Term Project Final Report

Title of the project

Light-Weight and Privacy-Preserving Authentication Protocol for Mobile Payments in the Context of IoT.

Group Code: G7 (Group No. 7)

Group Members: Names and Roll Numbers

Name	Roll Number	Email
Ashutosh Chauhan	S20180010017	ashutosh.c18@iiits.in
Saumya Doogar	S20180010156	saumya.d18@iiits.in

Abstract

In the current world IoT is getting widespread over a wide range of scenarios. From Smart Home to smart cards etc, IoT is becoming the new norm. One of the major characteristics IoT is being lightweight in terms of size, power consumption, performance, storage, etc. The protocol mentioned here provides security while still making sure that the performance and storage are within the capabilities of a IoT system. The protocol provides privacy and authentication for mobile payments in context of IoT. There are a various number of use cases from payments from smart devices like smart electric meters, smart cars, smart watches, monitoring systems, etc. The protocol uses at unidirectional certificateless proxy re-signature scheme, which is of independent interest. Based on this signature scheme, the protocol achieves anonymity, unforgeability and low performance overhead. In this protocol the computational overhead is placed on the Pay Platform. To increase the efficiency of the protocol, a batch-verification mechanism is provided for the Pay Platform and Merchant Server. The security of the protocol is based on the CDH (Computational Diffie-Hellman) Problem.

Protocol Summary

When a payment protocol in any IOT enabled smart device is implemented, the involved computation and storage space should be low for the limited resourced devices. However, in traditional transaction protocols, a public key infrastructure is introduced to issue certificates for public key of the user. This validity of the public key can be verified based on the certificates issued by a certificate authority. It is easy to see that PKI caused a lot of communication and storage costs when the revocation, storage, and distribution of certificates are done.

To solve the above challenge, we propose a new mobile payment scheme that achieves anonymity, unforgeability and low resource consumption simultaneously.
All In all, this protocol accomplishes 3 things :

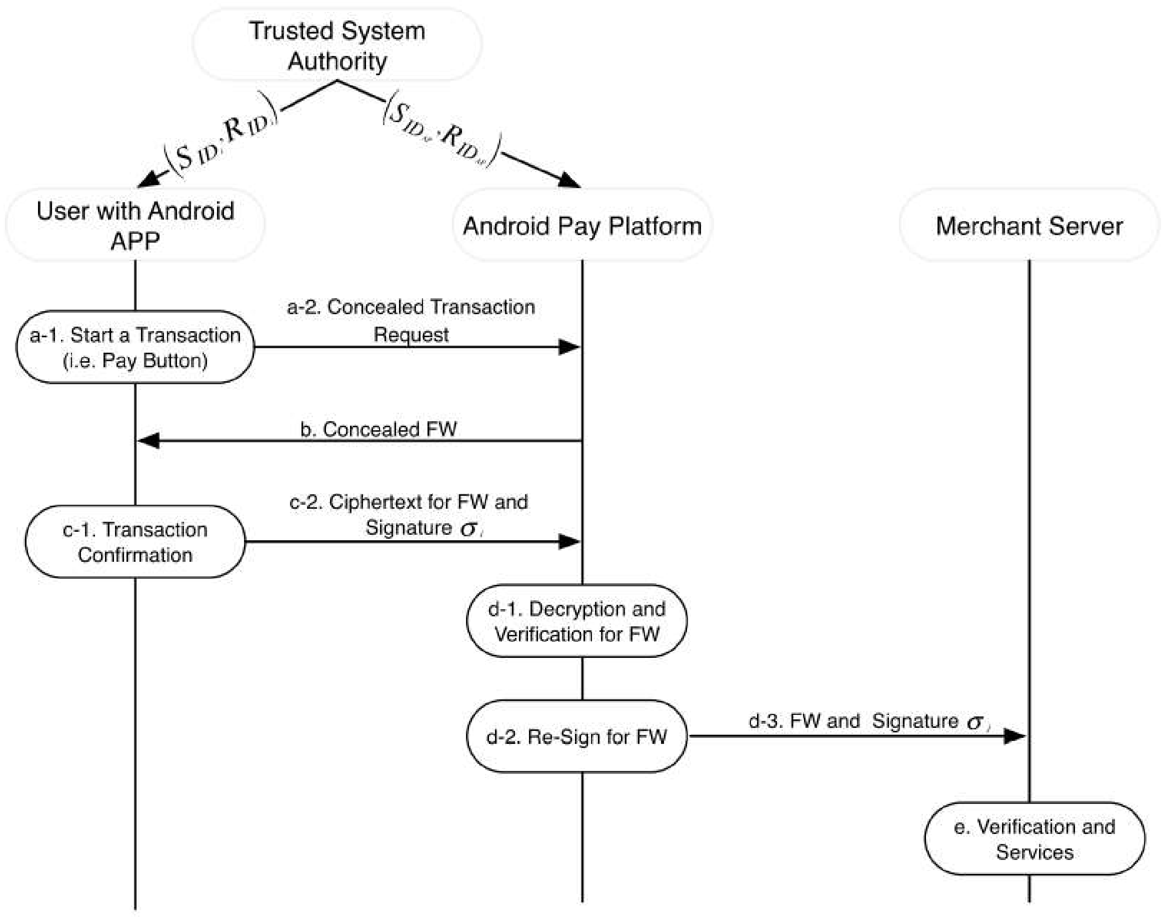
1. We propose the first unidirectional certificate-less proxy re-signature scheme which is of independent interest.
2. A mobile payment protocol with user anonymity is presented based on our proposed scheme.
3. Using Batch-verification to accelerate the signature verification process such that multiple signatures from different users on distinct messages can be verified quickly. Moreover, the signatures from the same user can be further batched to achieve higher efficiency

Plan of Implementation

SYSTEM MODEL OF OUR TRANSACTION PROTOCOL

The considered system consists of four types of entities: the trusted system authority (TSA), the user app, the merchant server, and the Pay Platform [9].

1. Trusted System Authority: TSA is a trusted third party organization that provides registration services for User’s App and Pay Platform. TSA also distributes system params and partial private keys for registered users to ensure the whole scheme successfully works.
2. User’s App: Any software that requires a payment function is called User’s App, such as Apple pay, etc. This application needs to be registered with the TSA to obtain the corresponding system params and partial private key. It also generates its own user secret value and public key. Then User’s App completes the signature using its full private key, which consists of partial private key.
3. Pay Platform: Pay Platform is an application offered by a trusted party, of course, it also needs to register with the TSA to obtain system params and private key. Simultaneously, in order to protect the user’s information of the transaction, Pay Platform will provide re-sign service, that is, the Pay Platform transforms signature of User’s App into signature of Pay Platform.
4. Merchant Server: Merchant Server is utilized by a merchant, it verifies the correctness of the transaction information to check the product is given to the right user.

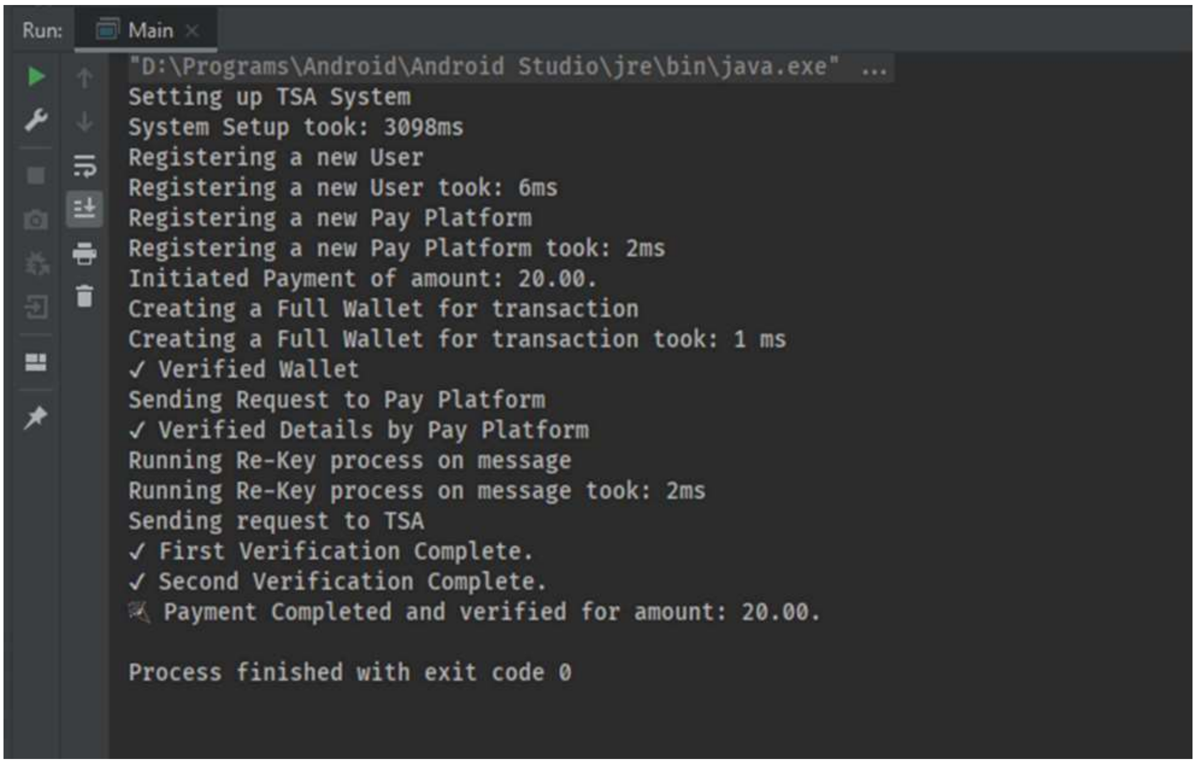

Trusted System Authority
(S_{ID_i}, R_{ID_i})
(S_{ID_w}, R_{ID_w})
User with Android APP
Android Pay Platform
Merchant Server
a-1. Start a Transaction (i.e. Pay Button)
a-2. Concealed Transaction Request
b. Concealed FW
c-1. Transaction Confirmation
c-2. Ciphertext for FW and Signature σ_i
d-1. Decryption and Verification for FW
d-2. Re-Sign for FW
d-3. FW and Signature σ_j
e. Verification and Services

Code and Experimental Results

We implemented the protocol in Java using the JPBC Library. We followed an OOP approach with interfaces to keep the code modular and mocked Message Passing between the various entities. We Initialized the Secret Parameters and the ECC Pairing generation followed by the process of User Registration and Pay Platform Registration. And finally making a payment.

We used Type A elliptical curve with G1 and G2 size 512 and 1024 bits respectively.

Single Run Results:



```
"D:\Programs\Android\Android Studio\jre\bin\java.exe" ...
Setting up TSA System
System Setup took: 3098ms
Registering a new User
Registering a new User took: 6ms
Registering a new Pay Platform
Registering a new Pay Platform took: 2ms
Initiated Payment of amount: 20.00.
Creating a Full Wallet for transaction
Creating a Full Wallet for transaction took: 1 ms
✓ Verified Wallet
Sending Request to Pay Platform
✓ Verified Details by Pay Platform
Running Re-Key process on message
Running Re-Key process on message took: 2ms
Sending request to TSA
✓ First Verification Complete.
✓ Second Verification Complete.
🎉 Payment Completed and verified for amount: 20.00.

Process finished with exit code 0
```

Results (Average Time (100)):

- System Setup: 4.1 seconds
- Registration: 10 ms
- Transaction: 1~2 ms

Observations and Conclusion

Performance Analysis:

Our local machine on which we run code was more powerful than the one mentioned in Paper but to due to lack of available hardware we ran the code and found really good performance.

Scope of Improvement:

We used JBPC which relies on Java VM runtime and can cause an overhead, using a native machine code implementation such as (PBC) for C++ will provide more improvement in performance.

Configuration for User's App (Original)

- CPU: PXA270 processor 624MHz
- RAM: 1GB memory

Configuration for Payment Platform (Original)

- CPU: Intel i3-380M processor 2.53GHz
- RAM: 8GB memory

Hash Function: SHA-

G_1 , Z_q → 64 Byte

G_2 → 128 Byte

ECC → $y^2 = x^3 + x$

Paper's Usage

- VC++ 6.
- PBC library

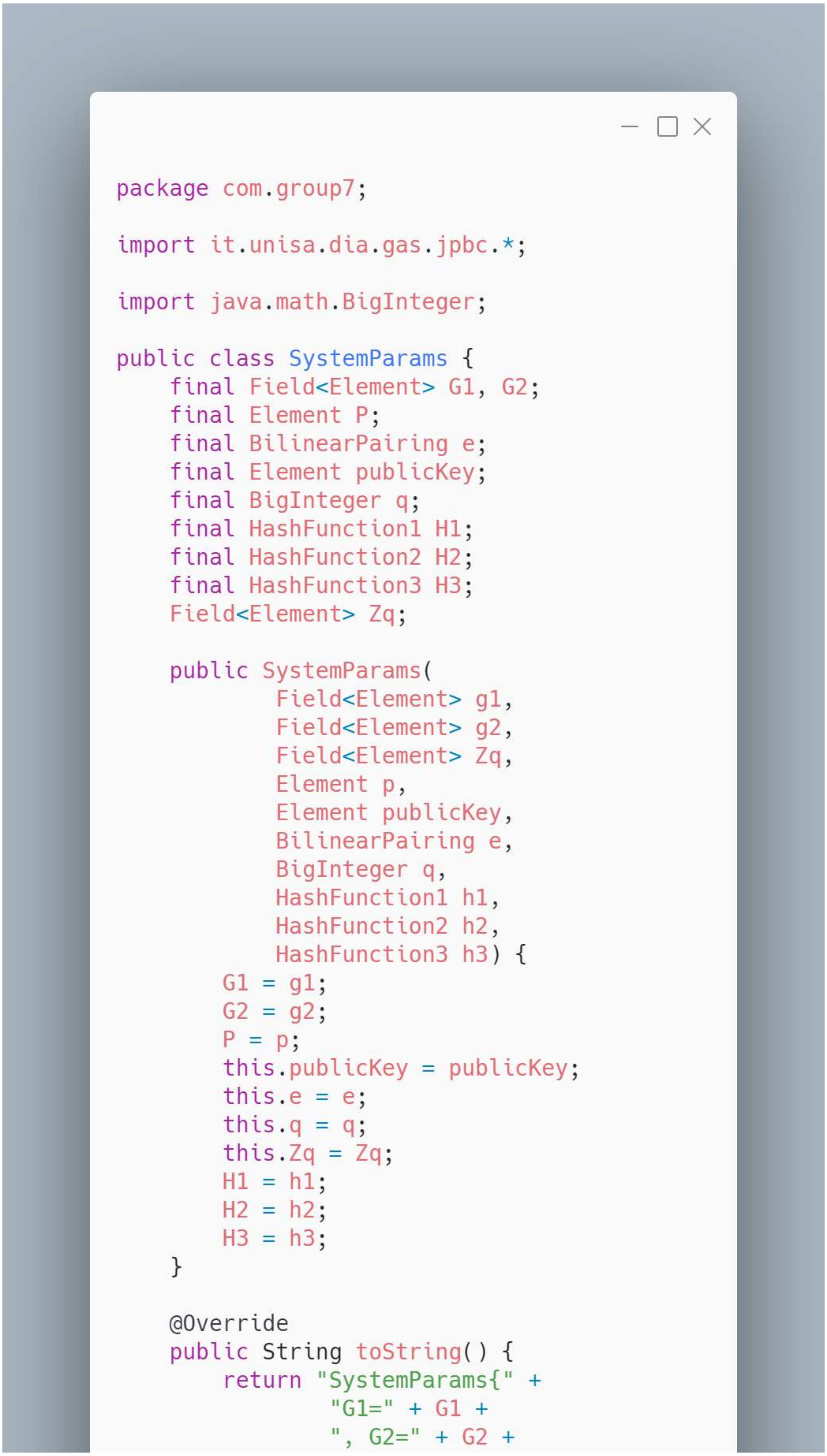
Our Usage

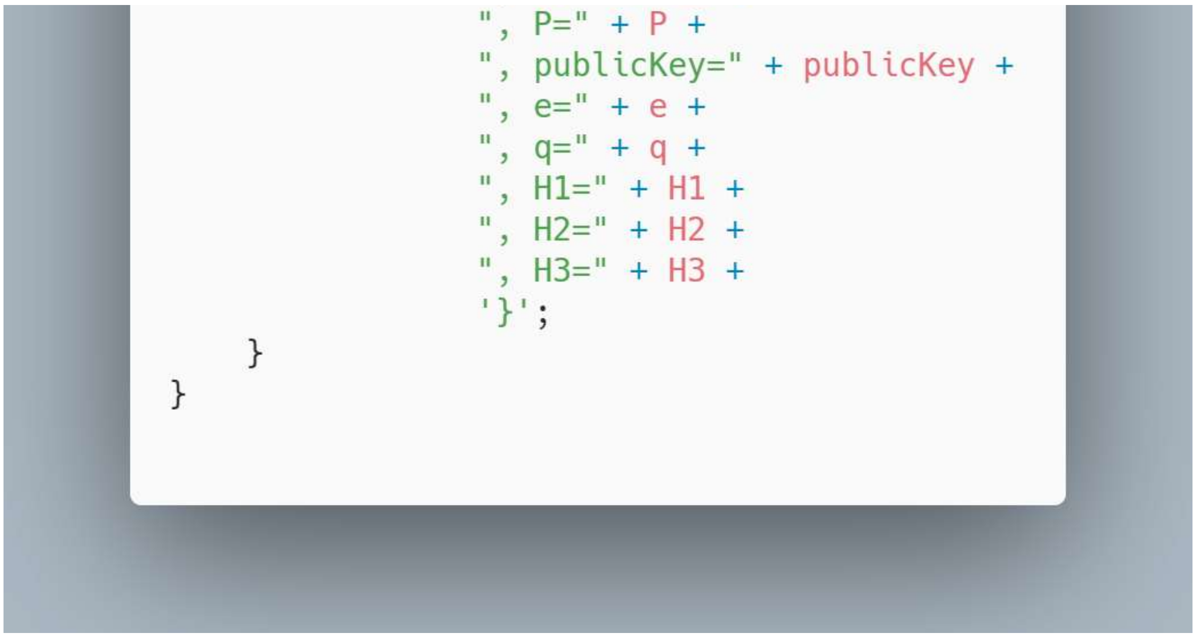
- GNU G++
- PBC library

Summary of the results

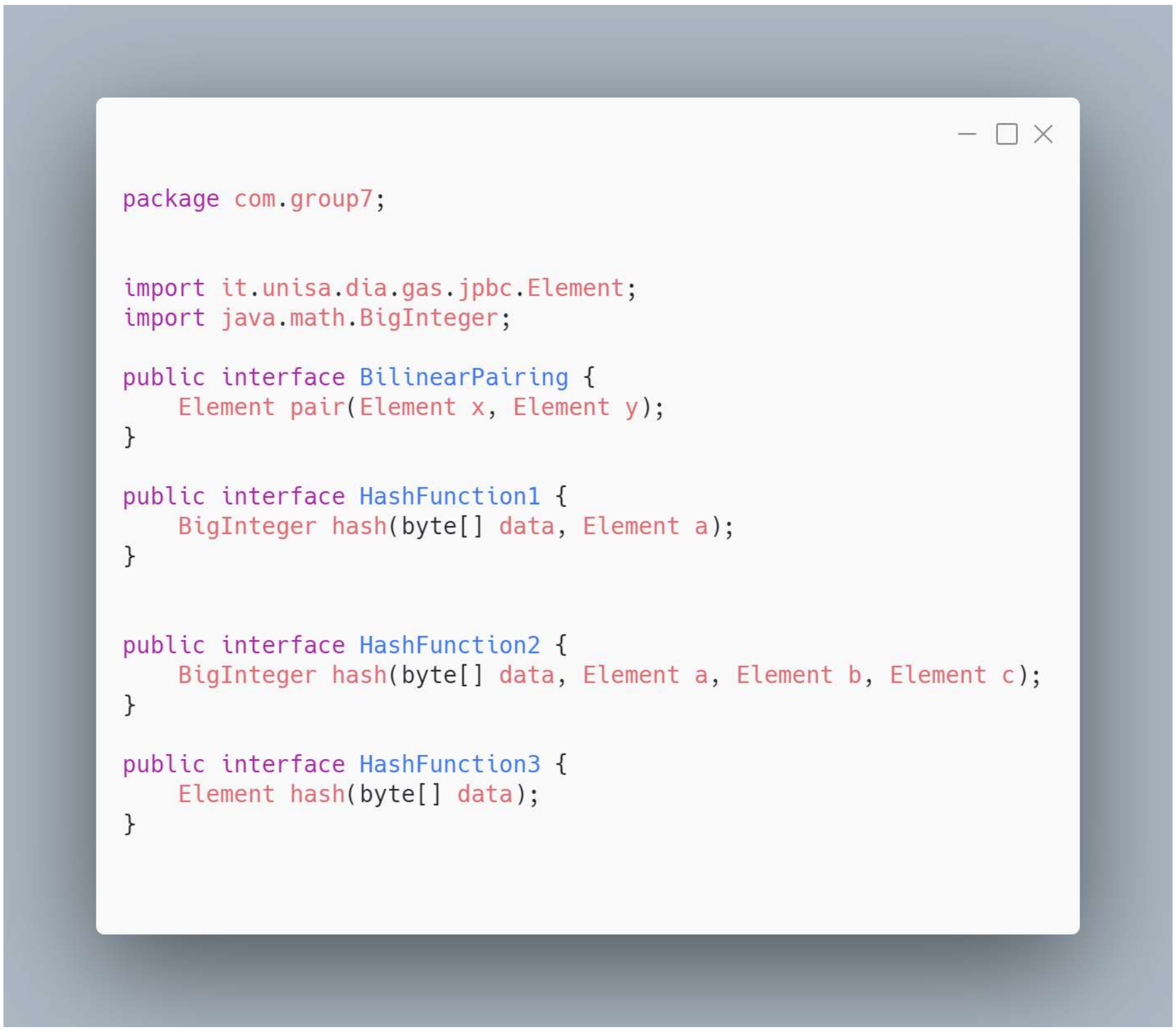
The Protocol is really secure and also works really fast and is also very low on computation and storage resources and hence is a good pick for IOT mobile payments. In our implementation we found that the User side computation took very few time hence improving the user experience with IOT based payments as well providing the same level of security as 2048 bit RSA.

Source Code

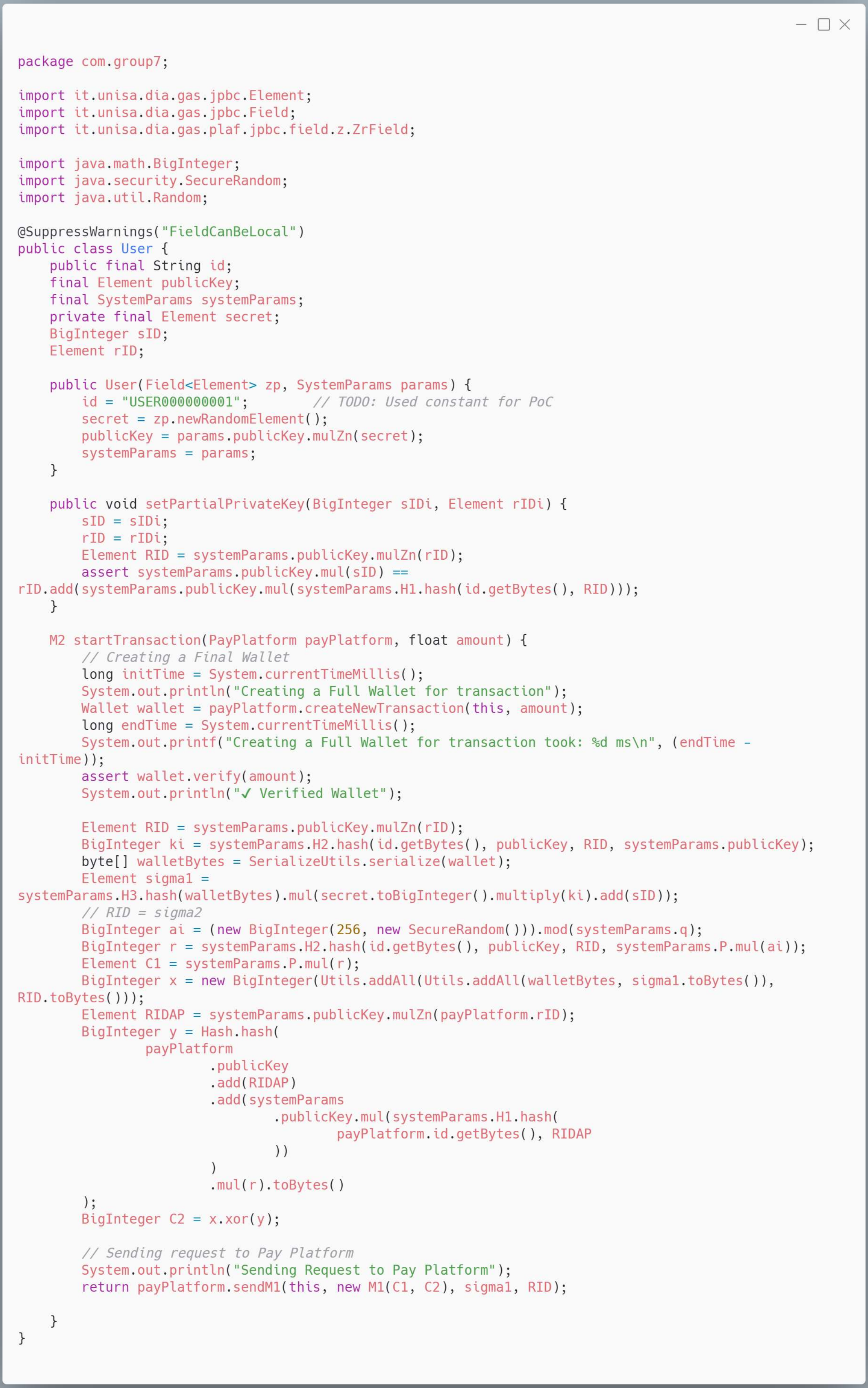

```java
package com.group7;

import it.unisa.dia.gas.jpbc.*;

import java.math.BigInteger;

public class SystemParams {
    final Field<Element> G1, G2;
    final Element P;
    final BilinearPairing e;
    final Element publicKey;
    final BigInteger q;
    final HashFunction1 H1;
    final HashFunction2 H2;
    final HashFunction3 H3;
    Field<Element> Zq;

    public SystemParams(
            Field<Element> g1,
            Field<Element> g2,
            Field<Element> Zq,
            Element p,
            Element publicKey,
            BilinearPairing e,
            BigInteger q,
            HashFunction1 h1,
            HashFunction2 h2,
            HashFunction3 h3) {
        G1 = g1;
        G2 = g2;
        P = p;
        this.publicKey = publicKey;
        this.e = e;
        this.q = q;
        this.Zq = Zq;
        H1 = h1;
        H2 = h2;
        H3 = h3;
    }

    @Override
    public String toString() {
        return "SystemParams{" +
                "G1=" + G1 +
                ", G2=" + G2 +
```



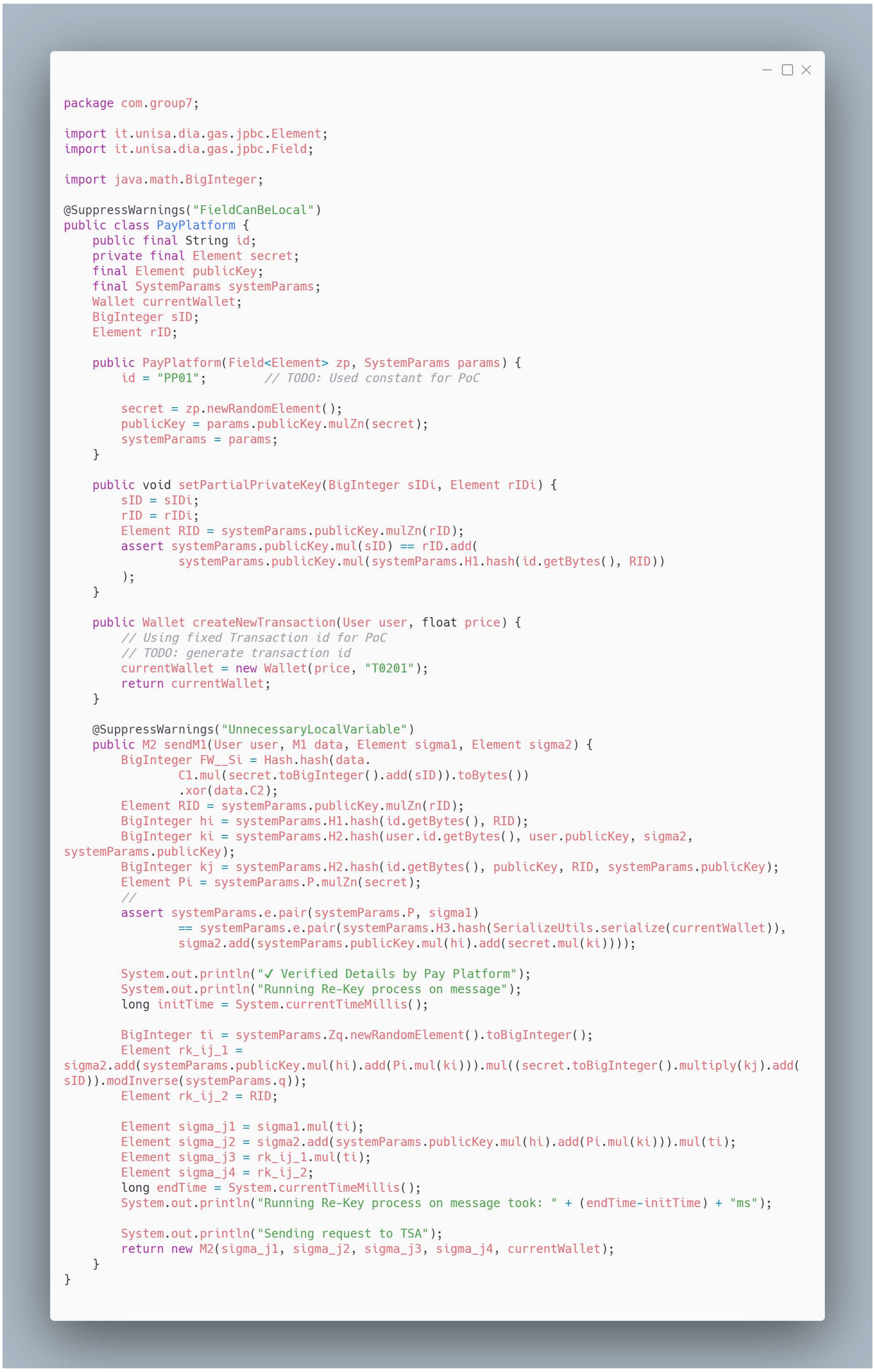
```
                ", P=" + P +
                ", publicKey=" + publicKey +
                ", e=" + e +
                ", q=" + q +
                ", H1=" + H1 +
                ", H2=" + H2 +
                ", H3=" + H3 +
                '}';
    }
}
```



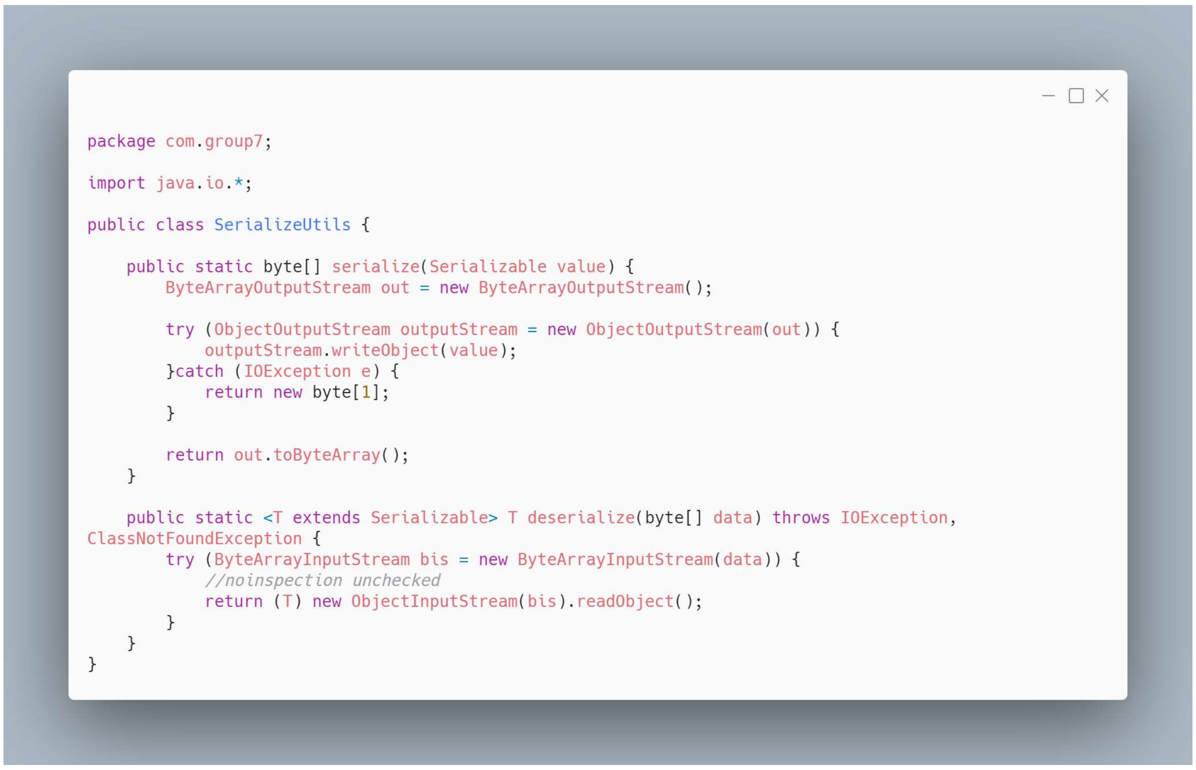
```
package com.group7;


import it.unisa.dia.gas.jpbc.Element;
import java.math.BigInteger;

public interface BilinearPairing {
    Element pair(Element x, Element y);
}

public interface HashFunction1 {
    BigInteger hash(byte[] data, Element a);
}


public interface HashFunction2 {
    BigInteger hash(byte[] data, Element a, Element b, Element c);
}

public interface HashFunction3 {
    Element hash(byte[] data);
}
```



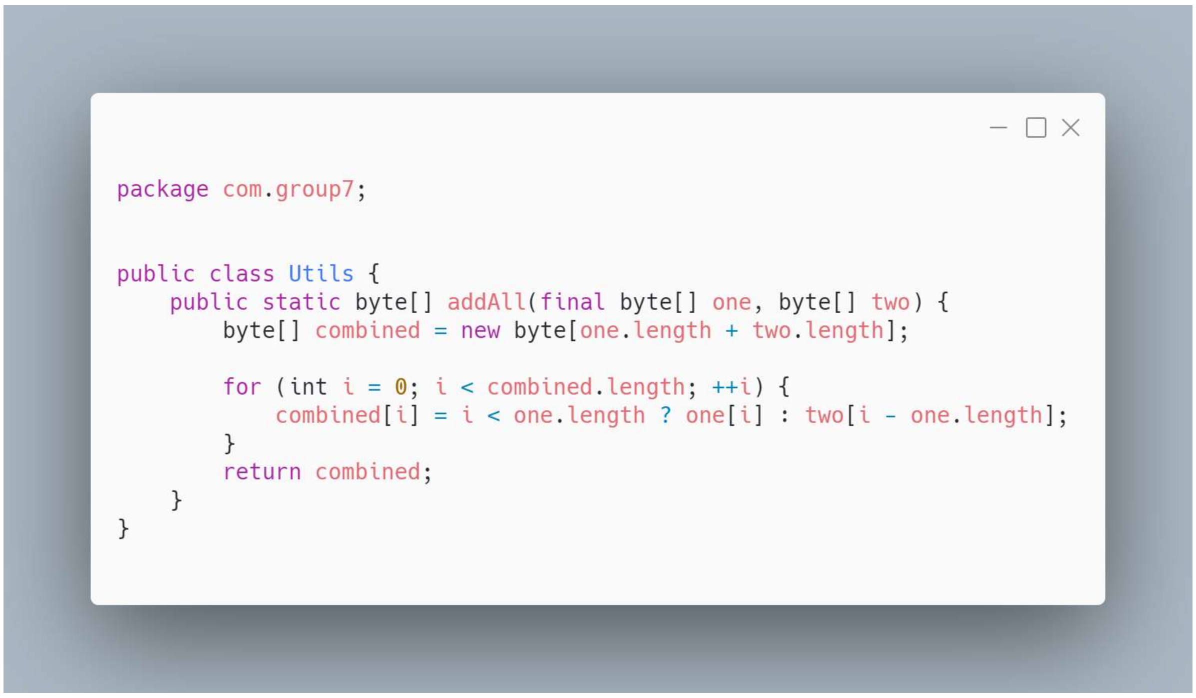
```java
package com.group7;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.plaf.jpbc.field.z.ZrField;

import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

@SuppressWarnings("FieldCanBeLocal")
public class User {
    public final String id;
    final Element publicKey;
    final SystemParams systemParams;
    private final Element secret;
    BigInteger sID;
    Element rID;

    public User(Field<Element> zp, SystemParams params) {
        id = "USER000000001";        // TODO: Used constant for PoC
        secret = zp.newRandomElement();
        publicKey = params.publicKey.mulZn(secret);
        systemParams = params;
    }

    public void setPartialPrivateKey(BigInteger sIDi, Element rIDi) {
        sID = sIDi;
        rID = rIDi;
        Element RID = systemParams.publicKey.mulZn(rID);
        assert systemParams.publicKey.mul(sID) ==
rID.add(systemParams.publicKey.mul(systemParams.H1.hash(id.getBytes(), RID)));
    }

    M2 startTransaction(PayPlatform payPlatform, float amount) {
        // Creating a Final Wallet
        long initTime = System.currentTimeMillis();
        System.out.println("Creating a Full Wallet for transaction");
        Wallet wallet = payPlatform.createNewTransaction(this, amount);
        long endTime = System.currentTimeMillis();
        System.out.printf("Creating a Full Wallet for transaction took: %d ms\n", (endTime -
initTime));
        assert wallet.verify(amount);
        System.out.println("✓ Verified Wallet");

        Element RID = systemParams.publicKey.mulZn(rID);
        BigInteger ki = systemParams.H2.hash(id.getBytes(), publicKey, RID, systemParams.publicKey);
        byte[] walletBytes = SerializeUtils.serialize(wallet);
        Element sigma1 =
systemParams.H3.hash(walletBytes).mul(secret.toBigInteger().multiply(ki).add(sID));
        // RID = sigma2
        BigInteger ai = (new BigInteger(256, new SecureRandom())).mod(systemParams.q);
        BigInteger r = systemParams.H2.hash(id.getBytes(), publicKey, RID, systemParams.P.mul(ai));
        Element C1 = systemParams.P.mul(r);
        BigInteger x = new BigInteger(Utils.addAll(Utils.addAll(walletBytes, sigma1.toBytes()),
RID.toBytes()));
        Element RIDAP = systemParams.publicKey.mulZn(payPlatform.rID);
        BigInteger y = Hash.hash(
                payPlatform
                        .publicKey
                        .add(RIDAP)
                        .add(systemParams
                                .publicKey.mul(systemParams.H1.hash(
                                        payPlatform.id.getBytes(), RIDAP
                                ))
                        )
                        .mul(r).toBytes()
        );
        BigInteger C2 = x.xor(y);

        // Sending request to Pay Platform
        System.out.println("Sending Request to Pay Platform");
        return payPlatform.sendM1(this, new M1(C1, C2), sigma1, RID);

    }
}
```



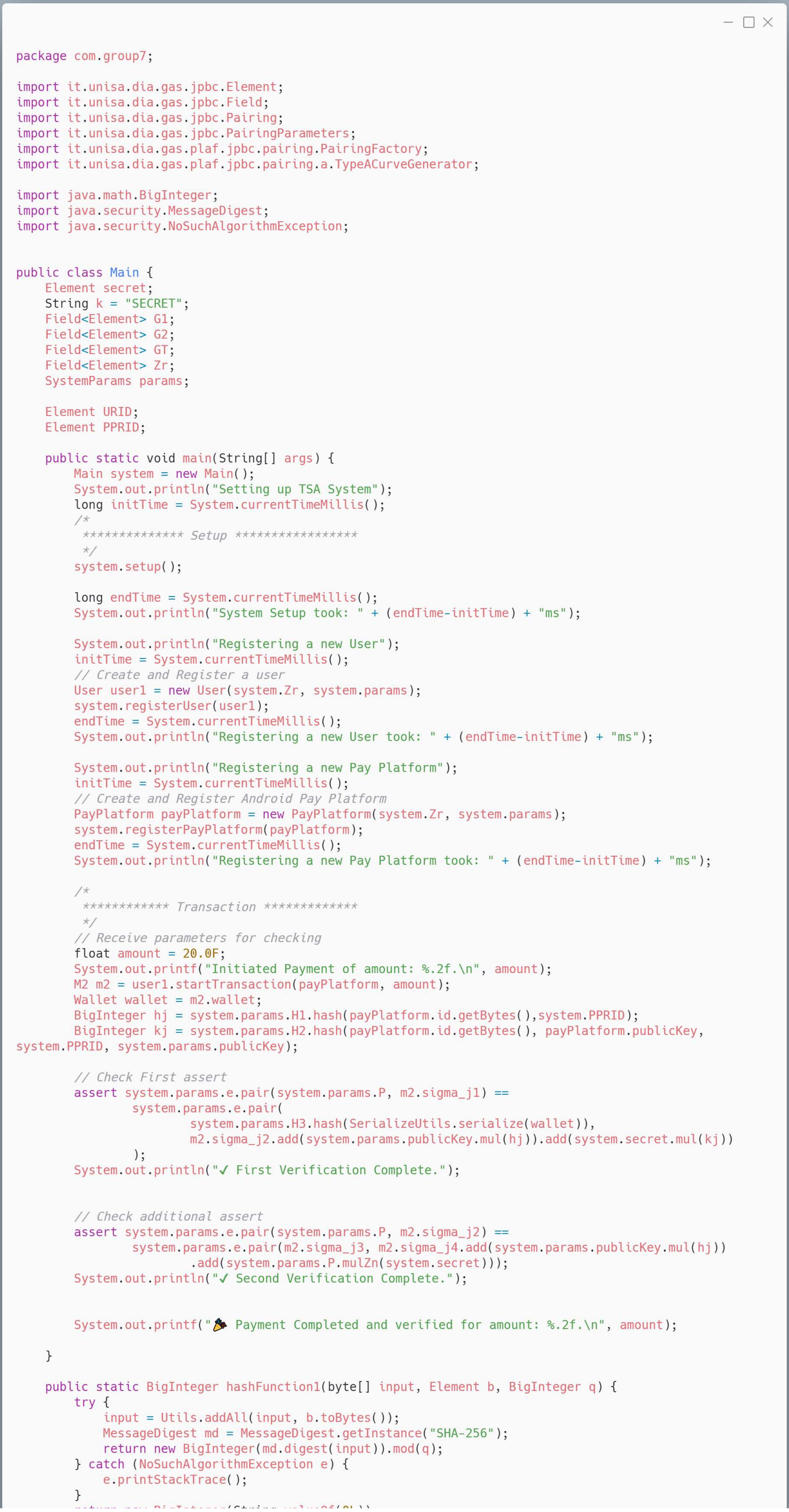
```java
package com.group7;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;

import java.math.BigInteger;

@SuppressWarnings("FieldCanBeLocal")
public class PayPlatform {
    public final String id;
    private final Element secret;
    final Element publicKey;
    final SystemParams systemParams;
    Wallet currentWallet;
    BigInteger sID;
    Element rID;

    public PayPlatform(Field<Element> zp, SystemParams params) {
        id = "PP01";        // TODO: Used constant for PoC

        secret = zp.newRandomElement();
        publicKey = params.publicKey.mulZn(secret);
        systemParams = params;
    }

    public void setPartialPrivateKey(BigInteger sIDi, Element rIDi) {
        sID = sIDi;
        rID = rIDi;
        Element RID = systemParams.publicKey.mulZn(rID);
        assert systemParams.publicKey.mul(sID) == rID.add(
                systemParams.publicKey.mul(systemParams.H1.hash(id.getBytes(), RID))
        );
    }

    public Wallet createNewTransaction(User user, float price) {
        // Using fixed Transaction id for PoC
        // TODO: generate transaction id
        currentWallet = new Wallet(price, "T0201");
        return currentWallet;
    }

    @SuppressWarnings("UnnecessaryLocalVariable")
    public M2 sendM1(User user, M1 data, Element sigma1, Element sigma2) {
        BigInteger FW__Si = Hash.hash(data.
                C1.mul(secret.toBigInteger().add(sID)).toBytes())
                .xor(data.C2);
        Element RID = systemParams.publicKey.mulZn(rID);
        BigInteger hi = systemParams.H1.hash(id.getBytes(), RID);
        BigInteger ki = systemParams.H2.hash(user.id.getBytes(), user.publicKey, sigma2,
systemParams.publicKey);
        BigInteger kj = systemParams.H2.hash(id.getBytes(), publicKey, RID, systemParams.publicKey);
        Element Pi = systemParams.P.mulZn(secret);
        //
        assert systemParams.e.pair(systemParams.P, sigma1)
                == systemParams.e.pair(systemParams.H3.hash(SerializeUtils.serialize(currentWallet)),
                sigma2.add(systemParams.publicKey.mul(hi).add(secret.mul(ki))));

        System.out.println("✓ Verified Details by Pay Platform");
        System.out.println("Running Re-Key process on message");
        long initTime = System.currentTimeMillis();

        BigInteger ti = systemParams.Zq.newRandomElement().toBigInteger();
        Element rk_ij_1 =
sigma2.add(systemParams.publicKey.mul(hi).add(Pi.mul(ki))).mul((secret.toBigInteger().multiply(kj).add(
sID)).modInverse(systemParams.q));
        Element rk_ij_2 = RID;

        Element sigma_j1 = sigma1.mul(ti);
        Element sigma_j2 = sigma2.add(systemParams.publicKey.mul(hi).add(Pi.mul(ki))).mul(ti);
        Element sigma_j3 = rk_ij_1.mul(ti);
        Element sigma_j4 = rk_ij_2;
        long endTime = System.currentTimeMillis();
        System.out.println("Running Re-Key process on message took: " + (endTime-initTime) + "ms");

        System.out.println("Sending request to TSA");
        return new M2(sigma_j1, sigma_j2, sigma_j3, sigma_j4, currentWallet);
    }
}
```



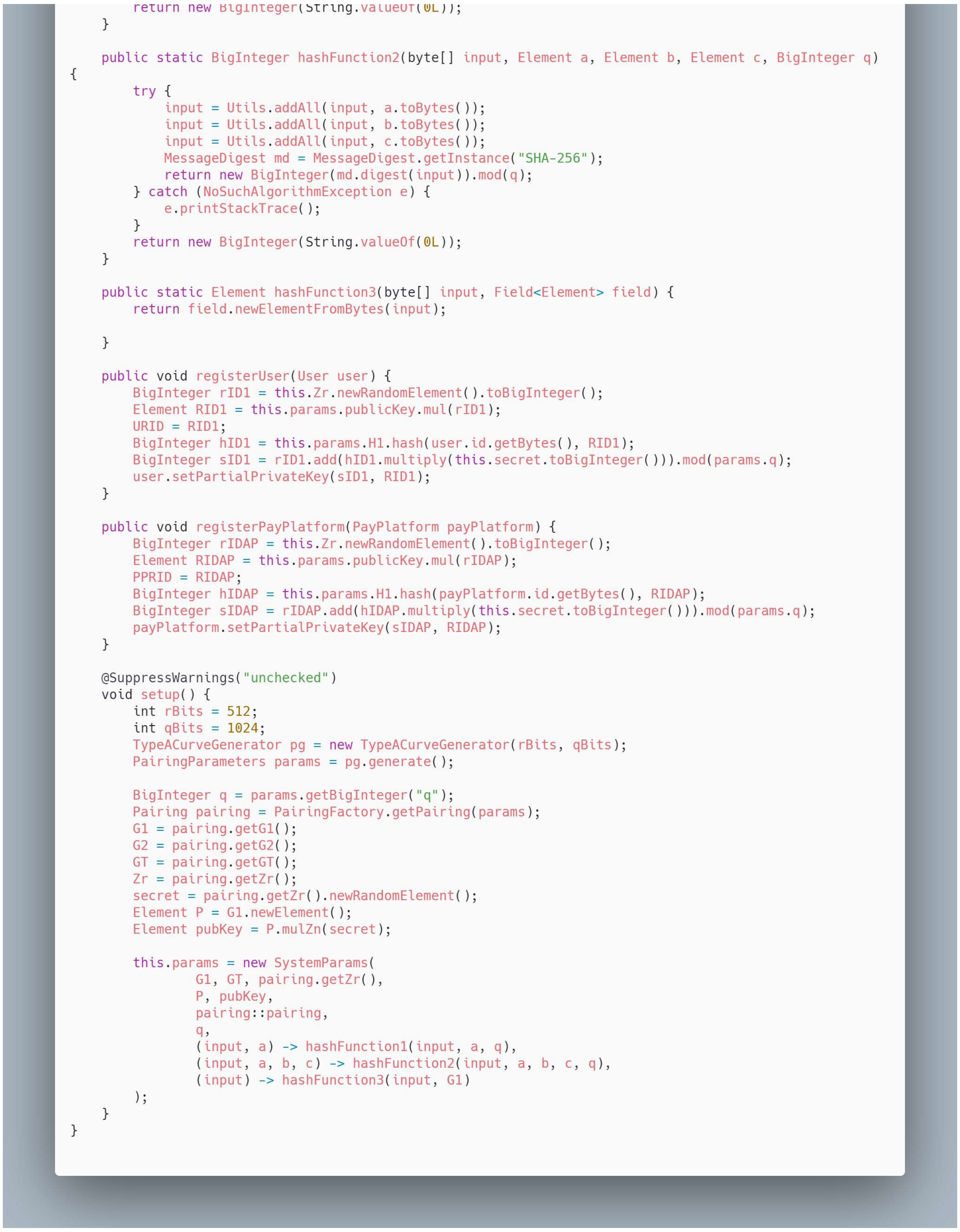
```java
package com.group7;

import java.io.*;

public class SerializeUtils {

    public static byte[] serialize(Serializable value) {
        ByteArrayOutputStream out = new ByteArrayOutputStream();

        try (ObjectOutputStream outputStream = new ObjectOutputStream(out)) {
            outputStream.writeObject(value);
        }catch (IOException e) {
            return new byte[1];
        }

        return out.toByteArray();
    }

    public static <T extends Serializable> T deserialize(byte[] data) throws IOException,
ClassNotFoundException {
        try (ByteArrayInputStream bis = new ByteArrayInputStream(data)) {
            //noinspection unchecked
            return (T) new ObjectInputStream(bis).readObject();
        }
    }
}
```



```java
package com.group7;


public class Utils {
    public static byte[] addAll(final byte[] one, byte[] two) {
        byte[] combined = new byte[one.length + two.length];

        for (int i = 0; i < combined.length; ++i) {
            combined[i] = i < one.length ? one[i] : two[i - one.length];
        }
        return combined;
    }
}
```



```java
package com.group7;

import it.unisa.dia.gas.jpbc.Element;
import it.unisa.dia.gas.jpbc.Field;
import it.unisa.dia.gas.jpbc.Pairing;
import it.unisa.dia.gas.jpbc.PairingParameters;
import it.unisa.dia.gas.plaf.jpbc.pairing.PairingFactory;
import it.unisa.dia.gas.plaf.jpbc.pairing.a.TypeACurveGenerator;

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;


public class Main {
    Element secret;
    String k = "SECRET";
    Field<Element> G1;
    Field<Element> G2;
    Field<Element> GT;
    Field<Element> Zr;
    SystemParams params;

    Element URID;
    Element PPRID;

    public static void main(String[] args) {
        Main system = new Main();
        System.out.println("Setting up TSA System");
        long initTime = System.currentTimeMillis();
        /*
         ************** Setup *****************
         */
        system.setup();

        long endTime = System.currentTimeMillis();
        System.out.println("System Setup took: " + (endTime-initTime) + "ms");

        System.out.println("Registering a new User");
        initTime = System.currentTimeMillis();
        // Create and Register a user
        User user1 = new User(system.Zr, system.params);
        system.registerUser(user1);
        endTime = System.currentTimeMillis();
        System.out.println("Registering a new User took: " + (endTime-initTime) + "ms");

        System.out.println("Registering a new Pay Platform");
        initTime = System.currentTimeMillis();
        // Create and Register Android Pay Platform
        PayPlatform payPlatform = new PayPlatform(system.Zr, system.params);
        system.registerPayPlatform(payPlatform);
        endTime = System.currentTimeMillis();
        System.out.println("Registering a new Pay Platform took: " + (endTime-initTime) + "ms");

        /*
         ************ Transaction *************
         */
        // Receive parameters for checking
        float amount = 20.0F;
        System.out.printf("Initiated Payment of amount: %.2f.\n", amount);
        M2 m2 = user1.startTransaction(payPlatform, amount);
        Wallet wallet = m2.wallet;
        BigInteger hj = system.params.H1.hash(payPlatform.id.getBytes(),system.PPRID);
        BigInteger kj = system.params.H2.hash(payPlatform.id.getBytes(), payPlatform.publicKey,
system.PPRID, system.params.publicKey);

        // Check First assert
        assert system.params.e.pair(system.params.P, m2.sigma_j1) ==
                system.params.e.pair(
                        system.params.H3.hash(SerializeUtils.serialize(wallet)),
                        m2.sigma_j2.add(system.params.publicKey.mul(hj)).add(system.secret.mul(kj))
                );
        System.out.println("✔ First Verification Complete.");


        // Check additional assert
        assert system.params.e.pair(system.params.P, m2.sigma_j2) ==
                system.params.e.pair(m2.sigma_j3, m2.sigma_j4.add(system.params.publicKey.mul(hj))
                        .add(system.params.P.mulZn(system.secret)));
        System.out.println("✔ Second Verification Complete.");


        System.out.printf("🎉 Payment Completed and verified for amount: %.2f.\n", amount);

    }

    public static BigInteger hashFunction1(byte[] input, Element b, BigInteger q) {
        try {
            input = Utils.addAll(input, b.toBytes());
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            return new BigInteger(md.digest(input)).mod(q);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return new BigInteger(String.valueOf(0L));
```



```java
        return new BigInteger(String.valueOf(0L));
    }

    public static BigInteger hashFunction2(byte[] input, Element a, Element b, Element c, BigInteger q)
{
        try {
            input = Utils.addAll(input, a.toBytes());
            input = Utils.addAll(input, b.toBytes());
            input = Utils.addAll(input, c.toBytes());
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            return new BigInteger(md.digest(input)).mod(q);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return new BigInteger(String.valueOf(0L));
    }

    public static Element hashFunction3(byte[] input, Field<Element> field) {
        return field.newElementFromBytes(input);

    }

    public void registerUser(User user) {
        BigInteger rID1 = this.Zr.newRandomElement().toBigInteger();
        Element RID1 = this.params.publicKey.mul(rID1);
        URID = RID1;
        BigInteger hID1 = this.params.H1.hash(user.id.getBytes(), RID1);
        BigInteger sID1 = rID1.add(hID1.multiply(this.secret.toBigInteger())).mod(params.q);
        user.setPartialPrivateKey(sID1, RID1);
    }

    public void registerPayPlatform(PayPlatform payPlatform) {
        BigInteger rIDAP = this.Zr.newRandomElement().toBigInteger();
        Element RIDAP = this.params.publicKey.mul(rIDAP);
        PPRID = RIDAP;
        BigInteger hIDAP = this.params.H1.hash(payPlatform.id.getBytes(), RIDAP);
        BigInteger sIDAP = rIDAP.add(hIDAP.multiply(this.secret.toBigInteger())).mod(params.q);
        payPlatform.setPartialPrivateKey(sIDAP, RIDAP);
    }

    @SuppressWarnings("unchecked")
    void setup() {
        int rBits = 512;
        int qBits = 1024;
        TypeACurveGenerator pg = new TypeACurveGenerator(rBits, qBits);
        PairingParameters params = pg.generate();

        BigInteger q = params.getBigInteger("q");
        Pairing pairing = PairingFactory.getPairing(params);
        G1 = pairing.getG1();
        G2 = pairing.getG2();
        GT = pairing.getGT();
        Zr = pairing.getZr();
        secret = pairing.getZr().newRandomElement();
        Element P = G1.newElement();
        Element pubKey = P.mulZn(secret);

        this.params = new SystemParams(
                G1, GT, pairing.getZr(),
                P, pubKey,
                pairing::pairing,
                q,
                (input, a) -> hashFunction1(input, a, q),
                (input, a, b, c) -> hashFunction2(input, a, b, c, q),
                (input) -> hashFunction3(input, G1)
        );
    }
}
```

References

[1]: V. Sureshkumar, A. Ramalingam, N. Rajamanickam, and R. Amin, “A lightweight two-gateway based payment protocol ensuring accountability and unlinkable anonymity with dynamic identity,” Comput. Elect. Eng., vol. 57, pp. 223–240, Jan. 2017.

[2]: J-H. Yang and P.-Y. Lin, “A mobile payment mechanism with anonymity for cloud computing,” J. Syst. Softw., vol. 116, pp. 69–74, Jun. 2016.

[3]: Z. Qin, J. Sun, A. Wahaballa, W. Zheng, H. Xiong, and H. Qin, “A secure and privacy-preserving mobile wallet with outsourced verification in cloud computing,” Comput. Standards Interfaces, vol. 54, pp. 55–60, Nov. 2017.

[4]: K.-H. Yeh, “A secure transaction scheme with certificateless cryptographic primitives for IoT-based mobile payments,” IEEE Syst. J., doi: 10.1109/JSYST.2017.

[5] Y. Liao, Y. He, F. Li, and S. Zhou, “Analysis of a mobile payment protocol with outsourced verification in cloud server and the improvement,” Comput. Standards Interfaces, vol. 56, pp. 101–106, Feb. 2018, doi: 10.1016/j.csi.2017.09.008.