Notary: A Device for Secure Transaction Approval

Athalye et al., presented by Jack Cook

Overview

- Goals and Big Ideas
- Threat Model
- Strengths
- Weaknesses
- Evaluation
- Attack Defenses
- Discussion Questions

Notary’s Goals

- Verify a wide variety of secure transactions, such as BTC transactions, DNS updates, and more
- Provide secure task switching between multiple agents running on a single device
- Defend against security vulnerabilities that have plagued multiple existing hardware wallets

Existing Solutions

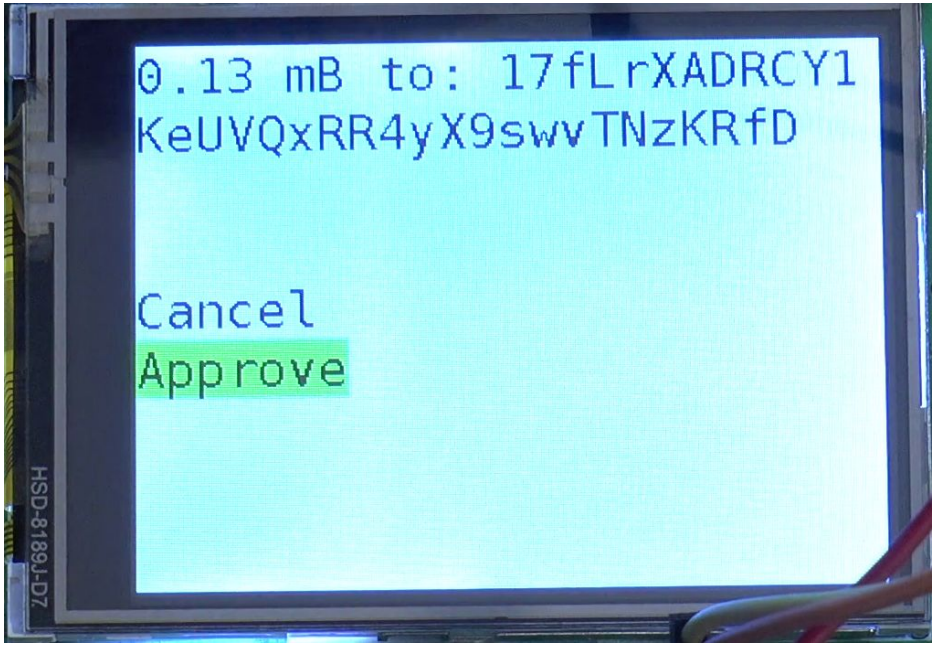
- Mobile apps (e.g. Bitcoin wallets, 2FA apps) are susceptible to process isolation issues, and smartphones have had bugs that can give adversaries root access
- Hardware wallets (e.g. Ledger, KeepKey, Trezor) have had OS bugs and exploitable side channels
    - System call vulnerabilities, memory protection errors, USB software bugs

Big Ideas

- Separating applications and the kernel into three components significantly reduces the attack surface
- Reset-based switching clears the device's microarchitectural state before executing code from a new agent
- A "trustworthy I/O path" between agent code and the user prevents adversarial tampering
- Deterministic start ensures that agents can't interfere with each other

Threat Model

- Notary defends against adversaries that want to approve an operation against the wishes of the device's owner
- It employs multiple defenses in order to do this, in a way that goes beyond existing hardware wallets



Strengths

- Notary solves a very real problem: the paper listed several motivations for wanting to verify important transactions
- Reset-based task switching is a simple but powerful concept, and allows multiple agents to be used on the same device
- Robust threat model: tainted kernel, malicious agents -- thwarted by, among other things, very strong isolation between processes

Weaknesses

- The “trustworthy I/O path” is susceptible to abuse, and weakens the practicality of the device
  - What happens if reviewers make an honest mistake?
  - What happens if the company reviewing new agents goes under?
- The paper uses LOC as a proxy for complexity, which can be misleading

Evaluation

- Notary has robust defenses against issues that have affected other hardware security wallets
- Notary has a verifiable deterministic start, which ensures security
- New agents are easy to develop for Notary
- Reset-based agent switching is fast and practical
- Notary is only slightly more expensive to produce than existing hardware wallets

Poll

- What types of attacks does Notary’s design defend against?
  - Rowhammer
  - Power side channels
  - Microarchitectural side channels
  - Kernel vulnerabilities
  - USB software bugs
  - All of the above

Poll

- What types of attacks does Notary's design defend against?
  - Rowhammer
  - Power side channels
  - Microarchitectural side channels
  - Kernel vulnerabilities
  - USB software bugs
  - All of the above

Physical Domain Separation

- Putting USB communication in its own domain blunts potential effects of USB software bugs
- Keeping the kernel in a separate domain also protects agents from kernel vulnerabilities
- Having multiple SoCs additionally defends against Rowhammer-type attacks

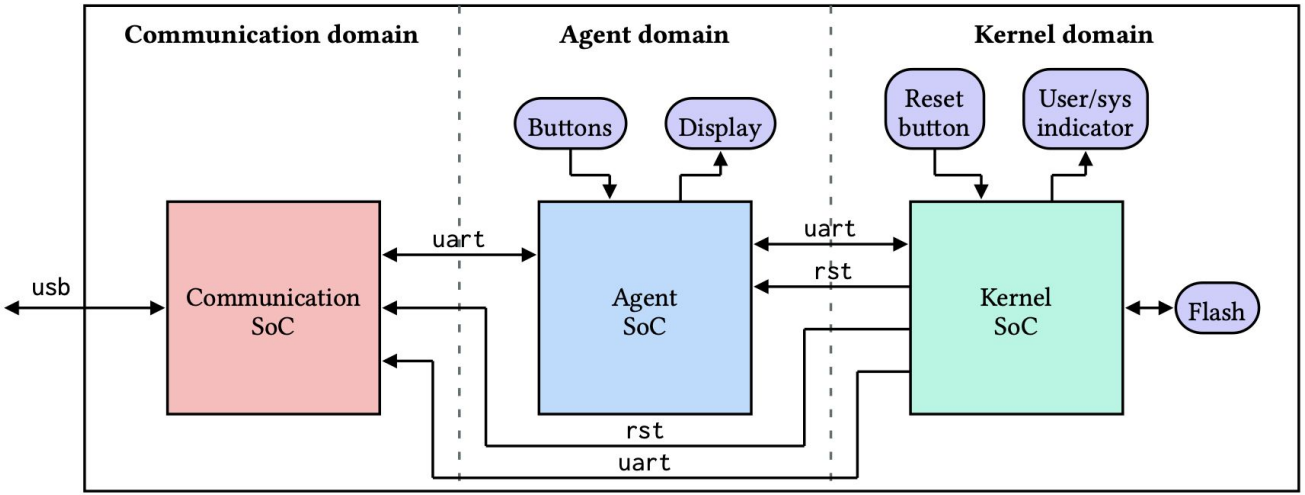


Figure 1. NOTARY's design physically separates trust domains with an SoC per domain and a simple interconnect between trust domains (reset wire and UART). NOTARY employs two such separations, the first between the kernel and the agent UI/signing code, and the second between the agent UI/signing code and the agent communication code.

Non-microarchitectural side channels

“Similarly, except for microarchitectural side channels, Notary’s threat model does not include arbitrary side channels [76] such as electromagnetic radiation [12], power analysis [44], and acoustic analysis [30].

Reset-based Task Switching

- To defend against microarchitectural side channels, Notary employed reset-based task switching
- While switching between separate agents, the microarchitectural state is reset to a deterministic default state

Reset-based Task Switching

- Goal: Completely clear internal state before executing code from a new agent
- Reset pin doesn’t guarantee complete reset — registers may be left untouched
- Power cycling can leave state in SRAM for minutes, which can be exploited through cold boot attacks

Reset-based Task Switching

- Solution: Use a software-assisted deterministic start
  - Code runs on CPU as the system resets
  - Clears all architectural state, microarchitectural state, RAM state, and peripheral state

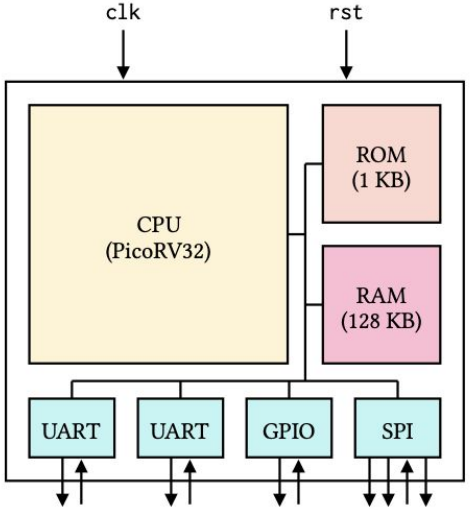


Figure 2. A schematic of NOTARY's agent domain SoC, which is formally verified to satisfy deterministic start.

Discussion Questions (Security)

- Is it acceptable to list power channels outside of the threat model? If my wallet is plugged into an arbitrary malicious usb port, is that port supplying power, which can then be monitored?
- Is there anything a malicious agent binary can do? Denial of service? Spoof being another program and confuse the user?
- Are replay attacks possible with the untrusted USB interface?

Discussion Questions (Practicality)

- How complicated does the agent CPU need to be? To simplify reasoning about the reset process, could we just use a simpler CPU rather than a RISC-V chip (such as an ATMega device)?
- Do the limits on agent storage/IO affect the expressiveness of potential agent code? For instance, are there any agents currently deployed on other HW key platforms which cannot be ported to Notary?
- Could you have hardware that ships with this style of wallet already built-in?

Discussion Questions (Agents)

- How does the kernel know when to switch applications?
- Can agents have any internal storage? How would they access and update it?
- Is there any good way to validate during registration that a public key is coming from a correct agent and not a malicious one, in a way securing the registration process as well -- possibly with a root-of-trust signature from the manufacturer?