Empowered by Innovation NEC

Secure Erasure and Code Update in Legacy Sensors*

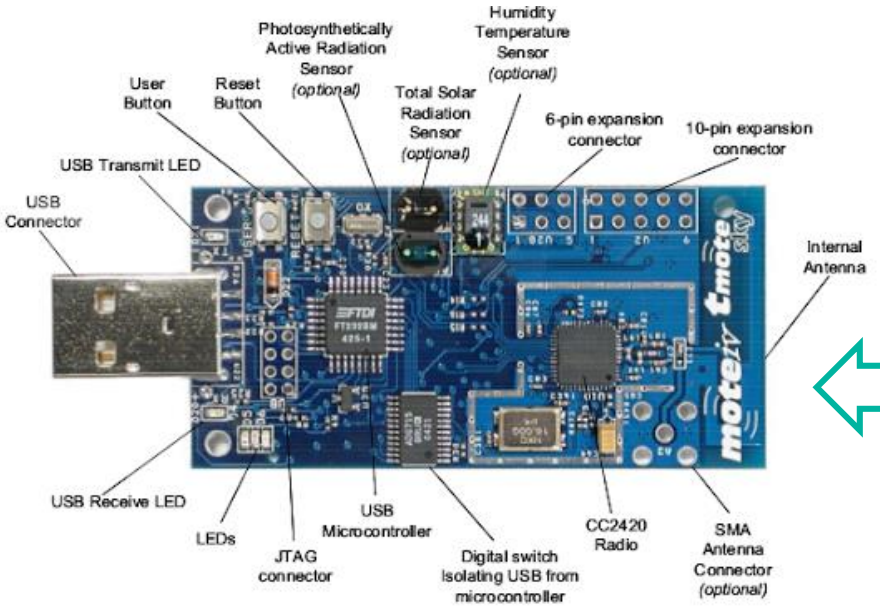
21. ITG Fachtagung Mobilkommunikation, 2016 Osnabrueck

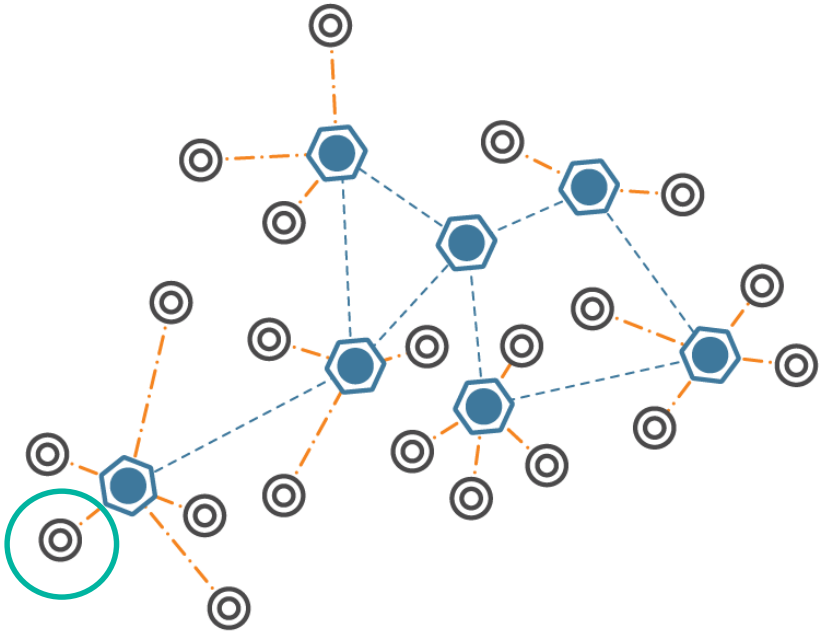
NEC Laboratories Europe
Ghassan Karame
Wenting Li

* Published in 8th International Conference on Trust & Trustworthy Computing (TRUST15')

Sensor network

- Our world is populated by sensors





How to update these devices securely and efficiently?

Goals

Secure Update:

- Confirm the device is in a known state
  - No malicious code remains
- Verify the device is clean (no malware) before update
  - Updates may contain sensitive information

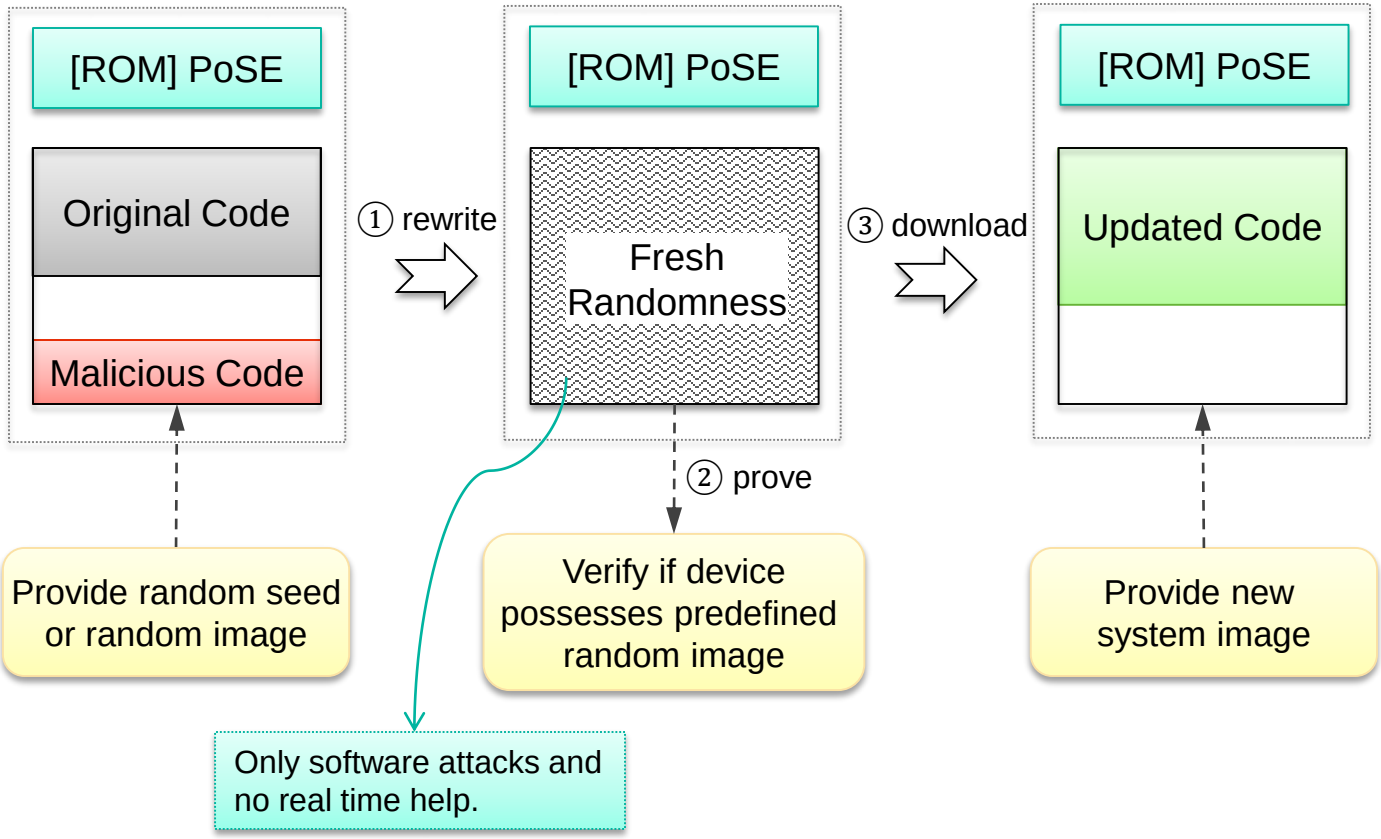
Efficient Update:

- We do not want to drain the batteries

Related Approaches

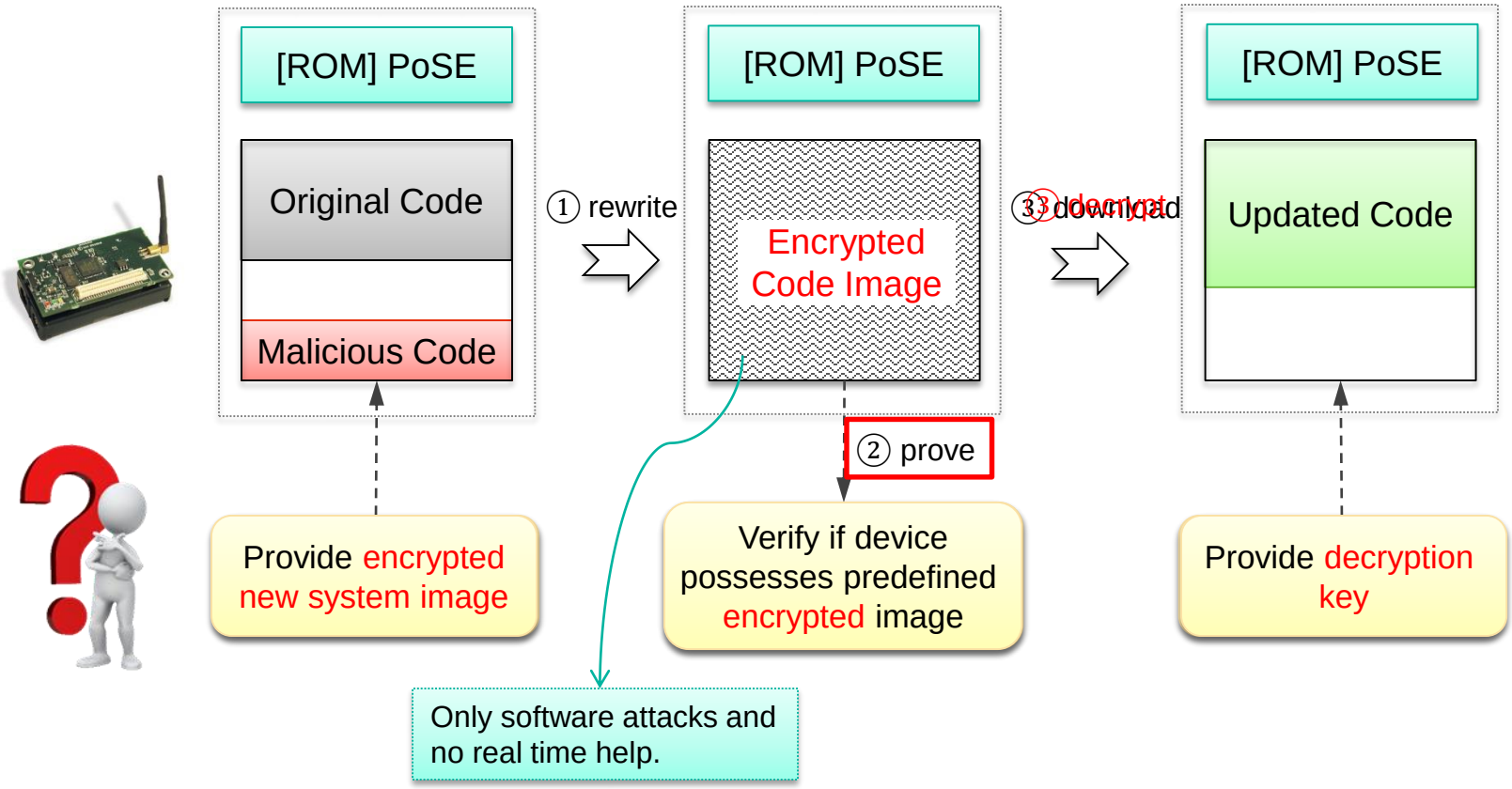
- Device attestation
  - Hardware-based: secure but not likely supported
  - Software-based: several attacks have been reported
- Secure memory erasure

Proofs of Secure Erasure (PoSE[1])



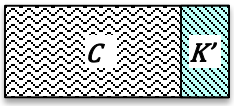
[1] Perito, D., and Tsudik, G. Secure code update for embedded devices via proofs of secure erasure. In Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, pp. 643–662.

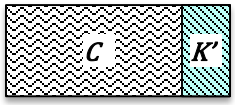
Secure Code Update based on PoSE[1]



[1] Perito, D., and Tsudik, G. Secure code update for embedded devices via proofs of secure erasure. In Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, pp. 643–662.

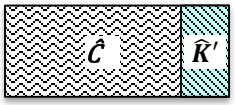
Approach 1: Computationally expensive





K

$I = MAC_{K'}(C)$

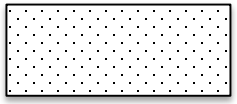


② $\hat{I} = MAC_{\hat{K}'}(\hat{C})$

$\hat{I} =? I$

③ decrypt with K

K



- Computationally costly*
- Extra code space for MAC (4.5KB of ROM)

*HMAC-SHA1 over 648KB in MicaZ requires ~90 seconds

Approach 2: Additional communication round

① $c\ c\ c\ K'$, $\boldsymbol{r}$

$c\ c\ c\ K'$ $\boldsymbol{K}$

$(i_1 \dots i_t) \leftarrow r$

$I = MAC_{K'}(c_{i_1} \dots c_{i_t})$

② $\hat{I} = MAC_{\hat{K}'}(\hat{c}_{\hat{i}_1} \dots \hat{c}_{\hat{i}_t})$

$\hat{c}\ \hat{c}\ \hat{c}\ \hat{K}'$, $\hat{\boldsymbol{r}}$

$(\hat{i}_1 \dots \hat{i}_t) \leftarrow \hat{r}$

$\hat{I} =? I$

③ decrypt with $\boldsymbol{K}$

$\boldsymbol{K}$

- Computation reduced to $\boldsymbol{t}$ blocks
- Probabilistic result*

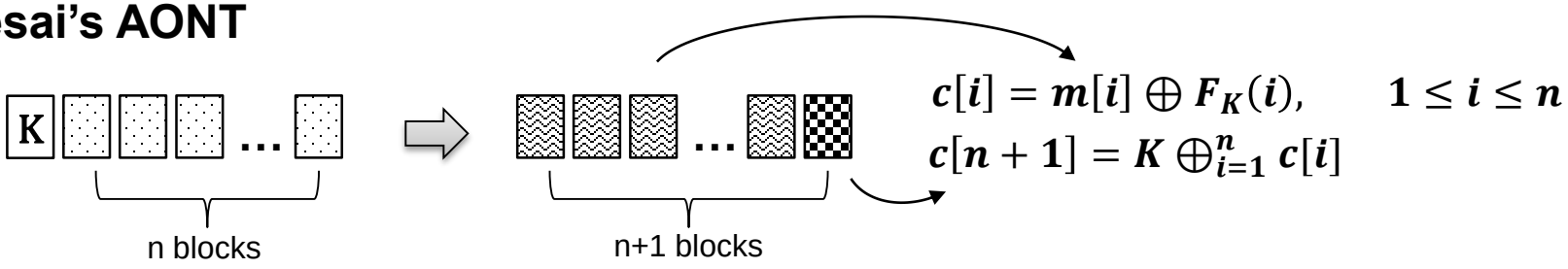
*648KB memory, block size 128 bit, in order to achieve 90% detection probability for 1,000 bits of malicious code, 30% of the data blocks need to be examined

Intuitive Idea

- Principle of using PoSE in secure code update:
  - Only by having all the bits of the encrypted code image (step ①) can we retrieve the updated code image (step ③).

- All or Nothing Transform (AONT)
  - → Given all but one of the output blocks, it is infeasible to compute any of the original input blocks

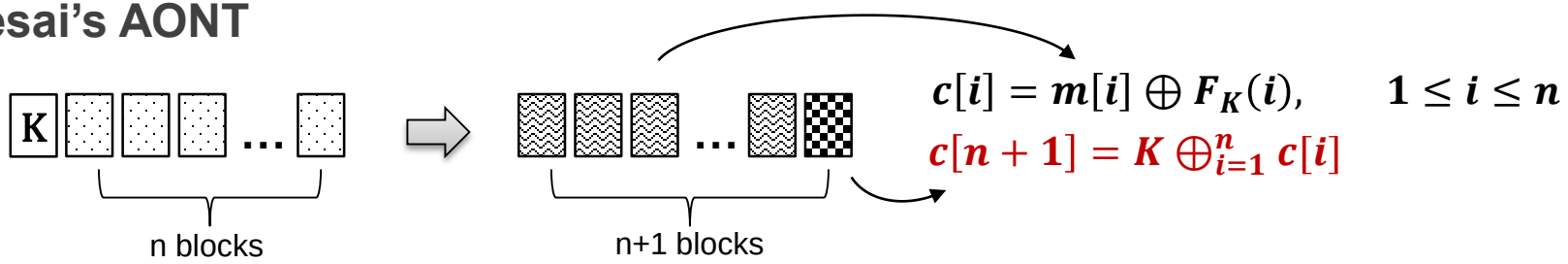
Desai's AONT



- *Would AONT deduce a more efficient secure code update protocol?*

Secure code update based on AONT

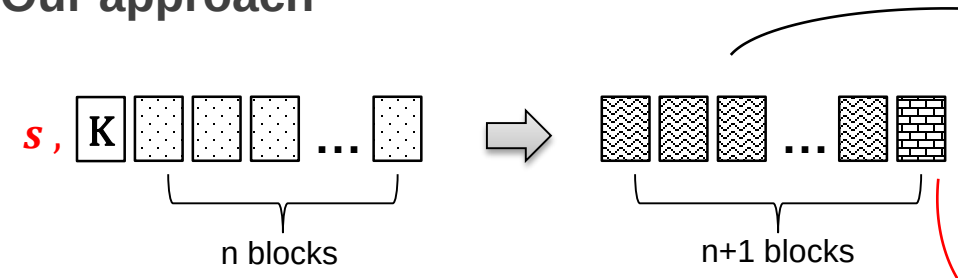
Desai's AONT



$$c[i] = m[i] \oplus F_K(i), \qquad 1 \le i \le n$$

$$c[n+1] = K \oplus_{i=1}^{n} c[i]$$

Our approach



$$c[i] = m[i] \oplus F_K(i), \qquad 1 \le i \le n$$

$$(r_1 \dots r_n) \leftarrow s,$$

$G_r(x)$ cyclic shift x for r bits

$$c[n+1] = K \oplus_{i=1}^{n} G_{r_i}(c[i])$$

The Complete Picture

① C , s

C K, s

$\hat{C}$, $\hat{s}$

② $(\hat{r}_1 \dots \hat{r}_n) \leftarrow s$,
$\hat{K} = \hat{c}[n+1] \oplus_{i=1}^{n} G_{\hat{r}_i} \hat{c}[i]$

③ decrypt with $\hat{K}$

- Single round
- No cryptographic operations for memory erasure in order to retrieve the decryption key

Security Analysis

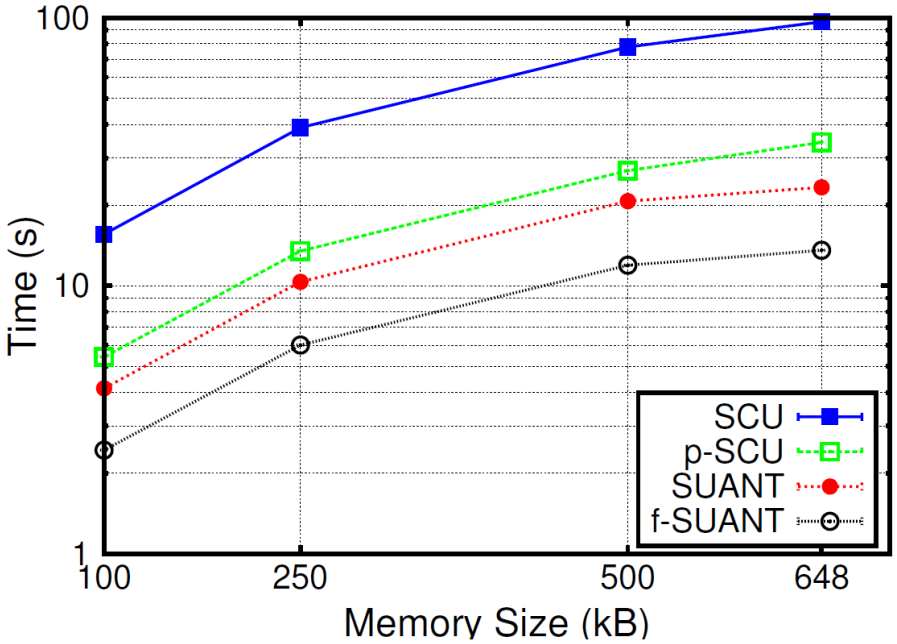
- Adversary has to guess the shifted number of bits for each block that she wants to drop.
- Every bit of each block affects the decryption key.
- To drop b m-bit blocks, $\mathrm{P_A} = \max(m^{-b}, 2^{-m})$

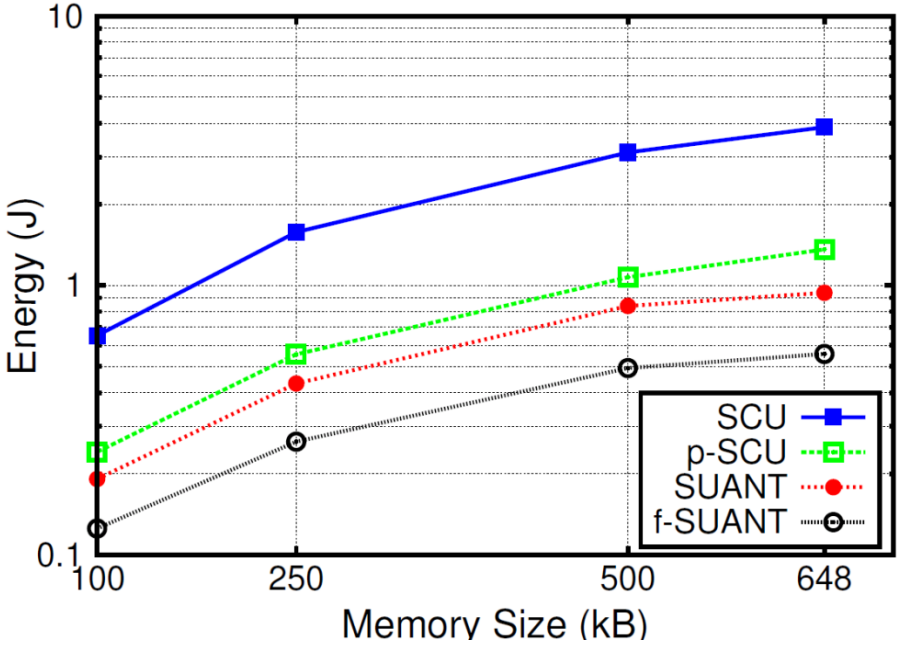
- Optimization on I/O (**f-SUANT**)
  - Compute key block on selected fraction f of the blocks.
    - I/O reduced by $1 - f$
    - $\mathrm{P_A} = \max(\mathrm{m}^{-b}, (1-f)^b)$

Experiments Evaluation

- Measurement Settings
  - MicaZ
    - 128KB internal flash, mask certain section to ROM, 4KB of EEPROM, 4KB of SRAM, 4KB EEPROM, 512KB external flash
    - TinyOS 2.1.2
  - Avrora simulator
    - Estimation of energy consumption

Time and Energy Consumption





Code Size

- SUANT requires less ROM space and leaves smaller footprints in RAM:

	Total Memory (bytes)	RAM (bytes)	ROM (bytes)
SUANT	15,516	371	6822
***f*-SUANT**	15,718	384	6960
SCU	19,256	610	8562
***p*-SCU**	19,436	614	9722

Table 1. Required code and volatile memory sizes.

Conclusion

- What we have gone through:
  - SCU with MAC is inefficient
  - How to construct secure code update protocol based on all-or-nothing transforms
- Gains
  - Much less energy and time consumption than SotA
    - 75% more efficient than MAC
    - 30% more efficient than MAC + PDP and more secure
  - Less ROM and RAM requirement

Thank you!!
Questions?

Empowered by Innovation

Extra – ROM discussion

- Mask ROM: e.g., MSP430 micro-controller
- Lockable memory: e.g. ATmega128 (unlock only by physical access)