Experimentálna aplikácia pre hardvérovú peňaženku Ledger Nano S

Daniel Oravec
doc. RNDr. Robert Lukoťka, PhD.

23. júna 2022

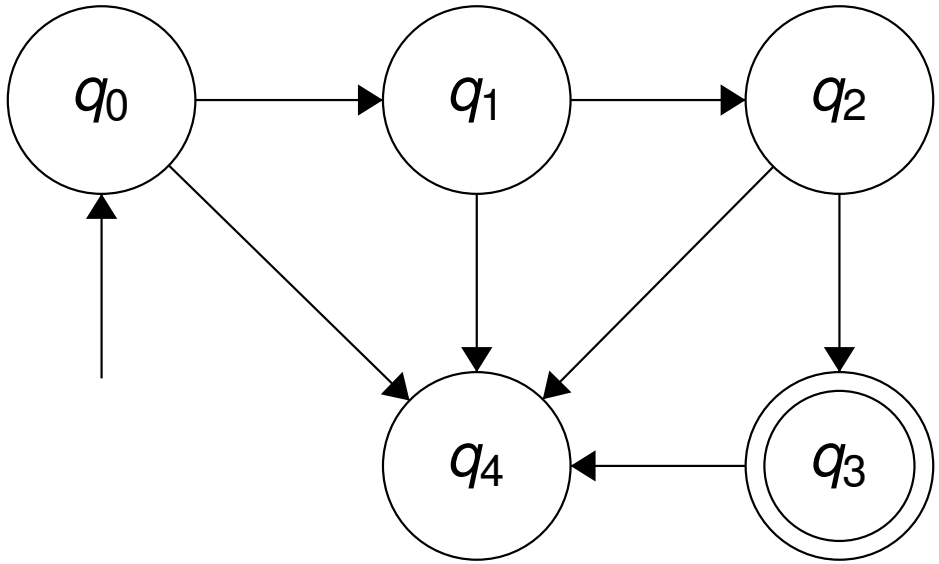
Hardvérová peňaženka

Zariadenie na ukladanie súkromných kľúčov.

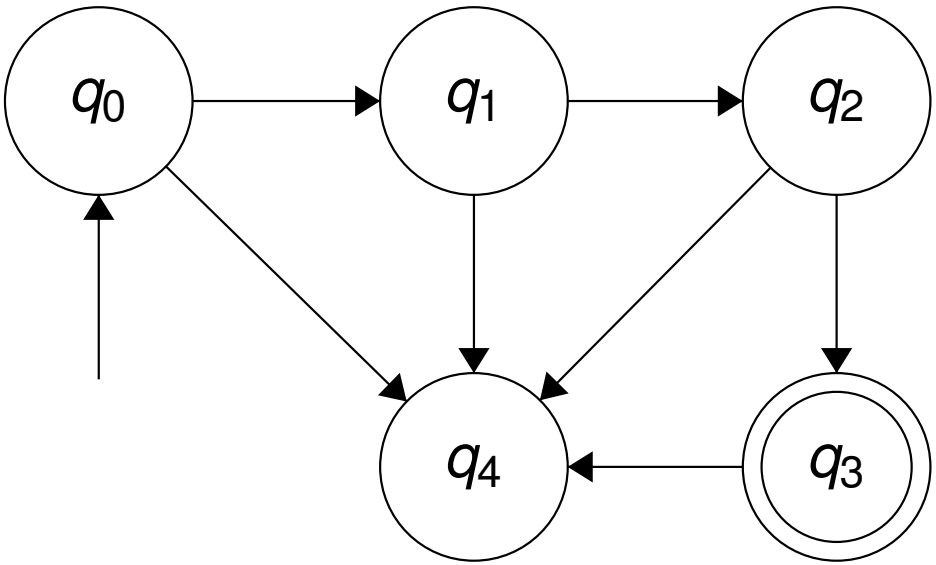
- Napadnutý SW klient
  - Zobrazí falošnú transakciu
- SW klient s HW peňaženkou
  - SW klient postaví transakciu
  - Používateľ skontroluje transakciu v HW peňaženke
    - Vie odhaliť falošnú transakciu
  - HW peňaženka podpíše hash transakcie

Bežná aplikácia

- Pamätá si povolené štruktúry transakcií
- Vnútorný stav: aká časť transakcie je očakávaná
- Konečný stavový automat pre každý typ transakcie



Príklad behu bežnej aplikácie



q_0 – hlavičky
q_1 – množstvo
q_2 – prijímateľ
q_3 – vytvorenie podpisu
q_4 – používateľ zamietol

1. Spustenie aplikácie, nastavenie stavu na q_0
2. Prijatie hlavičiek, ich validácia, posun stavu na q_1
3. Prijatie množstva, zobrazenie používateľovi, posun stavu na q_2
4. Prijatie príjemcu, zobrazenie používateľovi, posun stavu na q_3
5. Prijatie žiadosti o podpis, potvrdenie transakcie používateľom, podpísanie hashu transakcie

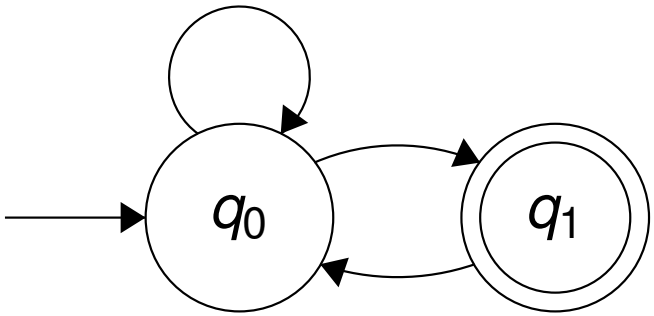
Problémy

- Obmedzené zdroje
  - 160 kB flash pamäte
  - 4 kB RAM
  - Pridanie nového typu transakcie do bežnej aplikácie môže aplikáciu príliš zväčšiť

Experimentálny prístup

- Automat chceme presunúť mimo aplikácie
- Hash transakcie ovplyvňujú iba nekonštantné dáta, napr. `6` v (“`Množstvo`”, `6`)
- Konštantné dáta sú predpísané, napr. “`Množstvo`” v (“`Množstvo`”, `6`)
- Okrem hashu transakcie sa počíta rolling hash z konštantných dát (integrity hash)
- Na konci aplikácia overí, či je integrity hash v zozname povolených hashov

Diagram behu experimentálnej aplikácie



q_0 – aplikácia ešte nebola ukončená
q_1 – integrity hash je povolený

Príklad behu experimentálnej aplikácie

```
INIT_HASH()
SEND_DATA(
    "Príjemca"
    "123456abcdef",
)
SEND_DATA(
    "Množstvo"
    61,
)
SEND_DATA(
    "Dátum"
    "23-06-2022",
)
END_HASH()
```

$$iH = sha256(\text{“INIT\_HASH”} \| \text{“SEND\_DATA”} \| \text{“Príjemca”} \| \text{“SEND\_DATA”} \| \text{“Množstvo”} \| \text{“SEND\_DATA”} \| \text{“Dátum”} \| \text{“END\_HASH”})$$

$$txH = sha256(\text{“123456abcdef”} \| 61 \| \text{“23-06-2022”})$$

Polia

- V transakcii môžu byť polia
- Potrebujeme: rôzne dlhé polia $\rightarrow$ rovnaký integrity hash
  - Kvôli počtu integrity hashov
- Náčrt riešenia bol v zadaní

Potenciálne výhody experimentálneho prístupu

- Zmenšenie veľkosti aplikácie
  - Napríklad aplikácia pre Cardano je veľká
- Presunutie zložitosti vývoja na klienta
  - Takmer žiadna zmena kódu aplikácie pri rozširovaní

Prečo vyvíjame experimentálnu aplikáciu

Chceme odhaliť problémy, akými môžu byť:

- Nedostatok dostupnej RAM
- Veľkosť aplikácie
- Rýchlosť aplikácie

Experimentálne inštrukcie

Sada inštrukcií pre podporu spracovávania rôznych typov transakcií:

1. `INIT_HASH`
2. `END_HASH`
3. `SEND_DATA`
4. `START_FOR`
5. `START_IT`
6. `END_IT`
7. `END_FOR`

Riešenie cyklov

1. `INIT_HASH()`
2. iH_1 = `sha256("INIT_HASH")`
3. `SEND_DATA("Príjemca", "123456abcdef", potvrdenie=true)`
4. iH_2 = `sha256(`iH_1`||"SEND_DATA"||"Príjemca"||true)`
5. `START_FOR(minIt, maxIt, hashOkItHashov)`
6. iH_3 =`sha256(`iH_2`||"START_FOR"||minIt||maxIt||hashOkItHashov)`
7. `START_ITER()`
8. $iterH_1$ = `sha256(`iH_3`||"START_ITER")`
9. `SEND_DATA("Poplatok", 11, potvrdenie=true)`
10. $iterH_2$ = `sha256(`$iterH_1$`||"SEND_DATA"||"Poplatok"||true)`
11. `END_ITER(okIterácie)`
12. $iterH_{final}$ = `sha256(`$iterH_2$`||"END_ITER")`
13. `if` $iterH_{final} \notin$ `okIterácie` $\vee$ `sha256(okIterácie)` $\neq$ `hashOkItHashov:`
14. `FAIL`
15. `END_FOR()` iH_4 = `sha256(`iH_3`||"END_FOR")`
16. `END_HASH()` iH_{final} = `sha256(`iH_4`||"END_HASH")`

Pridávanie nových transakcií na strane klienta

- Interpreter, ktorý dostane šablónu transakcie a dáta a vybaví komunikáciu s peňaženkou
- Šablóna je JSON súbor

Príklad šablóny

```
{
    instrukcie: [
        {
            meno: "INIT_HASH"
        },
        {
            meno: "SEND_DATA",
            parametre: {
                hlavicka: "Poplatok",
                potvrdenie: true
            },
        },
        {
            meno: "END_HASH"
        }
    ]
}
```

Výsledky

- Zvýšil sa počet potrebných komunikačných výmen
  - Používateľnosť to však príliš citeľne nezhoršilo
- Pridávanie podpory pre nový typ transakcie je jednoduché
  - Aplikácia: pridať hash
  - Klient: napísať JSON šablónu
- Ostal približne 1 kB voľnej RAM
- Aplikácia zaberá 34 kB pamäte (zo 160 kB)
- Experimentálna aplikácia zvláda viac ako bežná FIO aplikácia

Budúca práca

- Model aplikácie a dôkaz bezpečnosti v tomto modeli nie sú súčasťou práce
  - Dôkaz chceme dokončiť v budúcnosti
  - Súčasťou našej práce je iba neformálne zdôvodnenie bezpečnosti
- Aplikovanie experimentálneho prístupu na zložitejšie aplikácie
  - Napríklad aplikácia pre Cardano

Ďakujem za pozornosť

- Model aplikácie a dôkaz bezpečnosti v tomto modeli nie sú súčasťou práce
  - Dôkaz chceme dokončiť v budúcnosti
  - Súčasťou našej práce je iba neformálne zdôvodnenie bezpečnosti
- Aplikovanie experimentálneho prístupu na zložitejšie aplikácie
  - Napríklad aplikácia pre Cardano

Otázky

1. Chceme užívateľovi zobraziť iné dáta ako sú serializované do transakcie, napr. derivačnú cestu miesto hashu verejného kľúča.
2. Transakcia obsahuje pole, ktorého obsahom je hodnota, ktorá je niekedy nebezpečná (hoci prípustná) a treba ju zobraziť, ale bežne ju naopak chceme schovať, aby sme nezaťažovali užívateľa.
3. Transakcia obsahuje voliteľné polia, napr. Cardano ich má viac ako 10. Dá sa dosiahnuť, aby počet potrebných integritných hashov nerástol exponenciálne?