An Implimentation of DSA

by

Maziar Kaveh

Electrical and Computer Engineering Dep. University of Stavanger (UiS)

m.kaveh@stud.uis.no

Abstract

People have traditionally used signatures as a means of informing others that the signature has read and understood a document. Digital signature in a document is bound to that document in such a way that altering the signed document or moving the signature to a different document invalidates the signature. This security eliminates the need for paper copies of documents and can speed the processes involving documents that require signatures. Digital Signatures are messages that identify and authenticate a particular person as the source of the electronic message, and indicate such person's approval of the information contained in the electronic message. Emerging applications like electronic commerce and secure communications over open networks have made clear the fundamental role of public key cryptosystem as unique security solutions. On the other hand, these solutions clearly expose the fact, that the protection of private keys is a security bottleneck in these sensitive applications. This problem is further worsened in the cases where a single and unchanged private key must be kept secret for very long time (such is the case of certification authority keys, and e-cash keys). They help users to achieve basic security building blocks such as identification, authentication, and integrity.

Keywords: DSA algorithm, Public key, Digital Signature, hash, SHA.

Introduction

The Digital Signature Standard, created by the NIST, specifies DSA as the algorithm for digital signatures and SHA-1 for hashing. DSA is for signatures only and is not an encryption algorithm, although Schneier describes encryption mechanisms (ElGamel encryption and RSA encryption) based on DSA. DSA is a public key algorithm; the secret key operates on the message hash generated by SHA-1; to verify a signature, one recomputed the hash of the message, uses the public key to decrypt the signature and then compare the results.

The key size is variable from 512 to 1024 bits which is adequate for current computing II. capabilities as long as you use more than 768 bits. Signature creation is roughly the same speed as with RSA, but is 10 to 40 times (Schneier) as slow for verification. However, these numbers depend partially on the assumptions made by the bench marker. Since verification is more frequently done than creation, this is an issue worth noting.

The only known cracks (forgery) are easily circumvented by avoiding the particular module (prime factor of p - 1 where p is the public key) that lead to weak signatures. Schneier states that DSS is less susceptible to attacks than RSA; the difference is that RSA depends on a secret prime while DSA depends on a public prime -- the verifier can check that the prime number is not a fake chosen to allow forgery. It is possible to implement the DSA algorithm such that a "subliminal channel" is created that can expose key data and lead to forgeable signatures so one is warned not to used unexamined code. A Digital Signature is a checksum which depends on the time period during which it was produced. It depends on all the bits of a transmitted message, and also on a secret key, but which can be checked without knowledge of the secret key. A major difference between handwritten and digital signatures is that a digital signature cannot be a constant; it must be a function of the document that it signs. If this were not the case then a signature, could be attached to any document. Furthermore, a signature must be a function of the entire document; changing even a single bit should produce a different signature.

A digital signature algorithm authenticates the integrity of the signed data and the identity of the signatory. A digital signature algorithm may also be used in proving to a third party that data was actually signed by the generator of the signature. Is intended for use in electronic mail, electronic data interchange, software distribution, and other applications that require data integrity assurance and data origin authentication. The wireless protocols, like HiperLAN and WAP have specified security layers and the digital signature algorithm have been applied for the authentication purposes.

Design and Implementation

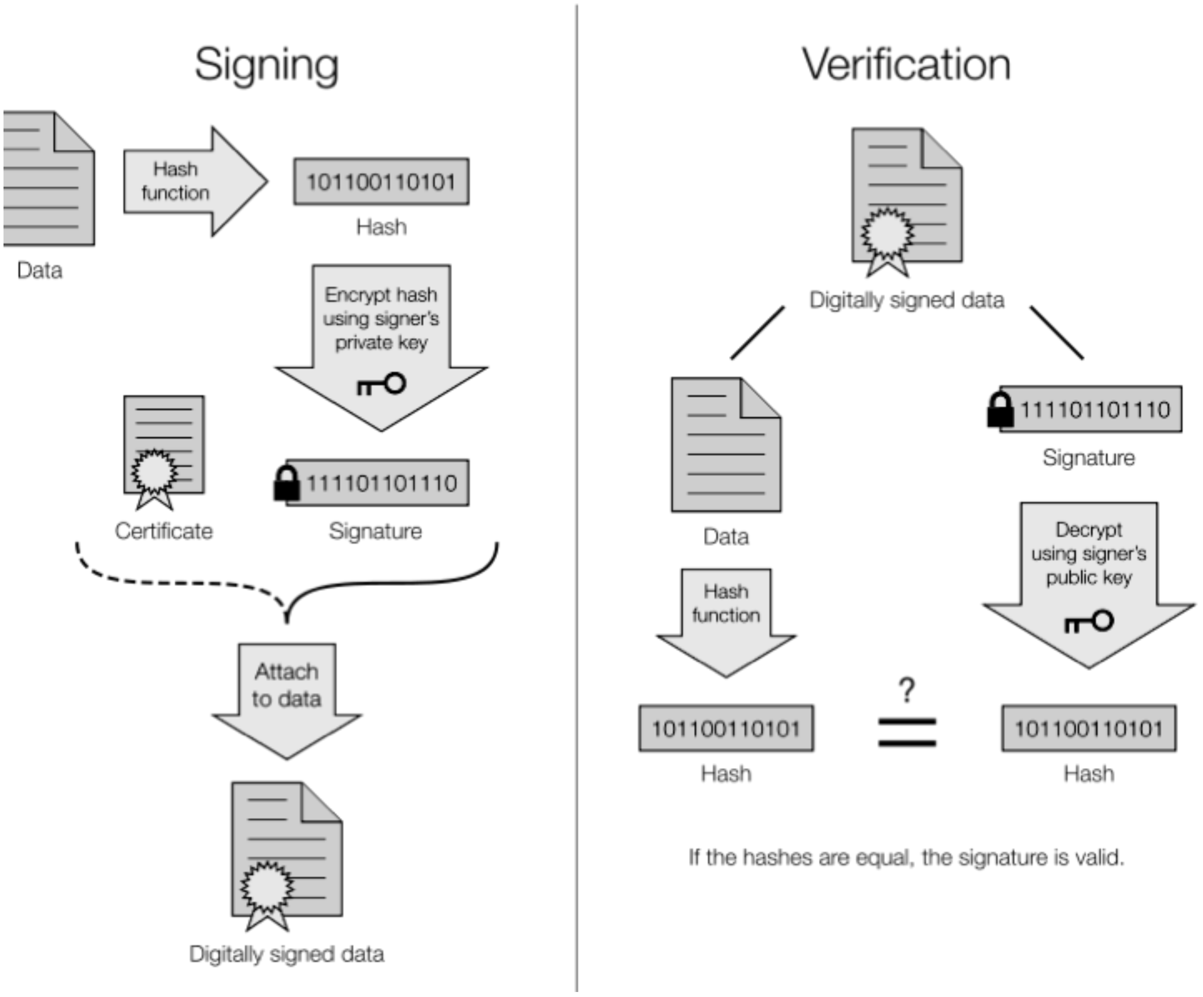
The term digital signature encompasses a great many variety of "signatures". Electronic signatures are simply an electronic confirmation of identity. This definition is deliberately broad enough to encompass all forms of electronic identification, from biometric signatures such as iris scans and fingerprints to non-biometric signatures, such as

Digital signatures. Electronic signatures can be further subdivided into the highly secure and the insecure. Digital signature must serve the same essential functions that we expect of documents signed by handwritten signatures, namely integrity, non- repudiation, authentication and confidentiality. In the digital realm, integrity means ensuring that a communication has not been altered in the course of transmission. It is concerned with the accuracy and completeness of the communication. The recipient of an electronic communication must be confident of a communication's integrity before she can rely on and act on the communication. Integrity is critical to e- commerce transactions, especially where contracts are formed electronically.

The process of digitally signing starts by taking a mathematical summary (called a hash code) of the check. This hash code is a uniquely-identifying digital fingerprint of the check. If even a single bit of the check changes, the hash code will dramatically change. The next step in creating a digital signature is to sign the hash code with your private key. This signed hash code is then appended to the check. How is this a signature? Well, the recipient of your check can verify the hash code sent by you, using your public key. At the same time, a new hash code can be created from the received check and compared with the original signed hash code. If the hash codes match, then the recipient has verified that the check has not been altered. The recipient also knows that only you could have sent the check because only you have the private key that signed the original hash code.

A digital signature is computed using a set of parameters and authenticates the integrity of the signed data and the identity of the signatory. An algorithm provides the capability to generate and verify signature. Signature generation makes use of a private key to generate a digital signature. Signature verification makes use of a public key, which corresponds to, but is not the same as, the private key. Each user possesses a private and public key pair. Public keys are assumed to be known to the public in general. Private keys are never shared. Anyone can verify the signature of a user by employing that user public key. Only the possessor of the user private key can perform signature generation.

A hash function is used in the signature generation process to obtain a condensed version of data, called a message digest. The message digest is then input to the digital signature algorithm to generate the digital signature. The digital signature is sent to the intended verifier along with the message. The verifier of the message and signature verifies the signature by using the sender's public key.



The same hash function must also be used in the verification process. The hash function is specified in a separate standard, the Secure Hash Standard, FIPS 180-1, and FIPS approved digital signature algorithms must be implemented with the Secure Hash Standard. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data. The Digital Signature Standard (DSS) uses three algorithms for digital signature generation and verification. The Digital Signature Algorithm (DSA), the RSA digital signature algorithm as defined in ANSI X9.31 and Elliptic Curve digital signature algorithm

(ECDSA) as define in ANSI.

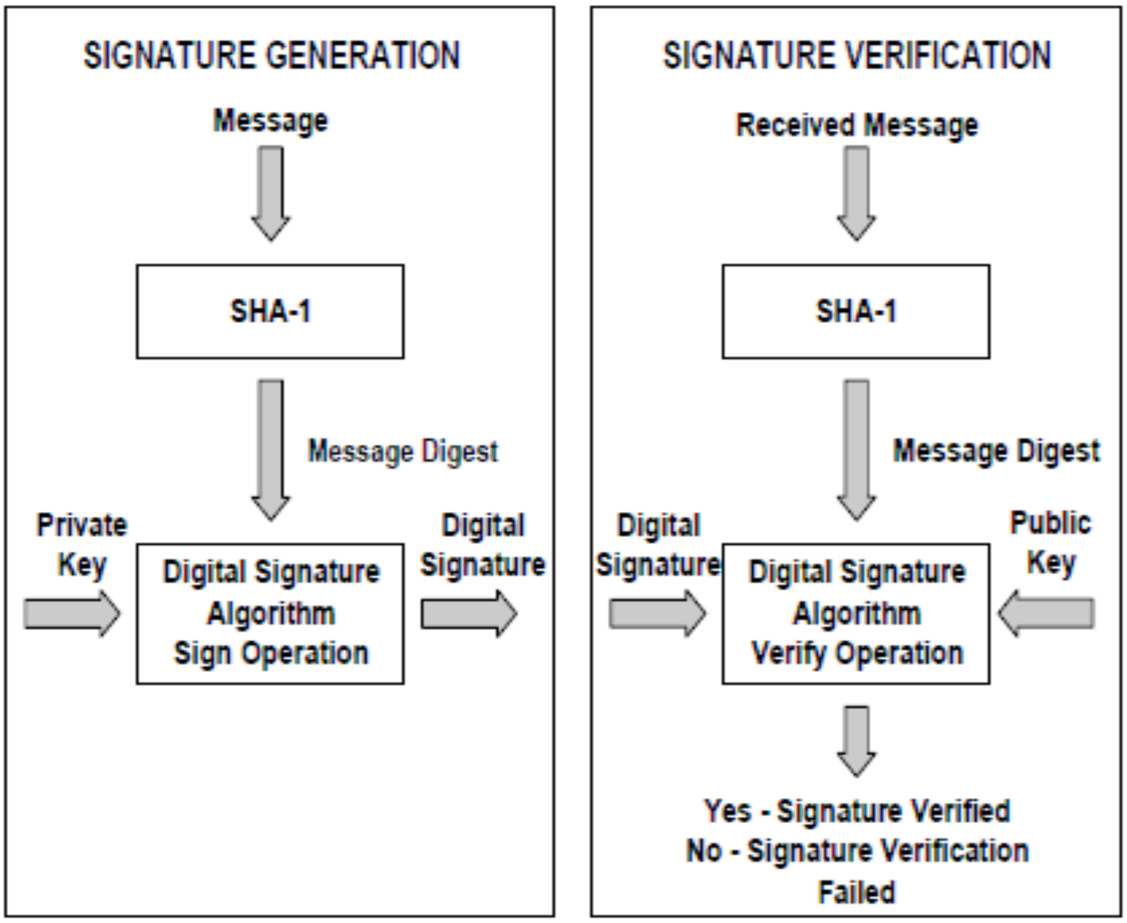
DSA DESCRIPTION

DSA Parameters

A DSA digital signature is computed using a set of domain parameters, a private key x, a per message secret number k, data to be signed, and a hash function.

These parameters are defined as follows:

- p a prime modulus, where $2^{L-1} < p < 2^{L}$, and L is the bit length of p.
- q a prime divisor of (p–1),where 2N–1<q< 2 N, and N is the bit length of q.
- g a generator of the subgroup of order q mod p, such that 1 < g < p.
- x the private key that must remain secret; x is a randomly or pseudo randomly generated integer, such that 0 < x < q, i.e., x is in the range [1, q–1].
- y the public key ,where $y=g^{x}$ mod p.
- k a secret number that is unique to each message; k is a randomly or pseudo randomly generated integer, such that 0 < k < q, i.e., k is in the range [1, q–1].



Key Pairs

Each signatory has a key pair: a private key x and a public key y that are mathematically related to each other. The private key will be used for only a fixed period of time in which digital signatures may be generated; the public key may continue to be used as long as digital signatures that were generated using the associated private key need to be verified.

Key generation has two phases: The first phase is a choice of algorithm parameters which may be shared between different users of the system:

· Choose an approved cryptographic hash function H.

· Decide on a key length L and N. This is the primary measure of the cryptographic strength of the key.

· Choose an N-bit prime q. N must be less than or equal to the hash output length.

· Choose an L-bit prime modulus p such that p–1 is a multiple of q.

· Choose g, a number whose multiplicative order modulo p is q. This may be done by setting g = h^ (p-1) /q mod p for some arbitrary h (1 < h < p-1), and trying again with a different h if the result comes out as 1.

The second phase computes private and public keys for a single user:

· Choose x by some random method, where 0 < x < q.

· Calculate $y=g^{x}$ mod p.

· Public key is (p, q, g, y). Private key is x.

Signing

Let N be the bit length of q. Let min (N, outlen) denote the minimum of the positive integers N and outlen, where outlen is the bit length of the hash function output block.

The signature of a message M consists of the pair of numbers r and s that is computed according to the following equations:

· r=(g^{k} mod p) mod q.

· z = the leftmost min (N, outlen) bits of Hash(M).

·s= (k−1 (z + xr)) mod q.

When computing s, the string z obtained from Hash(M) will be converted to an integer.

DSA Signature Verification and Validation

Signature verification may be performed by any party using the signatory’s public key. A signatory may wish to verify that the computed signature is correct, perhaps before sending the signed message to the intended recipient. The intended recipient verifies the signature to determine its authenticity.

The signature verification process is as follows:

1. The verifier shall check that $0 < r' < q$ and $0 < s' < q$; if either condition is violated, the signature shall be rejected as invalid.

2. If the two conditions in step 1 are satisfied, the verifier computes the following: w = (s′)–1 mod q. z = the leftmost min(N, outlen) bits of Hash(M′). u1 = (zw) mod q. u2 = ((r′)w) mod q. v = (((g)u1 (y)u2) mod p) mod q.

3. A technique is provided in Appendix C.1 for deriving (s′)–1 (i.e., the multiplicative inverse of s′ mod q).The string z obtained from Hash(M′) shall be converted to an integer. If v = r′, then the signature is verified. For a proof that v = r′ when M′ = M, r′ = r, and s′ = s,

4. If v does not equal r′, then the message or the signature may have been modified, there may have been an error in the signatory’s generation process, or an imposter may have attempted to forge the signature. The signature will be considered invalid. No inference can be made as to whether the data is valid, only that when using the public key to verify the signature, the signature is incorrect for that data.

Code

For downloading the code you can go to

https://github.com/maziarkaveh/security

Or clone the source with git using this URL

git clone ***https://github.com/maziarkaveh/security***

You need to go to Oracle website to download Java JDK for compiling the source code.

After downloading and installing JDK you may set JAVA_HOME environment variable to the path that JDK has been installed.

Use

```
java –version
```

To check your Java installed correctly.

For building the project you need to download and install in your hard drive apache Maven project.

Again you can use

```
mvn  -version .
```

To install the project you just need to go to the root of source code which pom.xml located and type

```
mvn install
```

Whole source codes will be compiled and unit tests will run after compiling the jar files will be copied into target folder.

By running run.sh in UNIX environment or run.bat in windows the command user interface will come and you can input your data.

In this project tried to used Object Oriented and Domain Driven Design concepts, Dependency injection using Spring framework, ORM using JPA, Hibernate as vendor,H2 a lightweight database, JUnit and Spring test framework were used for unit and integration testing of the project which was tried to developed base on TDD, Finally SLF4J over LOG4j was used for logging.

There is an interface in the project named **HashService** which has a method that gets a string or array of bytes and digest it to 160 bits hash code. SHA-1 is implemented in this service.

We have 4 entity models for this project which all are able to persist, we used JPA2 to persist them into database (H2 database which is embedded in project)

1. **GlobalPublicKey**

   This model is able to generate and hold our P , Q, G variables for global public key.

2. **User**
   We declare an entity that the signature can be assigned to. It has capability to be extended later for authentication and maybe authorization by adding Role.

3. **UserKeys**
   Each user can have one or many public and private keys with different global public keys. This object generates private and public key using User and GlobalPublicKey objects.

4. **Message**

Message is the entity that a user creates the message and is able to sign it using GlobalPublicKey and user public key. This model also has a verify method that a boolean returns true if the message signed by the user and false if it is altered.

Test results

```
mvn test
```

Runs all tests

All dependency libraries were added using maven to the project.

We have command line interface for working and testing manually, for testing purpose first we have to add some users, then global public keys and user keys, finally we can create new message and sign by persisted user keys.

Discussion:

I was trying to implement standard DSA using SHA-1.After creating and signing a message with some user, any changes to stored message, S or R will be detected by verify method in Message object. However the way private and public keys are stored in database is not good solution since any body that has access to databases can obtain users private and public key pairs. Just owner user should access private keys. Another issue might be that user authentication is not implemented in project so that every body can generate new public key or can sign on behalf of others. An authentication and authorization feature should be added so each user can sign its own messages however user can verify all other messages. Also manipulating global public and generating key should be done just by admin role.

Conclusion

Digital signatures utilizing the public key cryptography system have every potential to achieve the same level of legal recognition as handwritten signatures. However, the main obstacle at present is in the functional element of non-repudiation. This element, unlike the other three elements of handwritten signatures discussed, cannot be achieved by technology alone. Assistance is required from the law to help it attain the functional element of non-repudiation. Once non-repudiation has been achieved, then and only then, can electronic commerce be expected to be successfully taken up. A certification authority in turn can be validated by higher certification authorities, thus creating a certificate chain. Hence, the trustworthiness of a certification authority may depend on its reputation in traditional business transactions, or, it may be a subscriber of a higher certification authority, and use the certificate of the higher certification authority to reassure subscribers and relying parties that it is not a bogus certification authority. The certification authority at the pinnacle of the certification authority hierarchy is known as a root certification authority and it issues root certificates. The root certification authority self- authenticates for purposes of determining the validity of the certificates.

References

1. -William-Stallings "Cryptography and Network Security".
2. Digital signature Algorithm Gunjan Jain ISSN : 2319-8257)
3. S. W. Changchien and M. S. Hwang, "A batch verifying and detecting multiple RSA digital signatures," International Journal of Computational and Numerical Analysis and Applications, vol. 2, no. 3, pp. 303-307, 2002.
4. Dorothy E. R. Denning, Cryptography and Data Se curity. Massachusetts: Addison-Wesley, 1982.
5. T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms" IEEE Transactions on Information Theory, vol. IT-31, pp. 469-472, July 1985.
6. L. Harn, "Batch verifying multiple DSA-type digital signatures" Electronics Letters, vol. 34, no. 9, pp. 870-871, 1998.
7. L. Harn, "Batch verifying multiple RSA digital signatures" Electronics Letters, vol. 34, no. 12, pp. 1219- 1220, 1998.
8. M. S. Hwang, I. C. Lin, and K. F. Hwang, "Cryptanalysis of the batch verifying multiple RSA digital signatures," Informatica, vol. 11, no. 1, pp. 15-19, 2000.
9. M. S. Hwang, C. C. Chang, and K. F. Hwang, "An ElGamal-like cryptosystem for enciphering large messages," IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 2, pp. 445-446, 2002.
10. http://www.h2database.com/html/quickstart.html
11. http://www.hibernate.org/
12. http://www.springsource.org/about
13. http://maven.apache.org/
14. http://junit.org/