**//Angela Fogel**

**//CS625 Cryptography**

**//Digital Cash Protocol #4**

**June 24, 2014**

**File: FogelCash**

**DIGITAL SIGNATURE WITH CRYPTO++ RSA.**

RSA Libraries of Crypto++ include **Signature Schemes** and **Encryption Schemes**. Signature Schemes include Signature Schemes with Recovery which include embedding a message for encryption with the signature. Using the public key the recipient decrypts the message and verifies the signature from the same object. Encryption and signature is performed with the Private Key. Verification and decryption with the Public Key

A signature scheme uses pipelining to pass the message source through various filters to a sink. The Signature Scheme with Recovery used in the Digital Cash example relies on **SignerFilter** interface and its corresponding **SignatureVerificationFilter**. The functions for the filters rely on FLAGS parameters to distinguish the message to be recovered (decrypted) from the signature to be verified and the where to forward the attached transformation. Crypto++ has a RSA Probabilistic Signature Scheme with Recover that does not use filters. It includes SecByteBlock to allocate a block of memory and MaxSignatureLength and SignMessageWithRecovery functions to determine start of message, sign and store the encrypted message.

The method uses a Hash function (SHA-1 a Secure Hash Algorithm) to achieve a signature *“with appendix”* which means message is first processed with the Hash function then signed. SHA-1 security has been reduced to 261 and is no longer recommended by NIST for digital signatures. SHA-3 was proposed through a NIST competition which ended on October, 2012 with a winning algorithm by Kecccak. SHA-2 is available in Crypto++ but not used in the Digital Cash project due to technical difficulties. Whirlpool and PKCS version 1.5.

**SignerFilter:**

SignerFilter(RandomNumberGenerator &rng, const PK\_Signer &signer,

BufferedTransformation \*attachment=NULL, bool putMessage=false)

**SignatureVerificationFilter Constructor:**

SignatureVerificationFilter(const PK\_Verifier &verifier, BufferedTransformation

\*attachment = NULL, word32 flags = DEFAULT\_FLAGS)

**Signature Code from Digital Cash Project:**

static void BlindSignature(string& MO\_par, RSA::PrivateKey my\_privateKey)

{

string message = MO\_par, signature, recovered;

AutoSeededRandomPool rng;

//

// Sign and Encode

RSASS<PSSR, SHA1>::Signer signer(my\_privateKey);

StringSource ss1(message, true,

new SignerFilter(rng, signer,

new StringSink(signature),

true // putMessage for recovery

) // SignerFilter

); // StringSource

MO\_par = signature;

}

**Verification and Recovery Code from Digital Cash Project:**

string BankBlindSignatureRecovery(string MO\_par, RSA::PublicKey my\_publicKey)

{

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Verify and Recover

string signature = MO\_par;

string recovered;

////////////////////////////////////////////////

// Verify and Recover

RSASS<PSSR, SHA1>::Verifier verifier(my\_publicKey);

StringSource ss2(signature, true,

new SignatureVerificationFilter(

verifier,

new StringSink(recovered),

SignatureVerificationFilter::THROW\_EXCEPTION | SignatureVerificationFilter::PUT\_MESSAGE

) // SignatureVerificationFilter

); // StringSource

cout << "Verified signature on message" << endl;

return recovered;

}

**PROGRAM TRACE AND EXPLANATION:**

**COMPLETE PROGRAM:**

**Objects:**

Customer Alice;

BankMoneyOrder AliceBMO;

Merchant MerchantMO;

int main(int argc, char\* argv[])

{

// CUSTOMER PROCESS:

1. Alice.GetCustomerInfo(Alice, NumberOfOrders);
2. Alice.CreateCustomerMO(Alice, Alice.CustomerMOamt, NumberOfOrders);
3. SelectBankMoneyOrder(Alice, AliceBMO);
4. AliceBMO.CustomerPublicKey = Alice.CustomerPublicKey;
5. UnblindOrders(Alice, AliceBMO);
6. VerifyOrders(Alice, AliceBMO);
7. UnblindMerchantOrder(Alice, AliceBMO, MerchantMO);

system("pause");

return 0;

**Function Descriptions:**

**1ST PROGRAM LINE: Gets customer ID and money order amount from user.**

void Customer::GetCustomerInfo(Customer& NewCustomer\_Par, int numOrders\_par)

{

cout << "Enter New Customer ID" << endl;

getline(cin, NewCustomer\_Par.CustomerID);

cout << "Enter Money Order Amount" << endl;

getline(cin, NewCustomer\_Par.CustomerMOamt);

NewCustomer\_Par.numOrders = numOrders\_par;

cout << endl << endl;

}

**2nd PROGRAM LINE:**

**Creates the Customer’s Private and Public keys.**

**Duplicates money order encrypts/signs and stores in class vector**

void Customer::CreateCustomerMO(Customer& NewCustomer\_par, string amount, int numOrders\_par)

{

cout << "DUPLICATE MONEY ORDERS, CREATE AND STORE KEYS, BLIND ALL ORDERS" << endl;

string IDleft\_par = "987654321";

string IDright\_par = NewCustomer\_par.CustomerID;

stringstream ss;

//generate private and public keys for bank blind signature

AutoSeededRandomPool rng;

InvertibleRSAFunction parameters;

parameters.GenerateRandomWithKeySize(rng, 1024);

RSA::PrivateKey CustomerPrivateKey(parameters);

RSA::PublicKey CustomerPublicKey(parameters);

NewCustomer\_par.CustomerPrivateKey = CustomerPrivateKey;

NewCustomer\_par.CustomerPublicKey = CustomerPublicKey;

for (int i = 0; i < 5; i++)

{

string MOamount = amount;

cout << "Pre-blinded Amount for Order # : " << i << " is: " << MOamount << endl;

BlindSignature(MOamount, CustomerPrivateKey);

NewCustomer\_par.MOvector[i].MOamount = MOamount;

cout << "RSA Blinded Amount : " << NewCustomer\_par.MOvector[i].MOamount << endl;

string Ustring = GenerateUString();

NewCustomer\_par.MOvector[i].Ustring = Ustring;

cout << "Ustring : " << NewCustomer\_par.MOvector[i].Ustring << endl;

string IDright = BitXOR(IDleft\_par, IDright\_par);

BlindSignature(IDright, CustomerPrivateKey);

NewCustomer\_par.MOvector[i].IDright = IDright;

cout << "RSA Blinded IDright this is the XOR: " << NewCustomer\_par.MOvector[i].IDright;

string IDleft = IDleft\_par;

BlindSignature(IDleft, CustomerPrivateKey);

NewCustomer\_par.MOvector[i].IDleft = IDleft;

cout << "RSA Blinded IDleft after XOR: " << NewCustomer\_par.MOvector[i].IDleft << endl;

}

system("pause");

}

**3rd PROGRAM LINE:**

**Selects random money order to leave blinded so n-1 orders can be verified as well formed.**

**Random index calculated in the Bank Money Order constructor.**

void SelectBankMoneyOrder(Customer Customer\_par, BankMoneyOrder& NewBankMO\_par) //extracts randomly selected money order to keep blinded

{

//selects random money order to sign. Unblinds and sends to Merchant.

int randIndex = NewBankMO\_par.randIndex;

cout << "SELECT RANDOM MONEY ORDER AT INDEX: " << randIndex << "TO LEAVE BLINDED" << endl;

//enter selected money order into bank money order

NewBankMO\_par.MOamount = Customer\_par.MOvector[randIndex].MOamount;

cout << "Signed MO amount " << Customer\_par.MOvector[randIndex].MOamount << endl;

NewBankMO\_par.Ustring = Customer\_par.MOvector[randIndex].Ustring;

cout << "Signed MO Ustring " << Customer\_par.MOvector[randIndex].Ustring << endl;

NewBankMO\_par.IDleft = Customer\_par.MOvector[randIndex].IDleft;

cout << "Signed MO IDleft " << Customer\_par.MOvector[randIndex].IDleft << endl;

NewBankMO\_par.IDright = Customer\_par.MOvector[randIndex].IDright;

cout << "Signed MO IDright " << Customer\_par.MOvector[randIndex].IDright << endl <<endl;

system("pause");

}

**4th PROGRAM LINE:**

**Passes Customer’s Public Key to the bank.**

AliceBMO.CustomerPublicKey = Alice.CustomerPublicKey;

**5th PROGRAM LINE:**

**Orders are unblinded and signature verified.**

void UnblindOrders(Customer& Customer\_Par, BankMoneyOrder NewBankMO\_par)

{

int randIndex = NewBankMO\_par.randIndex;

string ID\_par;

cout << "UNBLIND REMAINING ORDERS FOR VERIFICATION: "<< endl;

for (int i = 0; i < 5; i++)

{

if (i != randIndex) //skips money order that is left blinded

{

cout << "Ustring : " << Customer\_Par.MOvector[i].Ustring << endl;

string BMOamount = Customer\_Par.MOvector[i].MOamount;

Customer\_Par.MOvector[i].MOamount = BankBlindSignatureRecovery(BMOamount, Customer\_Par.CustomerPublicKey);

cout << "Unblind BMOamount: " << Customer\_Par.MOvector[i].MOamount << endl;

string BMO\_IDleft = Customer\_Par.MOvector[i].IDleft;

Customer\_Par.MOvector[i].IDleft = BankBlindSignatureRecovery(BMO\_IDleft, Customer\_Par.CustomerPublicKey);

cout << "Unblind IDleft: " << Customer\_Par.MOvector[i].IDleft << endl;

string BMO\_IDright = Customer\_Par.MOvector[i].IDright;

Customer\_Par.MOvector[i].IDright = BankBlindSignatureRecovery(BMO\_IDright, Customer\_Par.CustomerPublicKey);

cout << "Unblind IDright: " << Customer\_Par.MOvector[i].IDright << endl;

Customer\_Par.MOvector[i].IDright = BankBitXOR(Customer\_Par.MOvector[i].IDleft, Customer\_Par.MOvector[i].IDright);

cout << "Unblind Recombined ID after XOR: " << Customer\_Par.MOvector[i].IDright<<endl<<endl;

}

}

system("pause");

**6th PROGRAM LINE:**

**Orders are checked that they are all well formed by comparing that all data is the same for each money order.**

void VerifyOrders(Customer& Customer\_Par, BankMoneyOrder NewBankMO\_par)

{

int randIndex = NewBankMO\_par.randIndex;

string AmountOld\_par = "0";

string ID\_par;

string ID\_old\_par;

int old\_par\_count = 0;

for (int i = 0; i < 5; i++)

{

if (i != randIndex && (old\_par\_count == 0)) //skips money order that is left blinded

{

AmountOld\_par = Customer\_Par.MOvector[i].MOamount;

old\_par\_count++;

}

if (i != randIndex) //skips money order that is left blinded

{

if (Customer\_Par.MOvector[i].MOamount != AmountOld\_par)

{

cout << "amount: " << Customer\_Par.MOvector[i].MOamount << "old Amount :" << AmountOld\_par;

cout << "error - orders not well formed";

}

}

AmountOld\_par = Customer\_Par.MOvector[i].MOamount;

}

cout << "Orders are all well formed " << endl<<endl;

system("pause");

}

**7th PROGRAM LINE:**

**Money order is unblinded and forwarded to the Merchant class.**

void UnblindMerchantOrder(Customer Customer\_Par, BankMoneyOrder NewBankMO\_par, Merchant& NewMerchantMO)

{

int randIndex = NewBankMO\_par.randIndex;

string ID\_par;

cout << "UNBLIND RANDOM MONEY ORDER / FORWARD TO MERCHANT: " << endl;

cout << "Merchant MO Ustring : " << Customer\_Par.MOvector[randIndex].Ustring << endl;

NewMerchantMO.Ustring = Customer\_Par.MOvector[randIndex].Ustring;

string BMOamount = Customer\_Par.MOvector[randIndex].MOamount;

NewMerchantMO.MOamount = BankBlindSignatureRecovery(BMOamount, Customer\_Par.CustomerPublicKey);

cout << "Unblind Merchant MO amount: " << NewMerchantMO.MOamount << endl;

string BMO\_IDleft = Customer\_Par.MOvector[randIndex].IDleft;

NewMerchantMO.IDleft = BankBlindSignatureRecovery(BMO\_IDleft, Customer\_Par.CustomerPublicKey);

cout << "Unblind Merchant MO IDleft: " << NewMerchantMO.IDleft << endl;

string BMO\_IDright = Customer\_Par.MOvector[randIndex].IDright;

NewMerchantMO.IDright = BankBlindSignatureRecovery(BMO\_IDright, Customer\_Par.CustomerPublicKey);

cout << "Unblind Merchant MO IDright: " << NewMerchantMO.IDright << endl;

NewMerchantMO.IDright = BankBitXOR(Customer\_Par.MOvector[randIndex].IDleft, Customer\_Par.MOvector[randIndex].IDright);

cout << "Unblind Recombined ID after XOR: " << NewMerchantMO.IDright << endl << endl;

system("pause");

}

**//Angela Fogel**

**//CS625 Cryptography**

**//Digital Cash Protocol #4**

**June 24, 2014**

**Project Errors and Omissions:**

**Merchant bit stream to select random left or right identity strings. Omitted because I was unable to get the bitXOR function to work properly.**

**Left and right halves of the identity string not repeated (n) times due to the above.**

**Objects not passed to separate Customer, Bank, Merchant programs and files but instead passed using class objects in the same program. So, more of a simulation than a money order program.**

**Uniqueness string not checked by the bank. – Oversight found at the last minute but too late to produce a solution.**