The speed of digital signatures.

Author - Rodrigo Lima, 98475

DETI, University of Aveiro

December 29, 2022

Contents

1 Introduction

Digital signatures can serve a variety of purposes, including linking the identity of a signer to a document and authenticating communication endpoints. While the speed of creating and validating a digital signature may not be a major concern in the first case, it can be an influential factor in the second.

When a communication endpoint uses a digital signature to authenticate itself, it typically needs to do so each time a distinct endpoint initiates an interaction.

The time it takes to create and validate a digital signature can affect the performance of both parties involved in communication. This is because the choice of technology and implementation of the algorithm can significantly impact this time.

In this project, we aim to assess the performance of different digital signature algorithms implemented in various languages and potentially using different cryptographic libraries.

2 Development

In this research project, we aimed to assess the performance of various digital signature algorithms implemented in different programming languages and potentially utilizing different cryptographic libraries. To accomplish this, we focused on measuring the time required to create and validate signatures using the following variations:

- RSA signatures:
  - Utilizing key sizes of 1024, 2048, and 4096 bits.
  - Using PKCS #1 and PSS paddings, with a consistent configuration for PSS (MGF function, digest function, salt, label, etc.).
- ECDSA (Elliptic Curve Digital Signature Algorithm):
  - Employing three different curve types (e.g. NIST P, K(oblitz), and B curves).
  - Using three different curves allows for small, medium, and large keys.

For all signatures, we utilized the same digest function (e.g. SHA-256) to compute and validate the signatures. We did not include the time required to generate the digest of the signed data in our performance figures.

In total, we calculated $2 \times (3 \times 2 + 3 \times 3) = 30$ performance results per library, considering both signature generation and validation separately for each variation. This enabled us to compare the performance of the various algorithms and implementations and to gain an understanding of how they might be applied in different situations.

In order to ensure the validity of our performance assessments, we took care to utilize the same algorithms and keys for all programming languages under consideration. To be specific, the keys were generated using a program written in a single language and then reused in all programs, regardless of their language of implementation.

To acquire reliable results, we implemented a loop with a predetermined number of iterations, during which the same operation was executed a specific number of times. We measured the elapsed time required to execute these consecutive operations and selected the smallest time observed among the iterations. By dividing this time by the number of operations, we were able to calculate an upper bound for the time required for each individual operation.

It was important to choose a number of consecutive operations that was one order of magnitude greater than the measurement precision, but not so large that the elapsed time was significantly impacted by other computer operations such as interruptions or context switches. Additionally, we conducted the evaluations on a computer without a graphical interface in order to minimize the potential interference of external events on the accuracy of our measurements.

Overall, our approach allowed us to conduct a thorough evaluation of the performance of the various digital signature algorithms and gain an understanding of their potential applications in a range of situations.

3 Conclusion

3.1 Considerations

The signature algorithms tested included RSA schemes (PKCS1 and PSS) and elliptic curve schemes, with key sizes ranging from 1024 to 4096 bits for the RSA schemes and from 192 to 571 bits for the elliptic curve schemes.

The results of the study showed that the time required to sign and verify a signature varied depending on the specific algorithm and key size being used, as well as the programming language and implementation. In general, larger key sizes and more complex algorithms tended to require more time to sign and verify signatures, while faster hardware and more efficient software implementations tended to have shorter times.

It's worth noting that the times shown in the results are just rough estimates and will vary depending on the specific hardware and software being used. In this study, we measured the performance of various signature algorithms and implementations using a computer with the following hardware configuration:

- Processor architecture: x86_64
- CPU operation modes: 32-bit and 64-bit
- Processor model: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- Threads per core: 1
- Cores per socket: 1

This hardware configuration, specifically the 9th generation Intel Core i7 processor with a base clock speed of 2.60 GHz, likely had an impact on the times shown in the results. The specific hardware and software being used can have a significant impact on the performance of digital signature algorithms, so it's important to consider these factors when comparing the results of different studies.

3.2 Results

The results of this study were obtained by measuring the time taken to sign or verify a message using a specific algorithm and implementation in seconds.

The table in the results includes labels indicating the type of algorithm and key size used. The RSA padding schemes PKCS1 and PSS are utilized for RSA signature algorithms, with the key size specified in bits following the padding scheme label. For elliptic curve signature algorithms, the curve type (K, P, or B) and key size in bits are specified in the label.

It is important to note that the unit of time used to measure performance can influence the absolute values shown in the results, but should not affect the relative performance of different algorithms and implementations.

Language	Go		Python		Java	
Operation	Sign	Verify	Sign	Verify	Sign	Verify
PKCS1-1024	0.000282	0.000018	0.000087	0.000032	0.000211	0.000011
PKCS1-2048	0.001398	0.000046	0.000514	0.000046	0.001118	0.000037
PKCS1-4096	0.00777	0.000128	0.00341	0.000092	0.007085	0.000122
PSS-1024	0.000285	0.00002	0.000101	0.000042	0.000213	0.000012
PSS-2048	0.001352	0.000048	0.000539	0.000057	0.00112	0.000039
PSS-4096	0.007949	0.000139	0.003541	0.000104	0.007093	0.000127
P-192			0.000202	0.000211	0.000082	0.000083
P-224	0.000101	0.000316				
P-256	0.000024	0.00007	0.000025	0.000078	0.000143	0.000135
P-384	0.000347	0.001221				
P-521	0.000925	0.003587	0.000262	0.000515	0.000537	0.000677
K-163			0.000191	0.000391	0.000197	0.000125
K-283			0.000468	0.000926	0.000415	0.000402
K-571			0.001737	0.003415	0.001681	0.001747
B-163			0.000206	0.000418	0.000196	0.000089
B-283			0.000486	0.00097	0.000415	0.000277
B-571			0.001867	0.003663	0.001724	0.001497

Go is generally considered to be a faster programming language than Python and Java, due in part to its use of statically-typed compiled code and a number of other performance optimizations. However, the results shown indicated that Go was slower than Python and Java for signing and verifying digital signatures using both RSA and elliptic curve algorithms.

There are a few possible reasons for this discrepancy between the general performance characteristics of Go and the specific results shown in the study. One possibility is that the Go implementation did not make use of goroutines, which are a type of lightweight thread that can be used to improve the performance of concurrent operations in Go.

Another possibility is that the Python implementation made use of C libraries for some of the operations needed to sign and verify signatures, which

can potentially lead to faster performance compared to implementations in other languages. Similarly, the Java implementation may have had more efficient software libraries or other optimizations that helped to improve its performance relative to Go.

Overall, it's important to keep in mind that while Go is generally considered to be a fast language, the results shown in the study indicate that in this particular case, Python and Java were faster for signing and verifying digital signatures using both RSA and elliptic curve algorithms.

The results shown indicate that, in general, the elliptic curve signature schemes using the K curves were faster than those using the B curves for signing and verifying messages. This is likely due to differences in the specific algorithms and implementation techniques being used for each curve.

The results also show that, among the elliptic curve signature schemes tested, those using the P curves were generally faster than those using the K and B curves, particularly for the 521-bit key size. However, it's worth noting that the only point of comparison in this case is only one key size, so it's difficult to draw conclusions about the relative performance of the different curve types without considering other key sizes.

It is worth mentioning that the Python implementation of the elliptic curve signature schemes using the K and B curves took a significantly longer amount of time to verify messages when compared to the Java implementation. This discrepancy in performance could potentially be attributed to an anomaly in the specific software libraries or implementation techniques employed by the Python implementation.