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.

Problems with SUPERCOP

SUPERCOP[1] is a testing suite that is made by Daniel J. Bernstein and Tanja Lange. It contains multiple different types of crypto systems. The problem I was running into with SUPERCOP is that there is no way to

run individual testing without having to completely rip the program apart. Dependencies for the most part are in the folders they should be in, but there are a few which I could not find. I had a hard time trying to figure out how to pull out SPHINCS from SUPERCOP because of this problem. Also since SPHINCS could not be pulled from SUPERCOP, I also couldn't figure out how to pull out ED25519.

Solution to SUPERCOP problem

Eventually I did find an implementation that is based on Erlang. The files it uses are taken from SUPERCOP and the author was able to figure out how to make it work using Erlang[2]. Also not being able to use ED25519, I went with using ECDSA from MIRACL as it's the only ECDSA implementation that I have currently available.

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 Erlang implementation which uses code from SUPERCOP. The authors of the SPHINCS paper created SUPERCOP testing suite which benchmarks all different kinds of crypto systems. This implementation is used as trying to compile individual testing from SUPERCOP is very complicated. The testing for RSA and ECDSA is done using MIRACL crypto library, which is an optimized library for performance [3].

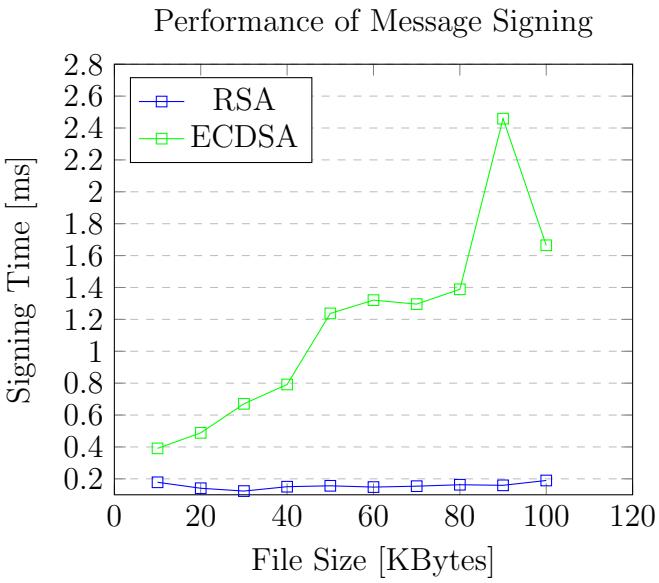
The benchmarking is ran using different files sizes, ranging from 10 Kb up to 100 Kb with file sizes being increased in 10 Kb increments. Below you will find two different tests for each cryptosystem, signing and verification.

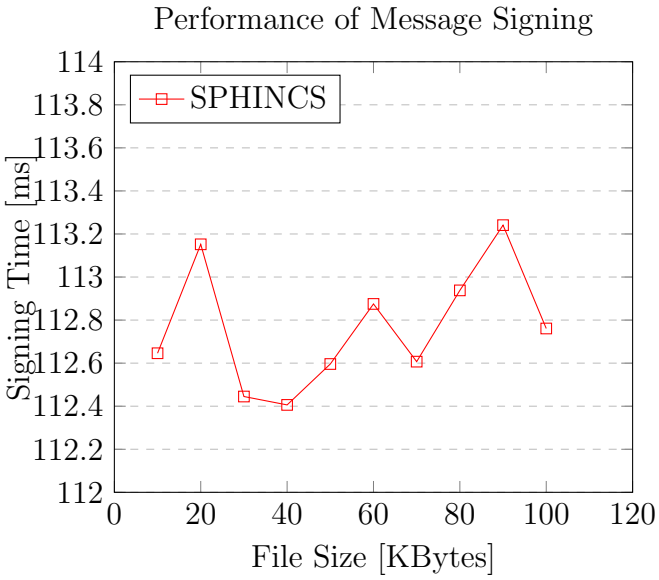
Table 1: Performance of message signing

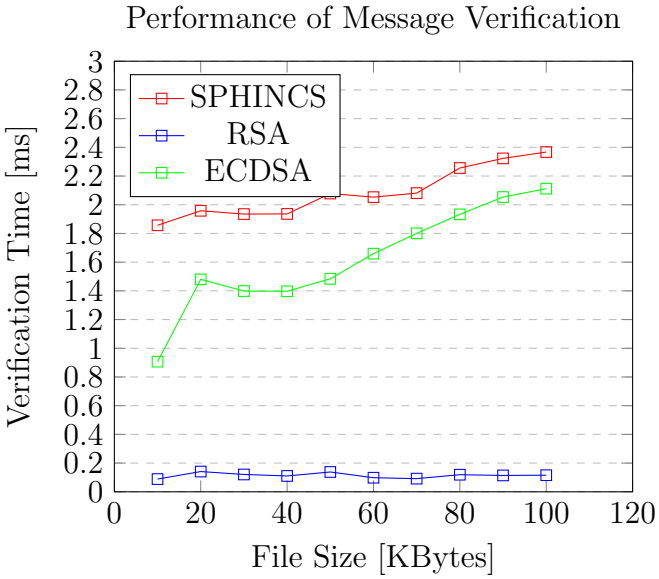
	SPHINCS	RSA	ECDSA
10 Kb	112.646ms	0.178332ms	0.390999ms
20 Kb	113.152ms	0.140999ms	0.488332ms
30 Kb	112.445ms	0.123333ms	0.670332ms
40 Kb	112.406ms	0.150666ms	0.791666ms
50 Kb	112.596ms	0.156332ms	1.237333ms
60 Kb	112.875ms	0.148333ms	1.320665ms
70 Kb	112.607ms	0.153999ms	1.295666ms
80 Kb	112.938ms	0.162999ms	1.388666ms
90 Kb	113.241ms	0.159333ms	2.458666ms
100 Kb	112.761ms	0.189666ms	1.664666ms

Table 2: Performance of message verification

	SPHINCS	RSA	ECDSA
10 Kb	1.857ms	0.088666ms	0.906633ms
20 Kb	1.958ms	0.140999ms	1.479999ms
30 Kb	1.935ms	0.120666ms	1.398666ms
40 Kb	1.936ms	0.110333ms	1.396999ms
50 Kb	2.078ms	0.137999ms	1.483999ms
60 Kb	2.053ms	0.097999ms	1.659333ms
70 Kb	2.081ms	0.091666ms	1.800666ms
80 Kb	2.256ms	0.118999ms	1.932999ms
90 Kb	2.323ms	0.113999ms	2.053666ms
100 Kb	2.367ms	0.115666ms	2.111999ms


Performance of Message Signing
Signing Time [ms]
File Size [KBytes]
RSA
ECDSA
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
20
40
60
80
100
120


Performance of Message Signing
Signing Time [ms]
File Size [KBytes]
SPHINCS
114
113.8
113.6
113.4
113.2
113
112.8
112.6
112.4
112.2
112
0
20
40
60
80
100
120



NOTE: Due to SPHINCS performance, signature generation was placed into its own graph.

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) http://bench.cr.yp.to/supercop.html
2) https://github.com/ahf/sphincs
3) https://github.com/CertiVox/MIRACL