth [19], Wi-Fi, GSM [22], or USB exploits [28]. Moreover, the attacker can infect, tamper, exploit, or replace the **client software**. This is the attack model most current hardware wallet manufacturers adhere to.

Finally, we assume that **man-in-the-middle** attacks are possible, where the attacker sniffs and deletes, modifies, or inserts packets between the hardware wallet and the client software, e.g., with a hardware implant. Physical attacks on hardware wallets that require the attacker to temporarily or permanently remove the token from the user are out of scope for this work. Thus, in this paper, we do not discuss secret extraction approaches such as fault injection, timing side-channels, transient execution attacks, IC microprobing, or bus snooping [3].

The **current architecture requires that token manufacturers and designers are trustworthy**, meaning that they are not purposefully backdooring hardware, software, or firmware.

5 METHODOLOGY

We physically inspected the products from Ledger, Trezor, and KeepKey and their security features, observed the initialization procedure, and read available publications, documentation, and standards. Complementing (and broadening) previous publications and attacks, we observed USB protocols of some of those devices (see Section 6).

Out of our observations, we developed a proof-of-concept attack (see Appendix A) for one selected hardware wallet (Trezor One) by (mostly) replaying recorded USB traffic, with only a little additional reverse engineering needed. We chose Trezor One since Ledger hardware wallets have already been subject to reverse engineering [12] and KeepKey is a fork of Trezor [11]. Thus, we demonstrate that Trezor One follows the same architectural design as the other hardware wallets and thus is subject to the same architectural vulnerabilities. Since all major manufacturers use a similar (standardized) wallet design (see Section 2, this demonstrates that fundamental architectural shortcomings exist across the industry.

Finally, based on our architectural findings, we developed improvements using recent advances in mutually verified protocols. We discuss the advantages and disadvantages of different approaches in Section 7. Additionally, we compare the proposed algorithmic trust method to other integrity methods (attestation) with and without the usage of a secure element.

6 DEPLOYED SECURITY ENGINEERING PRACTICES

Based on a literature review and inspection of the currently most widely used hardware wallets [21] from Trezor, Ledger, and KeepKey, we report on currently used defenses against possible attacks (see Section 4). These approaches combine packaging, hardware, and software measures (see Table 1).

Packaging. Tamper-evident packaging (e.g., cardboard boxes, shrink-wrap plastic) or holographic stickers are used to detect whether a package has been opened before delivery. These measures make supply-chain attacks slightly more challenging since an attacker would have to re-package a modified device or token replica in a genuine-looking way. However, all types of packages

can be reproduced. Many holographic stickers can easily be purchased [15] or removed with a blow dryer [22] or acetone. We found that all manufacturers use tamper-evident packaging or holographic stickers even though they are not or only marginally effective.

Hardware. To detect tampered or added hardware, some wallets are openable so that users can visually inspect the printed circuit board (PCB) and compare it with reference pictures by the manufacturer. However, not all manufacturers provide such pictures. Other manufacturers weld the casing to impede hardware-based attacks. Some wallets use secure elements to shield critical data and the PCB, provide side-channel resistance, and tamper-detection circuits within the integrated circuit (IC). However, we found that although very effective, secure elements are rarely deployed.

Software. The authenticity of the bootloader and firmware is verified using hashes or signatures. This is either done locally or remotely, involving a trusted third party and a challenge-response protocol. Although more effective, we found that no remote firmware attestation is implemented in Trezor One. While Ledger performs a remote attestation, it is not used to create a session key for integrity protection. Furthermore, some hardware wallets are shipped without firmware; thereby forcing users to manually load the firmware when initializing their wallet, wiping any pre-loaded keys or seeds. However, in some scenarios, such as our proof-of-concept attack, the firmware update could simply be dropped, and success messages could be faked, including presenting the new version identifier.

7 ARCHITECTURAL CRITIQUE AND SUGGESTED IMPROVEMENTS

Our analysis has shown that the chain of trust is either never enforced or weakly implemented in current implementations. The hardware wallet generation's architecture just shifted the single point of trust from the PC to the hardware token. Thus, its protection ability is limited to PC client manipulations. This design maximizes compatibility with software-only wallets but leaves much of the security improvement potential on the table. Blockchain's most remarkable property is to generate trust from untrusted parties with algorithmic assertions and game theory. In contrast, blockchain (hardware) wallets do not work like that. Trust originates exclusively from the manufacturer of the software and hardware. In BIP 32, the token is solely responsible for the key initialization, key derivation, and signature creation with little oversight by the PC client at critical steps. Consequently, manipulations on those functions might undetectably weaken keys, leak a key, or force a predetermined key onto the user, allowing an attacker to capture funds. This includes the choice of keys (addresses) and nonces in signatures. Tokens can be replaced or manipulated in a number of ways pre- or post-delivery to the user, as discussed and demonstrated in several publications [22, 25, 26, 29].

Moreover, we found that authenticity checks are insufficiently implemented . While our proof-of-concept would be defeatable by a number of attestation methods if implemented correctly (see below), it nonetheless demonstrates a fundamental architectural problem, i.e., that trust in the manufacturer is required. Instead of small incremental patch-ups, we focus on the principal questions. Therefore we suggest a solution that does not require blind trust into an external entity and can be implemented instead of or in addition to a costly secure element with or without remote attestation. The latter both may be infeasible in air-gaped or firewalled installations; or when retrofitting current devices.

7.1 Solution

In spirit with the blockchain's ability to create trust out of untrusted parties, we propose an orthogonal measure: intrinsic trust originating in mutually-verifiable algorithms for the most critical wallet functions: (a) the wallet initialization, (b) the key derivation function, and (c) the signature. This means that even a tampered hardware wallet or PC client software can not alter the outcome to an attacker's advantage, since the other party would immediately detect this. For all wallet functions, either the output must be verifiable by the other party (hardware wallet or client software), or the single-point-of-computation must be split into a collaborative computation so that the client software does not learn the secrets.

7.1.1 *Wallet Initialization.*

Problem. The seed (and subsequent master key) initialization is either performed by the client software or the token without strength and freshness guarantees.

Requirements. Initialization should be performed jointly, offering freshness guarantees, and the secret should only be revealed –if wished– for backup purposes.

Solutions. Collaborative (and distributed) pseudo-random number generation algorithms for the seed exist (e.g., Verifiable Random Function, VRF [9]), but fail to deliver a verifiable path from the master key to the seed, i.e., either the other party would necessarily learn the secret seed, or a zero-knowledge proof about the randomness of the seed would still lack the link to the master key. Thus, allowing hardware wallets to cheat by providing manipulated master keys. We propose a **joint generation of the master key pair** as introduced by Dauterman et al. [7] for U2F tokens. Their interactive two-party protocol creates ECDSA keys in an auditable way with randomness from both parties. The PC client only learns the public key, and the private key never leaves the hardware device.

7.1.2 *Signature.*

Problem. The nonce of the signature is chosen by the hardware wallet and offers an in-protocol covert channel to possibly leak information [10] or allow secret reconstruction (through nonce reuse).

Requirements. The software client should be assured that the hardware token could not have manipulated the random nonce or chosen a weak or non-random nonce without ever learning the secret key.

Solutions. We suggest to adopt a collaborative protocol for **firewalled signature** generation introduced by Dauterman et al. [7], originally developed for U2F tokens. By jointly working on the signature, each party can verify the correctness and offer secret exfiltration resistance. The PC software client does not learn the private key used. Signatures created that way are compatible with Bitcoin signatures and indistinguishable for external observers.

7.1.3 *Key Child Derivation Function.*

Problem. A fraudulent wallet could return weak or predetermined keys for any of the leaves or sub-trees in the hierarchical key structure.

Requirements. The client needs to have the ability to verify the lineage of a public key up to the master key. Additionally, keys must be unique, unforgeable, and unlinkable.

Solutions. Current BIP32 wallets already support two types of keys. So-called *non-hardened keys* are verifiable with the knowledge of the parent public key (and the path, index, or id i). This implies that knowledge of the parent (or master) public keys allows third parties to enumerate the child keys. In contrast, hardened keys are not linkable but also not verifiable. Dauterman et al. introduced **verifiable identity families** (VIF) [7]. With VIFs, only the holder of the secret key can produce new public child keys k/i_p with an index/id i. However, they can prove to anyone holding the parent public key that k_p is a unique public key for i.

*7.1.4 Compatibility to BIP32 and BIP44 wallets.*In conversations, hardware wallet manufacturer Ledger pointed out that adopting Dauterman et al.'s U2F-based techniques for hardware wallets is not straightforward. Although collaborative key generation of the master seed would be feasible, the adoption of VIFs and firewalled signatures would introduce incompatible changes to the hierarchical deterministic wallets (see Section 2.3) currently in use. Dauterman et al. discuss that VIFs (i) do not allow the public key holder (i.e., the client software) to verify that it corresponds to a particular master public key, or otherwise (ii) would not satisfy Σ-pseudorandomness, i.e., linkability protection). However, this is to be expected and corresponds with the hardened vs. non-hardened key problem. While upgrading current wallets is arguably challenging, there seems to be no roadblock to incorporate those techniques into the next generation of hardware wallets (or their firmware). Dual-standard wallets could support both wallet structures in parallel for a smooth transition.

7.2 Comparison of Algorithmic Trust to Effective Attestation

Attestation is often employed to achieve similar goals differently: it ensures the device's integrity as a proxy for not misbehaving. However, past attacks have shown, effective attestation requires a trusted and hardened secure element outside the control of the application processor. The attestation tests and assurances must be verifiable by the client software or a third party (e.g., remote attestation). Attestation methods currently used in popular hardware wallets have been circumvented in many ways with and without a secure element. Thus, our comparison (Table 2) assumes the best possible implementation of the secure element, where the secure element performs all critical tasks, and the client software verifies the outcome and the existence of the secure element.

Algorithmic trust can provide multiple guarantees that, otherwise, only the best-case flawlessly implemented secure-element-supported attestation can provide. Therefore, it can be used where no secure element is present for retrofitting current devices, or a secure element is not integrated for other reasons (e.g., financial). However, since algorithmic trust is mostly an orthogonal technique, both can (and for some attack mitigations need to) be combined.

7.3 Limitations

Above discussed changes to BIP32 do not prevent out-of-band exfiltrations, USB interface, and ransom attacks. They are no substitute for all the other currently employed security measures (Section 6).

Table 2: Comparison of algorithmic trust to attestation with and without a secure element (SE)

Attack	Goal	none	Attestation	Algorithmic Trust	Attestation w/SE[1]	Algorithmic w/SE[1]
Hardware Implants	Snooping and Exfiltration	-	-	-	+	+
	USB Exploit	-	-	-	-	-
	DoS Ransom	-	-	-	-	-
Token Replicas	Algorthmic/Fw. Changes	-	-	+	+	+
	Exfiltration	see above				
IC Modification	predictable RPNG	-	-	+	+	+
Firmware Modification	Key fixation	-	-	+	+	+
	predictable RPNG	-	-	+	+	+
	In-band exfiltration	-	-	+	+	+
	USB Exploit	-	-	-	+	+
	DoS Ransom	see above				
Social Engineering	Token pre-init	-	-	-	-	-

[1] We assume the best possible implementation where the secure element (SE) performs all the security critical tasks in a client-provable way, such as fimware attestation, UI interfacing, and cryptographic primitives.

8 CONCLUSION

The current generation of "high-security" hardware wallets for cryptocurrencies is built on transferring selected duties from a software-only architecture into single-purpose hardware. Adhering to the generic architecture (as defined in BIP32 and BIP44), the design misses the opportunity to introduce a mutually-checked hardware-software design. The current arrangement merely shifts the single point of trust to the hardware wallet without taking provisions to ensure its integrity.

A modified, counterfeit, or compromised hardware token or man-in-the-middle device can siphon off entrusted funds in several ways. Based on the review of standards, protocols, and attacks, we argue this is an industry-wide problem: The root cause is the architecture. Often discussed techniques, such as hardware attestation [20] and communication encryption [12] fall short of protecting against the fundamental architectural flaws.

We propose several modifications to the concept based on collaborative key generation and signatures. Thus, software clients and hardware tokens both provide input for the randomness of keys and signatures, and can mutually verify each other to detect manipulation attempts. Therefore, manufacturer-based trust is strengthened (or replaced) by an intrinsic algorithmic trust.

ACKNOWLEDGMENTS

We thank Phillip Schindler for his valuable feedback and support.

We gratefully acknowledge an ENDEAVOR award from the Donald Bren School of Information and Computer Sciences at UC Irvine. This material is based upon work partially supported by the Defense Advanced Research Projects Agency (DARPA) under contract N66001-20-C-4027, the United States Office of Naval Research (ONR) under contract N00014-17-1-2782, and the Austrian Research Promotion Agency's (FFG) *Industry-related Dissertation* funding program. SBA Research (SBA-K1) is a COMET Center within the framework of COMET – Competence Centers for Excellent Technologies Program and funded by BMK, BMDW, and the province of Vienna. The COMET program is managed by FFG.

REFERENCES

[1] Lawrence Abrams. 2021. Criminals are mailing altered Ledger devices to steal cryptocurrency. https://www.bleepingcomputer.com/news/cryptocurrency/criminals-are-mailing-altered-ledger-devices-to-steal-cryptocurrency/, accessed 2021-08-02.

[2] Alex Baumgarten, Michael Steffen, Matthew Clausman, and Joseph Zambreno. 2011. A case study in hardware Trojan design and implementation. *International Journal of Information Security* 10, 1 (2011).

[3] Swarup Bhunia and Mark Tehranipoor. 2018. *Hardware Security: A Hands-on Learning Approach.* Morgan Kaufmann. ISBN 978-0128124772.

[4] Michael Brengel and Christian Rossow. 2018. Identifying key leakage of Bitcoin users. In *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID).*

[5] Luke Champine. 2019. A Ransom Attack on Hardware Wallets. https://blog.sia.tech/534c075b3a92, accessed: 2020-06-17.

[6] Adrian Dabrowski, Heidelinde Hobel, Johanna Ullrich, Katharina Krombholz, and Edgar Weippl. 2014. Towards a Hardware Trojan Detection Cycle. In *International Workshop on Emerging Cyberthreats and Countermeasures, ECTCM.* https://doi.org/10.1109/ARES.2014.45

[7] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. 2019. True2F: Backdoor-resistant authentication tokens. In *IEEE Symposium on Security and Privacy.*

[8] DIGImend. 2019. USB HID device dumping utility . https://github.com/DIGImend/usbhid-dump. Accessed: 2019-02-12.

[9] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A Verifiable Random Function With Short Proofs and Keys. In *8th International Workshop on Theory and Practice in Public Key Cryptography.* 416–431.

[10] Qingkuan Dong and Guozhen Xiao. 2010. A Subliminal-Free Variant of ECDSA Using Interactive Protocol. In *International Conference on E-Product E-Service and E-Entertainment.*

[11] GitHub. 2021. Search for 'Trezor' within the keepkey firmware code base. https://github.com/keepkey/keepkey-firmware/search?q=trezor, accessed 2021-03-24.

[12] Andriana Gkaniatsou, Myrto Arapinis, and Aggelos Kiayias. 2017. Low-Level Attacks in Bitcoin Wallets. In *Information Security*, Phong Q. Nguyen and Jianying Zhou (Eds.).

[13] Mike Grover. 2021. [OMG]: Bitcoin Stealing Ledger Implant Upgraded: Now *Invisible*. https://youtu.be/oARxLV_vnh0, accessed 2021-08-02.

[14] Andrew Huang. 2019. Supply Chain Security: "If I were a Nation State...". BlueHat IL, video available: https://youtu.be/RqQhWitJ1As, accessed: 2020-06-17.

[15] Intertronix. 2020. Custom Hologram Sticker Online. https://www.intertronix.com/category-s/1673.htm, accessed: 2020-09-16.

[16] Inverse Path. 2018. Open Source Flash-Drive Sized Computer. https://inversepath.com/usbarmory.html. Accessed: 2018-09-21.

[17] kernel.org. 2017. Linux USB HID gadget driver. https://www.kernel.org/doc/Documentation/usb/gadget_hid.txt. Accessed: 2019-02-12.

[18] Ledger. 2019. Export your accounts. https://support.ledger.com/hc/articles/115005297709, accessed: 2020-06-17.

[19] Michael Leibowitz and Joe FitzPatrick. 2017. Secure Tokin' & Doobiekeys: How to roll your own counterfeit hardware security devices. DEF CON 25.

[20] Pieter Maene, Johannes Götzfried, Ruan De Clercq, Tilo Müller, Felix Freiling, and Ingrid Verbauwhede. 2018. Hardware-based trusted computing architectures for isolation and attestation. *IEEE Trans. Comput.* 67, 3 (2018).

[21] Mordor Intelligence. 2021. Global Hardware Wallet Market: Growth, Trends and Forecast (2019–2024). https://www.mordorintelligence.com/industry-reports/hardware-wallet-market, accessed: 2021-01-26.

[22] Dmitry Nedospasov, Josh Datko, and Thomas Roth. 2018. Wallet Fail. https://wallet.fail/, accessed: 2020-06-17.

[23] Marek Palatinus and Pavol Rusnak. 2014. BIP 0044: Multi-Account Hierarchy for Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki, accessed 2021-01-23.

[24] Katharina Pfeffer, Alexandra Mai, Adrian Dabrowski, Matthias Gusenbauer, Philipp Schindler, Edgar Weippl, Michael Franz, and Katharina Krombholz. 2021. On the Usability of Authenticity Checks for Hardware Security Tokens. In *Proceedings of the 30th Usenix Security Symposium.*

[25] Saleem Rashid. 2018. Breaking the Ledger Security Model. https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/, accessed: 2020-06-17.

[26] SatoshiLabs. 2018. Non-genuine Trezor One devices spotted. Be careful, buy only from Trezor Shop or authorized resellers. https://blog.trezor.io/979b64e359a7, accessed: 2020-06-17.

[27] Kai Sedgwick. 2018. Man's Life Savings Stolen from Hardware Wallet Supplied by a Reseller. https://news.bitcoin.com/mans-life-savings-stolen-from-hardware-wallet-supplied-by-a-reseller/, accessed: 2020-06-17.

[28] Jing Tian, Nolen Scaife, Deepak Kumar, Michael Bailey, Adam Bates, and Kevin Butler. 2018. SoK:" Plug & Pray" Today–Understanding USB Insecurity in Versions 1 Through C. In *IEEE Symposium on Security and Privacy.*

[29] Sergei Volokitin. 2018. Software attacks on hardware wallets. *Blackhat USA* (2018). https://www.blackhat.com/us-18/briefings/schedule/#software-attacks-on-hardware-wallets-10665.

[30] Trezor Wiki. 2018. Developers guide: Cryptography. https://wiki.trezor.io/Developers_guide:Cryptography, accessed: 2020-06-17.

[31] Pieter Wuille. 2012. BIP 0032: Hierarchical Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, accessed 2021-01-23.

A PROOF OF CONCEPT

To show an example of the inadequacy of current hardware wallets' design, we built a fake wallet based on replaying captured USB traffic, complementing previous work on firmware exploits and man-in-the-middle attacks. We include this proof of concept, not because it very hard nor complicated, but because it is so surprisingly simple, and thus emphassize the urgency.

Please note, that the herein described attack only fulfills the purpose of adding another aspect to previous shown attacks [12, 22, 25, 29], emphasizing that the architectural shortcomings of the BIP32/44 standard are an industry-wide problem by filling a gap in previous work.

The attack is actually simplier than many before (e.g., Gkaniatsou et al. [12]), as many aspects of the protocol do not need to be understood since (at large portions) replaying binary data is enough to fool the client.

A.1 Attack Scenario

The attacker's goal is to fixate the master seed (and thus, all therefrom generated private and public keys) before the money in transferred into the wallet in the first place. Thus, this scenario starts earlier than those aimed at redirecting outgoing transactions [12]. Therefore, the attacker builds a token replica (or replaces the firmware), which behaves like an authentic token, but generates attacker-chosen keys. Subsequentely, the attacker can monitor these addresses and siphon off money any time after it arrives. The attack is easy to perform for an attacker with limited programming skills in a short time with little financial cost.

A.2 Implementation

We analyzed the USB interface and traffic using UsbTreeView, usbhid-dump [8], Wireshark, and *extcap*. Some semantics needed mild reverse-engineering skills, but many parts can be replayed as verbatim copies without detailed protocol knowledge.

While this methodology simulates a replacement-style distribution attack, matching the USB Armory's form factor to Trezor is left as an engineering exercise (similar to YubiKey look-alikes [19]). It is —as we argue— unnecessary for the demonstration of architectural shortcomings.

Trezor also offers an open-source firmware, but that would trigger a boot loader warning for unsigned firmware. Using a new hardware setup bypasses any boot loader protection build into Trezor as well.

A.2.1 Plattform. USB Armory [16] is a "swiss army knife" tool for USB running Linux in a 66 mm thumb drive form factor. We dressed up the USB descriptor to roughly resemble a Trezor One but added in a USB Ethernet endpoint (RNDIS) for SSH connections. Trezor's client software (a bridge software accessible via a browser) did not object as it only uses the USB Human Interface Device (HID) endpoints [17]. Our proof-of-concept uses firmware version 1.6.3 and Bridge 2.0.19.

Linux USB Gadget driver offers convenient ways to implement

HID endpoints in userspace. Our whole emulation consists of a bash script for USB endpoint configuration via configFS and less than 1000 lines of C code (mostly replay packets in hex form and debug output).

A.2.2 Protocol. We noted that substantial traffic seems to be in cleartext even though it is built out of binary structures (for example the PIN, see Figure 3). After a communication setup or preamble, periodic keep-alive messages ensure the client is connected to the correct device. Some parts of the preamble change with different firmware versions but are fixed for a particular version. Apart from that, we found no apparent attestation tests (Figure 4).

We observed 16 message types, of which we named eight for their obvious functionality. Only a subset of these needs special handling (e.g., copying a sequence number from a request into the return packet). Apart from that, the proof-of-concept replays previously captured data bytes. We named the identified messages *IDENTIFY_ME, TESTNET, TESTNET_RECV, SEND, NO_ANS, OK_TESTNET, OK_SEND, OK_SEND_ADDR, COMMAND1, ..., COMMAND8*. As the names suggest, we operated our research on Bitcoin's official testnet to not disturb operations (and lose real money).

A.3 Results

While not feature complete, the simple analysis and replay of the 16 message types pass the initialization of a new Trezor One wallet and generation of (now predetermined) addresses in the original client software (Figure 7). They were thus giving the attacker direct access to all funds ever transferred to wallets of the device (Figure 5 and 6).

A.4 Responsible Disclosure

We disclosed the proof-of-concept to Satoshi Labs but they did not classify it as an attack. In their statement, they emphasized that in their open-source architecture, anyone can build their own Trezor clone. They did not acknowledge the architectural implications. We argue that the current architecture requires unchecked trust in the token's manufacturer without ever proving the device's actual origin.

When discovering multiple cases of unauthorized Trezor One clones [26] in 2018, the Satoshi Labs claimed that they pursued a "number of legal and other steps" to prevent fake tokens from being distributed. In fact, they issued an explicit warning to customers to check the holographic stickers on the package when receiving it. However, such stickers do not provide sufficient protection against token replica attacks (see Section 6). Hence, we argue that a crucial change in the hardware wallets architecture is needed in order to prevent such attacks as well as other currently possible attacks (see Section 4).

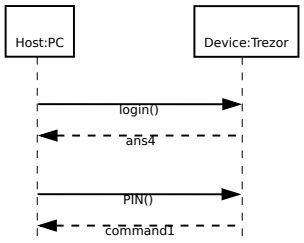

Figure 3: Cleartext PIN

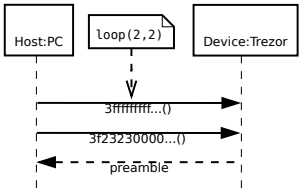

Figure 4: Preamble traffic

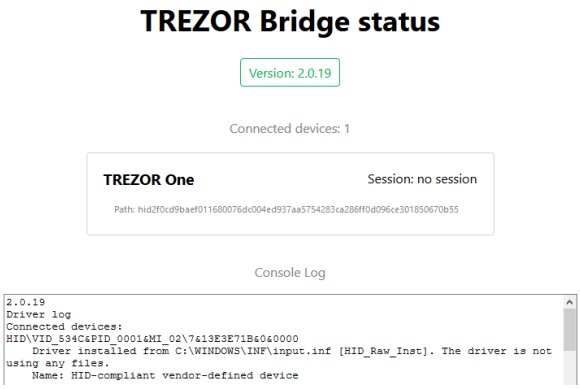

Figure 5: Trezor Bridge is successfully identifying our device as Trezor One.

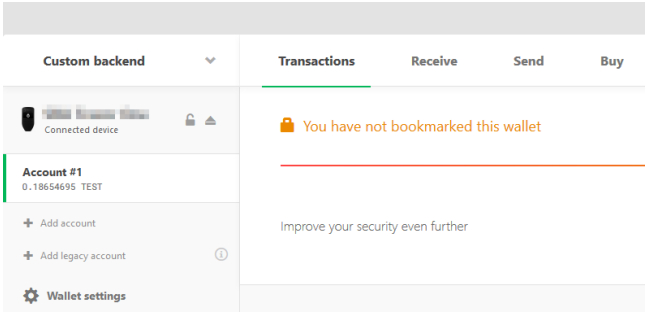

Figure 6: Screenshot of original client: fake Trezor One wallet after initialization.

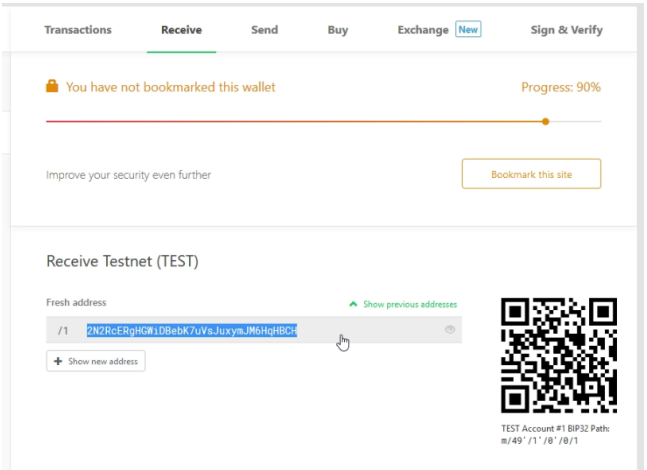

Figure 7: The client software presents the predetermined address as "fresh".