SPACECOP

System for Performance Assessment in Challenging Environments of Cryptographic Operations and Protocols

Jacob Appelbaum, Eindhoven University of Technology (TU/e)
Daniel J. Bernstein, University of Illinois Chicago (UIC)
Tanja Lange, Eindhoven University of Technology (TU/e)

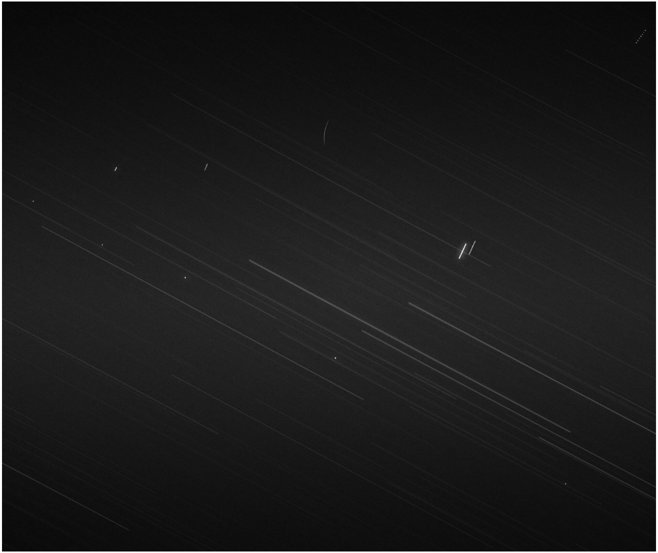

Figure: Photo credit: Dr. Trevor Paglen. Pictured: Thuraya and a companion

Our contribution: SPACECOP

- A new tool: `spacecop`
- `spacecop` predicts the performance of post-quantum cryptography in space
- `spacecop` generates reports based on user supplied cryptographic choices

Our contribution: SPACECOP

- A new tool: `spacecop`
- Cost analysis using structured decomposition including:
  - Environmental constraints such as radio channel bandwidth and latency
  - Example: LEO is 10-30ms rtt, 100Mb/s to 1000Mb/s
  - Size in bytes and round-trip considerations
  - Example: Estimate if extra costs due to post-quantum cryptographic primitives PQC are affordable and if so, for which system
  - Calculate computational costs of individual operations at relevant API levels
  - Example: generating key pairs, generating ciphertexts, performing encapsulation, etc

Our contribution: SPACECOP

All Constructions Are Benchmarkable:

- Designed to assist analysis, practical systems integrators, cryptographic protocol designers, and other users.
- We consider a **protocol** in a user-defined **scenario** with environmental time and latency considerations decomposed into cryptographic primitive operations with measurements running on selected **processor**
- We analyze post-quantum cryptographic primitives such as KEMs and signatures.
- We measure a variety of CPUs as selected including `arm32`, `arm64`, `amd64`, and `sparcv8`. Additional CPU support exists and is expanding; gathering your own data is supported but not required

Our contribution: SPACECOP

- We automate report generation of the chosen measurement matrix over implementations (e.g.: generic, optimized, etc), compilers, CPU architectures, and more
- Easy to use: simple configuration to generate reporting for your space ***scenarios***, your specific ***processors***, and customizable ***protocols***
- How easy to use?
  - ***Protocols***: protocol modeling by writing down the steps or as in a Noise-style
  - ***Processors***: select the CPUs of interest
  - ***Scenarios***: describe the context as a scenario
- Produce a comprehensive report for your selections by running `spacecop`
- Picking protocols, processors, and scenarios are easy configuration options
- There are additional advanced usage considerations

Figure: Photo credit: (Unknown; USA-207) 2010; Dr. Trevor Paglen. PAN also known as *Palladium at Night* or *NEMESIS* has been written about by Dr. Marco Langbroek and by others.

Protocols: SPACECOP

Protocols:

- Use one of the provided protocol definitions, or ...
- Describe a protocol:
  - ... give it a name
  - ... select the cryptographic primitives to compare
  - ... describe it as a protocol narration

Protocol: a-minimal

```
# minimal protocol
# no authentication of C
# no forward secrecy

use kem

# in advance, S computes and provides a KEM key:
S:               Spk,Ssk = kem.keypair()
S:               send Spk

---

# online phase, what the cost table is measuring:
C: c,result = kem.enc(Spk)
C: send c
S:               result = kem.dec(c,Ssk)

# now C and S have the same result, a shared session key
```

Protocols: SPACECOP

KEMs:

- bikel1
- bikel3
- hqc128
- hqc192
- hqc256
- kyber1024
- kyber512
- kyber768
- mceliece348864f
- mceliece460896f
- mceliece6960119f
- ntruhps2048677
- ntruhps4096821
- ntruhrss701
- sikep434
- sntrup1277
- sntrup761

Selecting other KEMs within those that have been benchmarked is as easy as adding one line to a text file:

```
crypto_kem sntrup1277
```

Protocols: SPACECOP

Signatures:

- `dilithium2`
- `dilithium3`
- `dilithium5`
- `falcon1024dyn`
- `falcon1024tree`
- `falcon512dyn`
- `falcon512tree`
- `sphincsf128shake256simple`
- `sphincsf192shake256simple`
- `sphincsf256shake256simple`
- `sphincss128shake256simple`
- `sphincss192shake256simple`
- `sphincss256shake256simple`

Selecting other signature systems within those that have been benchmarked is as easy as as adding one line to a text file:

```
crypto_sign ed25519
```

Protocols: SPACECOP

Included protocols:

- `a-minimal`
- `b-twoway`
- `c-3kem`
- `n-concise`
- `x-signed`

Adding additional protocols for analysis is straightforward and easy in `spacecop`. The `n-concise` is an example in a Noise-protocol framework style.

Protocol: b-twoway

```
use kem
use hash

C: Cpk,Csk = kem.keypair()
C: send Cpk
S:             Spk,Ssk = kem.keypair()
S:             send Spk
C+S:      pkhash = hash(Cpk,Spk)

---

C: c1,k1 = kem.enc(Spk)
C: send c1
S:             k1 = kem.dec(c1,Ssk)
S:             c2,k2 = kem.enc(Cpk)
S:            result = hash(pkhash,c1,c2,k1,k2)
S:             send c2
C: k2 = kem.dec(c2,Csk)
C: result = hash(pkhash,c1,c2,k1,k2)
```

Protocol: c-3kem (0)

```
use hash
use aead
use kem
use ekem

# pre-shared key and general setup:
C: C_key = randombytes(32)
C: send C_key
S:            S_key = randombytes(32)
S:            send S_key

C+S:      psk = hash(C_key,S_key)
C+S:      h_0 = hash('example c-3kem protocol token')
C+S:      k_0 = hash(psk) # simplification
C+S:      nonce0 = '000000000000'
C+S:      nonce1 = '000000000001'

---

# long-term identities:
C: C_pk,C_sk = kem.keypair()
C: send C_pk
S:            S_pk,S_sk = kem.keypair()
S:            send S_pk

---
```

Protocol: c-3kem (1)

```
C: h_1 = hash(h_0,c_0)
C: k_1 = hash(k_0,S_k) # simplification
C: E_pk,E_sk = ekem.keypair()
C: c_1 = aead.enc(E_pk,h_1,nonce0,k_1)
C: h_2 = hash(h_1,c_1)
C: send c_0,c_1

# receive_1:
S:            S_c = aead.dec(c_0,h_0,nonce0,k_0)
S:            S_k = kem.dec(S_c,S_sk)
S:            h_1 = hash(h_0,c_0)
S:            k_1 = hash(k_0,S_k) # simplification
S:            E_pk = aead.dec(c_1,h_1,nonce0,k_1)
S:            h_2 = hash(h_1,c_1)

# send_2:
S:            E_c,E_k = ekem.enc(E_pk)
S:            c_2 = aead.enc(E_c,h_2,nonce1,k_1)
S:            h_3 = hash(h_2,c_2)
S:            k_2 = hash(k_1,E_k)
S:            C_c,C_k = kem.enc(C_pk)
S:            c_3 = aead.enc(C_c,h_3,nonce0,k_2)
S:            h_4 = hash(h_3,c_3)
S:            k_3 = hash(k_2,C_k)
S:            c_4 = aead.enc('',h_4,nonce0,k_3)
S:            h_5 = hash(h_4,c_4)
```

Protocol: c-3kem (2)

```
S:              send c_2,c_3,c_4

# receive_2:
C: E_c = aead.dec(c_2,h_2,nonce1,k_1)
C: E_k = ekem.dec(E_c,E_sk)
C: h_3 = hash(h_2,c_2)
C: k_2 = hash(k_1,E_k) # simplification
C: C_c = aead.dec(c_3,h_3,nonce0,k_2)
C: C_k = kem.dec(C_c,C_sk)
C: h_4 = hash(h_3,c_3)
C: k_3 = hash(k_2,C_k) # simplification
C: n = aead.dec(c_4,h_4,nonce0,k_3)
C: assert n == ''
C: h_5 = hash(h_4,c_4)

# send_3:
C: c_5 = aead.enc('',h_5,nonce1,k_3)
C: result = hash(k_3,c_5) # simplification
C: send c_5

# receive_3:
S:              n = aead.dec(c_5,h_5,nonce1,k_3)
S:              assert n == ''
S:              result = hash(k_3,c_5) # simplification
```

Protocol: n-concise

```
use kem
use ekem
use aead
use hash

C+S: result = hash('example n-concise protocol token')
-> psk
<- psk

---

-> kempk
<- kempk

---

-> kemct ekempk
<- kemct ekemct confirm
-> confirm
```

Protocol: x-signed

```
use kem
use sign

# one-time setup of long-term identities
S:              Sidpk,Sidsk = sign.keypair()
S:              send Sidpk
C: Cidpk,Cidsk = sign.keypair()
C: send Cidpk
C+S:      idhash = hash(hash(Sidpk),hash(Cidpk))

---

# periodic broadcasts
S:              encpk,encsk = kem.keypair()
S:              signedencpk = sign(encpk,Sidsk)
S:              send signedencpk
C: encpk = sign.open(signedencpk,Sidpk)
C+S:      pkhash = hash(hash(encpk),idhash)

---

# online key exchange
C: c,k = kem.enc(encpk)
C: signedc = sign(c,Cidsk)
C: send signedc
S:              c = sign.open(signedc,Cidpk)
S:              k = kem.dec(c,encsk)
C+S:      result = hash(k,pkhash,hash(signedc))
```

Figure: Photo credit: NSA/GCHQ Surveillance Base, Bude, Cornwall, UK, 2014; Dr. Trevor Paglen

Processors: SPACECOP

Processors:

- Choose the processor for the mission control
- Choose the processor for the satellite or another orbital object
- Gathering data on large and small devices in a systematic and rigorous manner
  - Don't have a specific hardware device? Cryptographic measurement with `EMUCOP`
  - Easy microbenchmarking over the network made easy
  - Device access is no problem? The cryptographic measurement process runs locally or remotely by `ssh`.
  - Physical device measurement instruction and cycle counting with `SUPERCOP` if the underlying Operating System (OS) supports it for the specific CPU under test

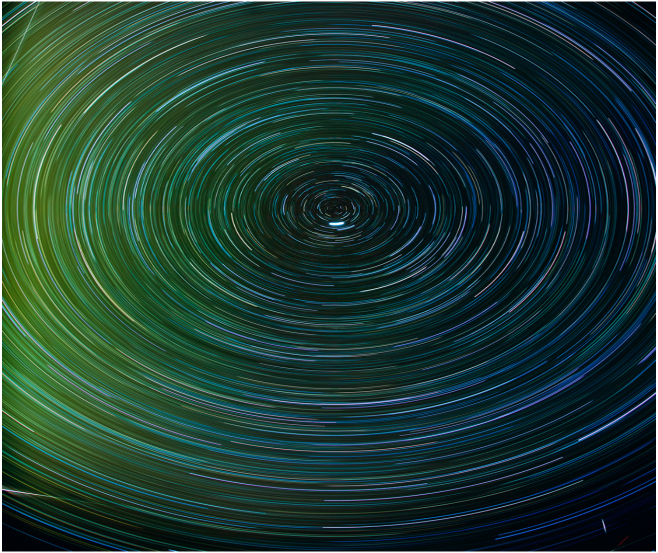

Figure: Photo credit: Singleton/SBWASS-R1 and Three Unidentified Spacecraft (Space Based Wide Area Surveillance System; USA 32), 2012; Dr. Trevor Paglen

Scenarios: SPACECOP

Scenarios:

- Use one of the provided scenario definitions, or ...
- Describe a scenario:
  - ... give it a name
  - ... adjust the space environment
  - ... describe the participants, their CPU(s), and their bandwidth constraints

Scenarios: SPACECOP

Included scenarios:

- `01-ops-sat`
- `02-starlink`
- `03-galileo`
- `04-mtg-s1`
- `05-ariel`

Adding additional scenarios for analysis which are already well defined is straightforward. Describing well understood scenarios as new scenarios in `spacecop` is easy.

Scenarios: 01-ops-sat

```
# only 515km above ground
# so allowed very low latency depending on ground location
rtt 30ms

C send 256 kbps
S send 1 Mbps
# https://esoc.esa.int/content/ops-sat

S cpuspeed 800MHz
S cpu armeabi-berry2
# actual CPU is a cortex-a9, so somewhat more powerful

# assume high-powered server at mission control
C cpu amd64-samba
C cpuspeed 3GHz
```

Scenarios: 02-starlink

```
# typical leo latency
rtt 40ms

C send 10Mbps
S send 100Mbps

C cpu aarch64-pi3aplus
C cpuspeed 1ghz

S cpu aarch64-pi3aplus
S cpuspeed 100mhz
```

Scenarios: 03-galileo

```
# galileo is 23222km above surface, but not always straight up
rtt 170ms

# https://space.oscar.wmo.int/satellites/view/mtg_s1
C send 2 kbps
S send 7.164 kbps

S cpu sparcv8-gr740
S cpuspeed 100MHz

# assume high-powered server at mission control
C cpu amd64-rome0
C cpuspeed 2.245GHz
```

Scenarios: 04-mtg-s1

```
# geosynchronous is 35786km above equator (120ms one-way)
# but latency is higher at non-equator regions
rtt 250ms

# https://space.oscar.wmo.int/satellites/view/mtg_s1
C send 2 kbps
S send 7.164 kbps

S cpu sparcv8-gr740
S cpuspeed 100MHz

# assume high-powered server at mission control
C cpu amd64-rome0
C cpuspeed 2.245GHz
```

Scenarios: 05-ariel

```
# ariel will be at l2 (1.5 million kilometers away from Earth)
# light travels 0.3 million kilometers per second
rtt 10 s

C send 16 kbps
S send 26 kbps

# various ariel documents say gr712 at 100mhz
# which should be fairly close to gr740 at 100mhz
S cpu sparcv8-gr740
S cpuspeed 100MHz

# assume high-powered server at mission control
C cpu amd64-samba
C cpuspeed 3GHz
```

Report generation

`SPACECOP` provides automatically generated numerical analysis of each protocol in each scenario for all permutations of the desired cryptographic primitives. The output makes a visual distinction between input from the user and output of the analysis based on that input.

How easy is the reporting?

One command produces a pdf with the selected **protocols** running on the selected **processors** for any number of **scenarios**.

Reporting

3.4 Protocol c-3kem cost estimates for scenario 02-starlink

Protocol: c-3kem	protocols/c-3kem
Scenario name: 02-starlink	scenarios/02-starlink
Round-trip time: 40.000 ms	rtt 40.000 ms
Mission-control CPU: aarch64-pi3aplus	C cpu aarch64-pi3aplus
Mission-control CPU speed: 1.000 GHz	C cpuspeed 1.000 GHz
Mission-control outgoing network (uplink) speed: 10.000 Mbps	C send 10.000 Mbps
Satellite CPU: aarch64-pi3aplus	S cpu aarch64-pi3aplus
Satellite CPU speed: 100.000 MHz	S cpuspeed 100.000 MHz
Satellite outgoing network (downlink) speed: 100.000 Mbps	S send 100.000 Mbps

kem, ekem, sign, aead, hash	total sec	C send bytes	C send sec	C CPU kcycles	C CPU sec	S send bytes	S send sec	S CPU kcycles	S CPU sec	
sikep434,	2.111455			206866	0.206866			178385	1.783850	StQ_1
sikep434, -,	2.112315	724	0.000579	206958	0.206958	740	0.000059	178470	1.784703	StQ_2
aes256gcmv1, sha256	2.113292			207064	0.207064			178568	1.785677	StQ_3
sikep434,	2.111419			206862	0.206862			178382	1.783817	StQ_1
sikep434, -,	2.112279	724	0.000579	206954	0.206954	740	0.000059	178467	1.784671	StQ_2
aes256gcmv1, sha3256	2.113256			207061	0.207061			178564	1.785644	StQ_3
sikep434,	1.562544			156958	0.156958			128485	1.284846	StQ_1
sikep434, -,	1.562903	772	0.000618	157002	0.157002	788	0.000063	128520	1.285197	StQ_2
hs1sivhiv2, sha256	1.563369			157064	0.157064			128566	1.285659	StQ_3
sikep434,	1.562546			156958	0.156958			128485	1.284847	StQ_1
sikep434, -,	1.562904	772	0.000618	157002	0.157002	788	0.000063	128520	1.285199	StQ_2
hs1sivhiv2, sha3256	1.563370			157064	0.157064			128566	1.285661	StQ_3
sikep434,	2.663887			274509	0.274509			226723	2.267233	StQ_1
bikel1, -,	2.666969	1935	0.001548	274843	0.274843	1967	0.000157	227019	2.270187	StQ_2
aes256gcmv1, sha256	2.670255			275227	0.275227			227325	2.273249	StQ_3

SPACECOP: advanced considerations

- `spacecop` is implemented in /bin/sh, C, Python, and with a little LaTeX
- `spacecop` to be released as Free/Libre Open Source Software (FL/OSS) and it optionally uses `qemu`, `dietlibc`, and other fine FL/OSS software packages
- Extending `spacecop` is straightforward

Questions?

Figure: Photo credit: They watch the moon, 2010; Dr. Trevor Paglen. Additional relevant works of art by Dr. Paglen are on display in Earth's top museums and on the world wide web