Improving communication security of open source UAVs: encrypting Radio Control link

Michal Podhradský

Galois Inc.

ICUAS 2017

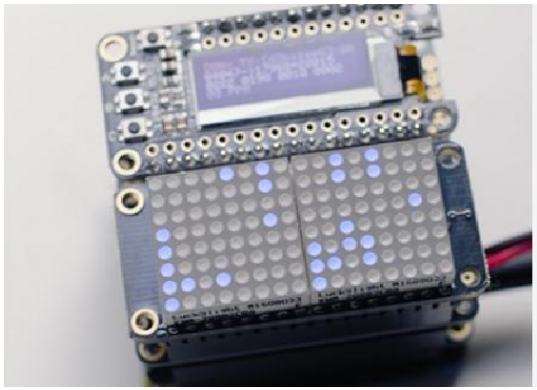
Communications security

Communications security (COMSEC) is the discipline of preventing unauthorized interceptors from accessing telecommunications in an intelligible form, while still delivering content to the intended recipients [Wikipedia]

Why should we care?
And why now?

Known attacks and existing tools:

- **Icarus**: Jonathan Andersson's attack on DSMx with SDR
- **Skyjack:** attack on ArDrone via WiFi
- **Dronedefender** from Batelle



Why should you care?

1. Increased reliability
2. Demonstrating the "best practices"

Different levels of security

1. encryption (our talk)
2. secure-by-design / memory-safe languages (**SMACCMPilot**)
3. formally verified (**seL4 microkernel**)

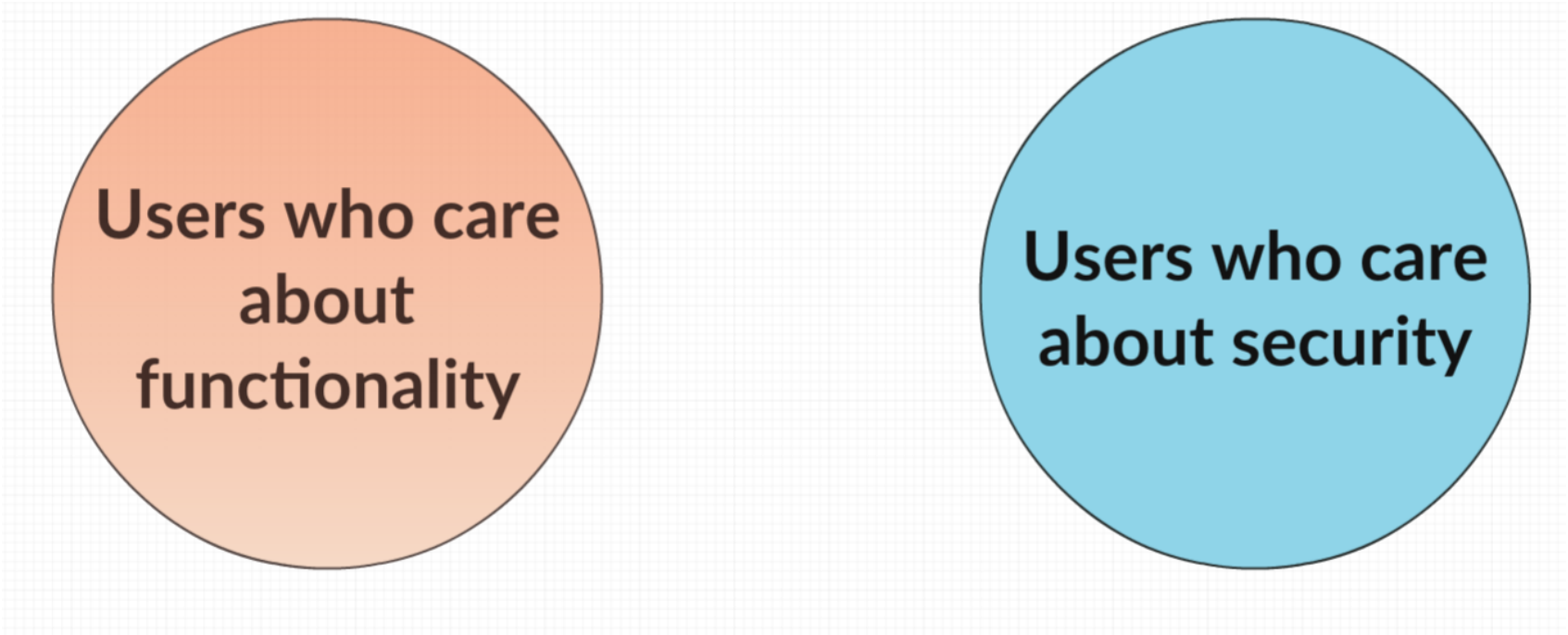
Different levels of security

1. **encryption** (our talk)
2. secure-by-design / memory-safe languages (**SMACCMPilot**)
3. formally verified (**seL4 microkernel**)

Current situation

To this date, no consumer grade encrypted radio control link available on the market!

Current situation



Ideal situation

Solution

Requirements

1. use the same hardware as stock RC modules
2. no hardware modifications
3. use strong and properly implemented cryptography algorithm
4. maintain a reasonable bandwidth in comparison with an unencrypted link

1. and 2. are necessary to make encrypted RC link accessible to a broad spectrum of users, while 3. and 4. are necessary for the functionality.

Symmetric key problem
ChaCha example

- IV - initialization vector
- K - symmetric key
- $||$ - concatenation
- $E()/D()$ - encryption/decryption
- P - plaintext
- C - ciphertext

$$E_K^{IV}(P) = C$$
$$D_K^{IV}(C) = P$$

Symmetric key problem
ChaCha example

Constructing a message:

- $E_K^{IV}(P) = C$
- $msg = IV||C||Auth()$

But:

- $E_K^{IV_1}(P_1) = C_1$ and $E_K^{IV_2}(P_2) = C_2$
- Problem if $IV_1 = IV_2$
- Possible to execute **attack on one-time pad**!

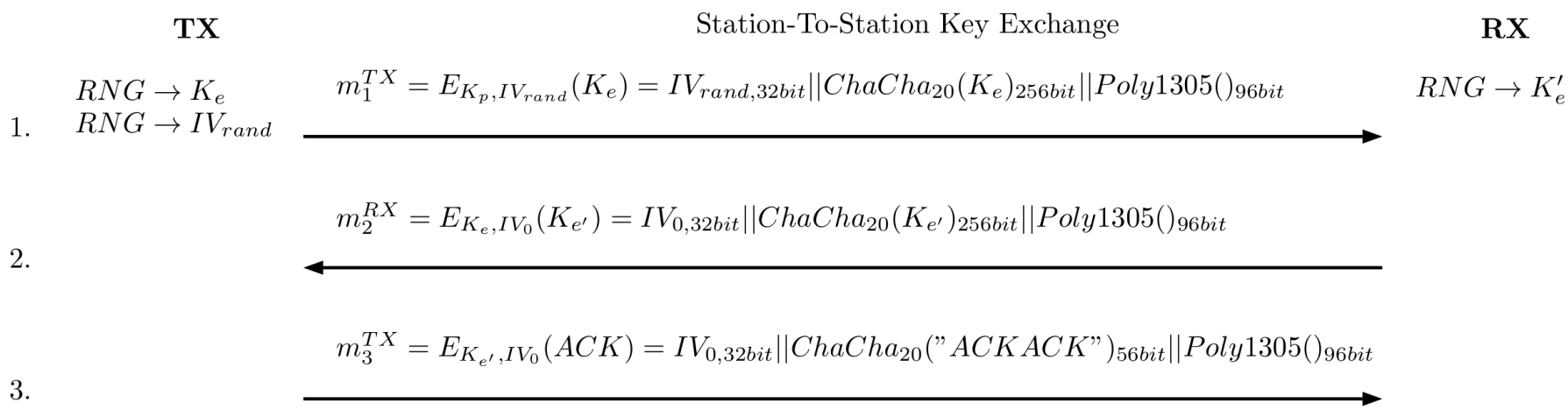
Use permanent key only for the key exchange!

Definitions

$|g|$

- TX - transmitter module
- RX - receiver module
- K_p - permanent key
- K_e - ephemeral TX key
- $K_{e'}$ - ephemeral RX key
- RNG - random number generator (use ChaCha)

Key exchange protocol



Ongoing communication

Ongoing Communication

$$m_i^{TX} = E_{K_e, IV^{TX}}(data) = IV_{32bit}^{TX} || ChaCha_{20}(data)_{datalength} || Poly1305()_{96bit}; IV^{TX} = (i+1) \times 16$$

$$m_j^{RX} = E_{K_{e'}, IV^{RX}}(data) = IV_{32bit}^{RX} || ChaCha_{20}(data)_{datalength} || Poly1305()_{96bit}; IV^{RX} = (j+1) \times 16$$

Implementation

OpenLRSng
openlrsng.org

| g |

- open source (GPL licence)
- readily available HW
- 40km+ range
- popular
- uses Arduino (limited memory and power!)
- can use existing Arduino Crypto library
- nice configuration GUI

Memory footprint

variable	size [bytes]
ChaChaPoly	221
K_p	32
K_e	32
$K_{e'}$	32
IV_rand	4
IV_RX	4
IV_TX	4
cnt_RX	4
extra buffer TX	16
extra buffer RX	16
Total memory	**365**

Time overhead

operation	time [μs]
encrypt/decrypt (per byte)	15
hashing (per byte)	26
key setup	43
finalize hash	500
Total time (one-device)	**1404**
Total time (plaintext-to-plaintext[1])	**2808**

[1] on top of transmission delay

Time overhead

operation	time [*ms*]
crypto overhead	3
trasmission time	20
Total time per packet[2] (no crypto)	**20**
Total time per packet[3] (crypto)	**23**
Max packet frequency (no crypto)	**50** [Hz]
Max packet frequency (crypto)	**40** [Hz]

Theoretically only 20% slower.

[2] 21 bytes
[3] 37 bytes

Results & Future work

OpenLRSng
openlrsng.org

1. ~~readily available HW~~ only one store, long shipping delays, faulty hardware
2. ~~popular~~ only one semi-active maintainer, not much community support
3. code structure not compatible with added encryption without major refactoring

Current status

Code at: `https://github.com/podhrmic/openLRSng`

1. Complete encryption algorithm
2. RNG for embedded devices
3. Proof-of-concept arduino sketch
4. Initial refactoring work

Future work

Code at: `https://github.com/podhrmic/openLRSng`

1. Remove unused features (bind, etc.)
2. Major code refactoring
3. open-source TX HW
4. switch to `Arduino-Makefile`
5. formally verify Arduino Crypto Lib

Future work

Code at: `https://github.com/podhrmic/openLRSng`

Help welcome!