Testing of SPHINCS, RSA, and ECDSA

In this section we will be looking at the performance of SPHINCS, RSA, and ECDSA. These three cryptosystems each sign and verify messages in different ways. Below goes into a quick overview on how each system works. Below is a summary of each the cryptosystems we will be looking at. In the next section we will look at how these three cryptosystems match up against each other.

SPHINCS

SPHINCS makes use of multiple trees to obtain a few time signature (FTS) scheme. This is done using a mixture of Winternitz one time signatures + (WOTS+) and HORS with trees (HORST). The idea of SPHINCS uses a hyper-tree which is borrowed from an idea by Goldreich, which turns a stateful scheme into a stateless scheme. WOTS+ eventually feeds into multiple HORST in the end, which allows for extending signatures to be multi-use instead of single use. SPHINCS has three steps: Key generation, signature generation and signature verification.

RSA

RSA is used for signing messages by using public-key cryptography. It is an asymmetric cryptosystem that is based on the difficulty of factoring two large prime numbers. With RSA there are four steps we go through: key generation, key distribution, encryption and decryption.

ECDSA

ECDSA uses elliptic-curve cryptography to obtain the needed parameters. Key generation in ECDSA requires less bits compared to DSA to obtain the same amount of security. The signature size requires the same amount of bit in both ECDSA and DSA to obtain the same amount of security. ECDSA contains three steps: Key generation, signature generation, and signature verification.

Benchmarks for SPHINCS, RSA, and ECDSA

Benchmarking is done on Arch Linux (4.4.1-2-ARCH), processor: Intel Quad-Core i7-4710HQ 2.50GHz, 8 gigabytes RAM). The testing for SPHINCS is

done using a python implementation that is not designed for optimization. This implementation was meant to learn and understand how individual pieces of SPHINCS work [1]. There are optimized versions of SPHINCS available, but compiling the from these implementations are very complex. The testing for RSA and ECDSA is done using MIRACL crypto library, which is an optimized library for performance [2].

The benchmarking is ran using different files sizes, ranging from 10 Kb up to 100 Kb with file sizes being increased in 10 Kb increments. Below you will find two different tests for each cryptosystem, signing and verification.

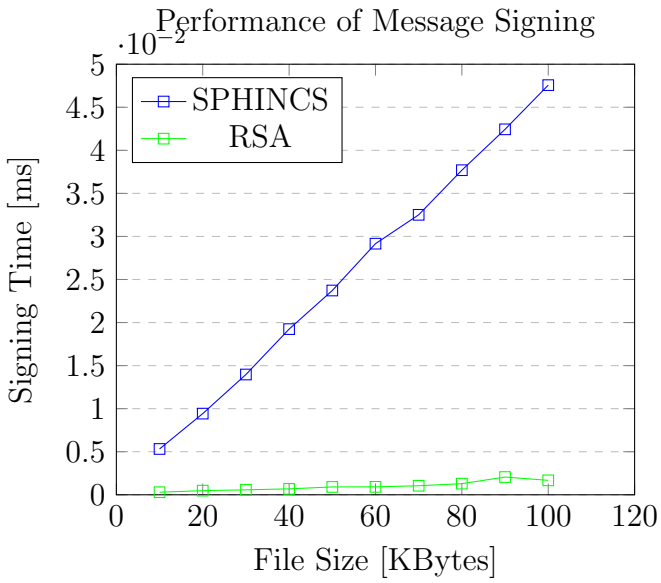
Table 1: Performance of message signing

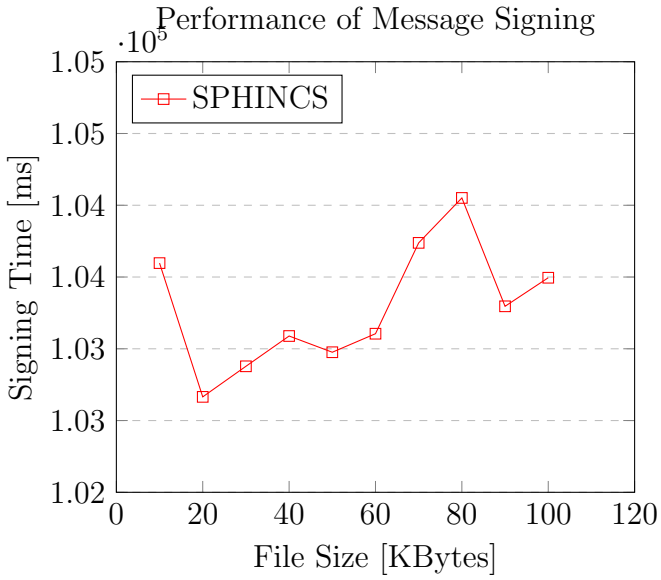
	SPHINCS	RSA	ECDSA
10 Kb	1m 43.597sec	0.005320ms	0.000299ms
20 Kb	1m 42.665sec	0.009426ms	0.000480ms
30 Kb	1m 42.878sec	0.013967ms	0.000574ms
40 Kb	1m 43.089sec	0.019238ms	0.000687ms
50 Kb	1m 42.976sec	0.023722ms	0.000911ms
60 Kb	1m 43.105sec	0.029159ms	0.000921ms
70 Kb	1m 43.738sec	0.032512ms	0.001048ms
80 Kb	1m 44.051sec	0.037693ms	0.001293ms
90 Kb	1m 43.296sec	0.042439ms	0.002060ms
100 Kb	1m 43.495sec	0.047571ms	0.001676ms

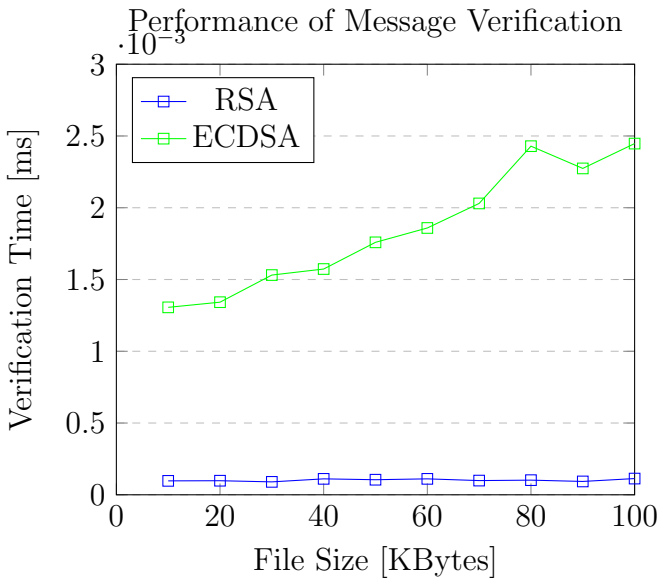
Table 2: Performance of message verification

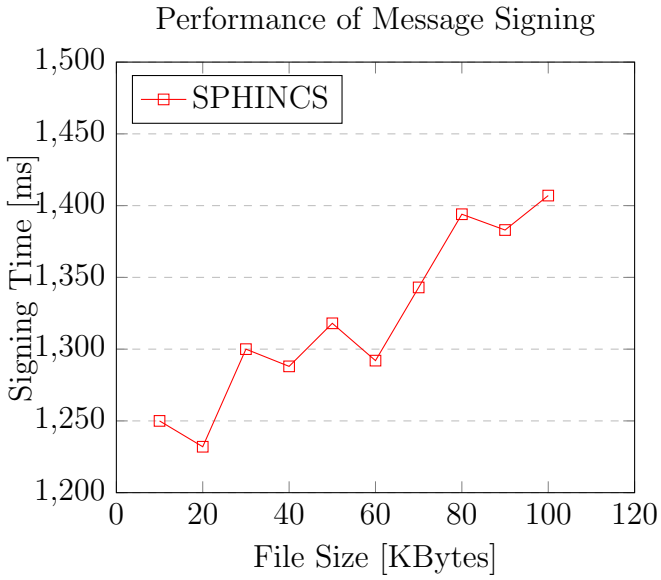
	SPHINCS	RSA	ECDSA
10 Kb	1.250sec	0.000097ms	0.001306ms
20 Kb	1.232sec	0.000098ms	0.001342ms
30 Kb	1.3sec	0.000090ms	0.001531ms
40 Kb	1.288sec	0.000111ms	0.001573ms
50 Kb	1.318sec	0.000105ms	0.001759ms
60 Kb	1.292sec	0.000111ms	0.001859ms
70 Kb	1.343sec	0.000099ms	0.002030ms
80 Kb	1.394sec	0.000102ms	0.002429ms
90 Kb	1.383sec	0.000093ms	0.002274ms
100 Kb	1.407sec	0.000113ms	0.002447ms

Below we will see the data put into another form which might be easier to comprehend the overall performance of each cryptosystem.


Performance of Message Signing
$\cdot 10^{-2}$
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
SPHINCS
RSA
Signing Time [ms]
0
20
40
60
80
100
120
File Size [KBytes]


Performance of Message Signing
$\cdot 10^{5}$
1.05
1.05
1.04
1.04
1.03
1.03
1.02
SPHINCS
Signing Time [ms]
0
20
40
60
80
100
120
File Size [KBytes]





NOTE: Due to SPHINCS performance, it was placed into its own graphs. If it were included in the same graphs with RSA and ECDSA, then they would not be visible.

Conclusion

I must stress, as stated in the previous section (Benchmarks), the SPHINCS implementation that was used for this was not optimized at all. The reason

for the use of this implementation of SPHINCS was due to ease of use. There are other optimized implementations available, but the complexity to use them is high.

With that being said, as you can see from the graphs above there are trade-offs with using any of these cryptosystems. As file sizes increase the time it takes to sign and verify messages will also increase.

Based on the implementations used for this benchmark, I would suggest using ECDSA for signatures. ECDSA performs faster than SPHINCS and RSA in terms of signing messages. As you can see though, RSA does verify messages faster than ECDSA, but if you look at the overall performance ECDSA is faster.

Works Cited

1) https://github.com/joostrijneveld/SPHINCS-py
2) https://github.com/CertiVox/MIRACL