CS463
Network Security

CWE-347: Improper Verification of Cryptographic Signature

Dept. of Computer Science and Engineering, National Institute of Technology

Submitted by

Rutwik Mulay 181CO144
VIII SEM B.Tech.

Contents

Introduction

A digital signature is a cryptographic output used to verify the authenticity of data. Digital signatures rely on asymmetric cryptography, also known as public-key cryptography. An asymmetric key consists of a public/private key pair. The private key is used to create a signature, and the corresponding public key is used to verify the signature.

In the evolving world, it is very well recognized that digital signatures play a very important role in the legal and commercial aspects. A complete signature method has the capacity to prevent misbehaviours such as repudiation, forgery of sensible information, and so on, by the signer or opponents.

As discussed above Digital signatures work using public-key cryptography. The private key encrypts the data and is available only to the signer. The public key decrypts the data pertaining to the digital document and is given to the receiver. However, both parties must have a registered digital certificate from an issuing certificate authority to connect the signer and their signature. Public key cryptography ensures the security, accuracy, and authenticity of the document. Encryption is the process of encoding data sent to the receiver in a form that can only be decoded by the receiver. Authentication is the process of

validating that the information from the sender is genuine and has not been altered in transit.

Agreements and transactions that were once signed on paper and delivered physically are now being replaced with fully digital documents and workflows as more businesses are conducted online. Malicious actors who want to steal or manipulate data for their own gain are often present whenever precious or sensitive data is shared. To minimize the risk of document tampering by malicious parties, businesses must be able to check and authenticate that these critical business documents, data, and communications are trusted and delivered securely.

In addition to protecting sensitive online data, digital signatures do not impede the effectiveness of online document workflows; in fact, when compared to paper processes, they often help improve document management. When digital signatures are in place, signing a document becomes simple and can be done on any computer or mobile device. And, since the digital signature is embedded in the file, it can be used anywhere it is transmitted and on any device. By providing the status of all documents, determining whether or not they've been signed, and watching an audit trail, digitally signed documents are also simple to control and keep track of.

Problem Statement

CWE-347: Improper Verification of Cryptographic Signature

The software does not verify or incorrectly verifies, the cryptographic signature for data.

It's critical to include unforgeability, non-transferability, invisibility, and zero-knowledge in the algorithm design of designated confirmer signatures in order to create proofs that meet security standards like unforgeability, non-transferability, invisibility, and zero-knowledge.

As discussed above, digital signatures rely on public and private keys. Those keys have to be guaranteed to ensure safety and to avoid forgery or malicious use. When you send or sign a document, you require assurance that the documents and the keys are created securely and that they are using legitimate keys.

It is possible that the user’s signature can be misused. There are malevolent websites that try to catch signatures under the mask of ramifications like “build your own signature”. These websites catch the image of the signatures and can use them in different places.

Therefore, proper verification of digital signatures is required.

Literature Survey

Cloud Key Management Service supports elliptic curve (EC) and RSA algorithms for digital signing. Both of these industry-standard algorithms offer choices of key size and digest algorithms.
Elliptic curve cryptography relies on one-way hash functions and point multiplication to compute points on an elliptic curve, combined with the intractability of determining the multiplicand for a point given its origin. This difficulty in determining the multiplicand is the cryptographic benefit of EC cryptography. The larger the size of the curve, the more difficult it is to compute the multiplicand. A benefit of EC cryptography is an EC key has a smaller key size compared to an RSA key that offers the same cryptographic strength.
RSA cryptography relies on the difficulty in factoring a large integer into two or more factors. The larger the key size, the more difficult it is to factor the integers.[1]

Many solutions for verifying the authenticity and the digital signatures have been developed. One of them is a designated confirmer signature scheme (RSA-DCSV). RSA encryption algorithm in form of extended modular computations thus achieves verifications on the validity of DCS as well as DCS recovery for a designated confirmer. By applying commitments to data consistency, proofs provided by a confirmer can be verified as expected expressions. Furthermore, those proofs are

characterized by the property that only a designated verifier can perform verifications in confirmation/denial protocol.
A verifier (V) has a designated confirmer signature σ , which is to be confirmed that it does actually consist the signer(S)'s regular signature (s) on an appointed document (m). The verifier puts forward a request Vreq to a designated confirmer, and the confirmer is required to provide proofs as well as its conclusion (confirmation or disapproval) on σ for the verifier.[2]

Another approach focuses on the improvement in the speed of the RSA algorithm. A scheme for the quick verification of the RSA signature has been proposed. This scheme pre-computes the quotients required to perform modular reduction during the verification and includes them as part of the signature that is being authenticated. This allows for replacing expensive modular reduction steps with less costly integer multiplications and subtractions. As a result, the scheme achieves significant speedup as well as the reduction in the code size and complexity. The quotients are computed using only the public key and the signature, and do not need to involve the signing procedure, nor require any changes in it. The verification scheme that uses these quotients as part of a quick verification flow is no weaker than the cryptographic primitive on which the signature scheme is based.[3]

A group signature protocol has been proposed. a group signature protocol with sub-signature verification using (t, k) threshold

RSA signature algorithm without trusted center for ad hoc network environment. The sub-signature verification protocol improves the efficiency of signature compute and provides the method to detect forged sub-signature.[4]

Proposed Methodologies

This report proposes a methodology to verify digital signatures that ensures Integrity, Authenticity and Non-repudiation.

For the simulation purpose, a Digital signature generator has been developed, which uses the JDK Security API provided by Java. For digital signatures to work, we need a public and private key pair. We invoke the getInstance() method of the KeyPairGenerator class. We instantiate this and use securerandom.strongAlgorithms to use the stong algorithms that Java provides. For simulation purposes, we use the SHA-1 algorithm.

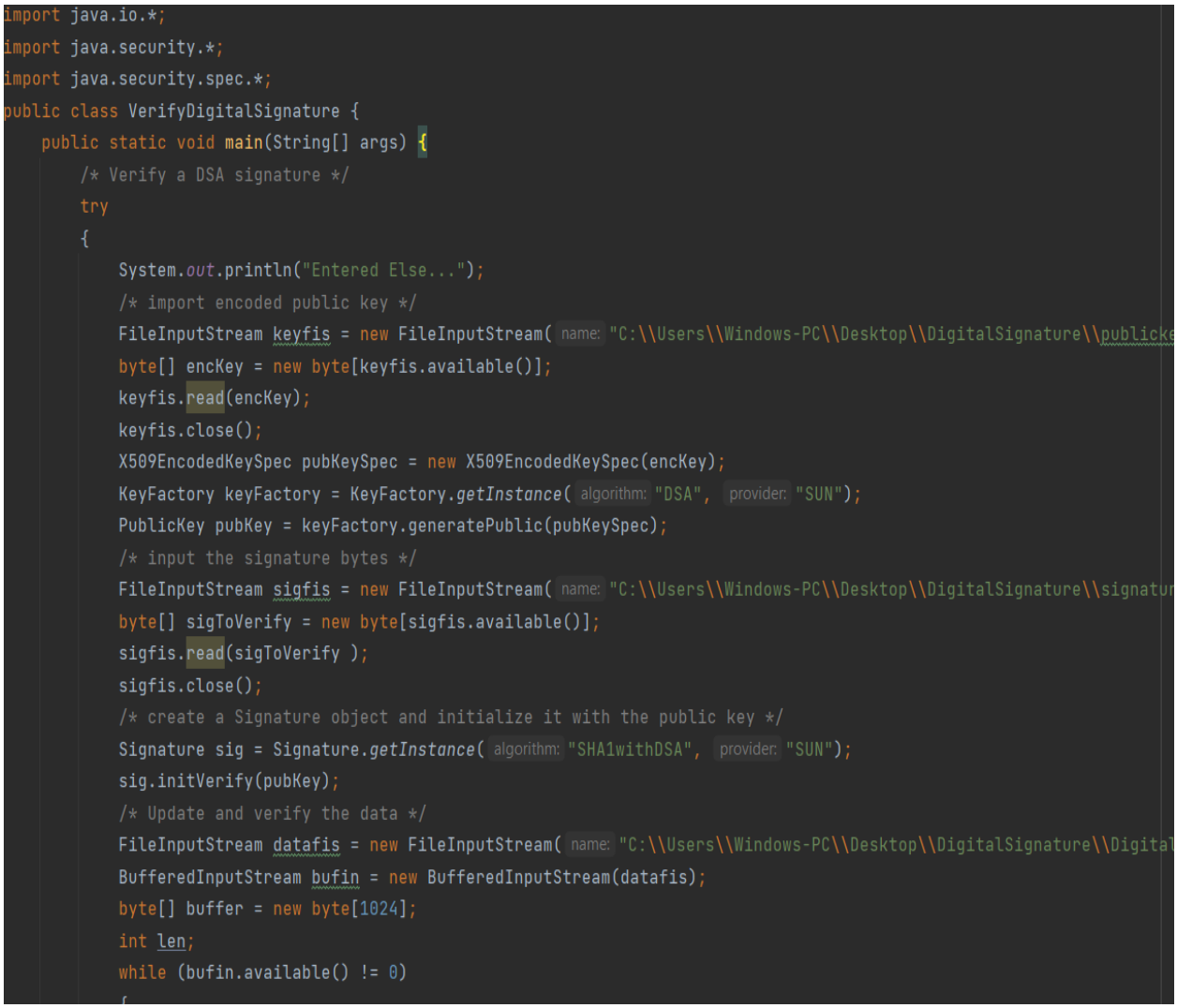
Here, the public and private keys required for the digital signature are created.

```
import java.io.*; //input the file data to be signed
import java.security.*; //provides methods for signing the data

public class GenerateDigitalSignature {
    public static void main(String[] args) {

        try {
            /* Generate a key pair */
            System.out.println("Entered else.....");
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance( algorithm: "DSA",  provider: "SUN");
            SecureRandom random = SecureRandom.getInstance( algorithm: "SHA1PRNG",  provider: "SUN");
            keyGen.initialize( keysize: 1024, random);
            KeyPair pair = keyGen.generateKeyPair();
            PrivateKey priv = pair.getPrivate();
            PublicKey pub = pair.getPublic();
            /* Create a Signature object and initialize it with the private key */
            Signature dsa = Signature.getInstance( algorithm: "SHA1withDSA",  provider: "SUN");
            dsa.initSign(priv);
            /* Update and sign the data */
            FileInputStream fis = new FileInputStream( name: "C:\\Users\\Windows-PC\\Desktop\\DigitalSi
            BufferedInputStream bufin = new BufferedInputStream(fis);
            byte[] buffer = new byte[1024];
            int len;
            while (bufin.available() != 0)
            {
                len = bufin.read(buffer);
                dsa.update(buffer,  off: 0, len);
            }
            bufin.close();
            /* Now that all the data to be signed has been read in,
            generate a signature for it */
            byte[] realSig = dsa.sign();
```

Code for generating keys

For the verification purpose, a package named java.security.spec that provides the X509EncodedKeySpec class has been imported. KeyFactory class has been used. It provides conversion between opaque keys (key type) and key specifications.

Opaque key uses the X509 standard that is used to determine the algorithm name, a format name, and the encoded key bytes.



```
import java.io.*;
import java.security.*;
import java.security.spec.*;
public class VerifyDigitalSignature {
    public static void main(String[] args) {
        /* Verify a DSA signature */
        try
        {
            System.out.println("Entered Else...");
            /* import encoded public key */
            FileInputStream keyfis = new FileInputStream( name: "C:\\Users\\Windows-PC\\Desktop\\DigitalSignature\\publicke
            byte[] encKey = new byte[keyfis.available()];
            keyfis.read(encKey);
            keyfis.close();
            X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);
            KeyFactory keyFactory = KeyFactory.getInstance( algorithm: "DSA",  provider: "SUN");
            PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);
            /* input the signature bytes */
            FileInputStream sigfis = new FileInputStream( name: "C:\\Users\\Windows-PC\\Desktop\\DigitalSignature\\signatur
            byte[] sigToVerify = new byte[sigfis.available()];
            sigfis.read(sigToVerify );
            sigfis.close();
            /* create a Signature object and initialize it with the public key */
            Signature sig = Signature.getInstance( algorithm: "SHA1withDSA",  provider: "SUN");
            sig.initVerify(pubKey);
            /* Update and verify the data */
            FileInputStream datafis = new FileInputStream( name: "C:\\Users\\Windows-PC\\Desktop\\DigitalSignature\\Digital
            BufferedInputStream bufin = new BufferedInputStream(datafis);
            byte[] buffer = new byte[1024];
            int len;
            while (bufin.available() != 0)
            {
```

Code fo Verification of Digital signatures

Conclusion and Future Work

In this paper, I have stated the problem of Improper Verification of Digital Signatures. There are many problems caused by this issue. A literature review of some papers has been presented and the solution proposed has been discussed. A Java code for the simulation of working of verification of the digital signature has been made. It consists of two sections, one for generating the public key and the private key, and the other for the verification of the digital signature. Future work includes enhancing the performance of the algorithms used for verification purpose.

References

[1]https://www.networkcomputing.com/networking/using-wireshark-identify-application-signatures

[2] A Protocol for Designated Confirmer Signatures Based on RSA Cryptographic Algorithms, Institute of Computer Science and Communications, Hunan University, Changsha, by Li Ping, Lin Yaping

[3] Quick Verification of RSA Signatures, Department of Mathematics, University of Haifa University of Haifa, by Haifa, Israel

[4] An Sub-signature Verification Protocol Using (t, k) Threshold RSA Signature Without Trusted Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, by Yanjiong Wang, Qiaoyan Wen, Hua Zhang

[5] http://www.emptrust.com/blog/benefits-of-using-digital-signatures/

[6] https://en.wikipedia.org/wiki/Digital_signature