Class Project: KoolKash 1.0
Final Report

CS55500 - Cryptography

William Harding and Jonathan McCluskey

April 4, 2015

Abstract

The authors present their implementation of a digital cash protocol, using RSA blind signatures, bit commitment using sha256, and secret splitting. Three executables are used, a Bank, a Buyer, and a Merchant. Using the three executables the digital cash protocol is run through its entirety with the Buyer having made an anonymous purchase.

Contents

1 Problem Statement

1.1 Overview

Given money in a bank account, a buyer wishes to make an anonymous purchase using that money; the merchant selling to the buyer wishes to avoid being cheated and wants deposit the money (resulting from the anonymous transaction) into his/her bank account; and the bank wishes neither to be cheated nor to lack recourse if (or when) it is cheated. The protocol, in the following section, seeks to accomplish our needs.

1.2 Protocol

The following protocol is pulled directly from Protocol #4 from Section 6.4 in [1]:

1. Alice prepares n anonymous money orders for a given amount. Each of the money orders contains a different random uniqueness string, X, one long enough to make the chance of two being identical negligible. On each money order, there are also n pairs of identity bit strings, I_1, I_2, ..., In. (Yes, that's n different pairs on each check.) Each of these pairs is generated as follows: Alice creates a string that gives her name, address, and any other piece of identifying information that the bank wants to see. Then, she splits it into two pieces using the secret splitting protocol (see Section 3.6). Then, she commits to each piece using a bit-commitment protocol. For example, I_{37} consists of two parts: I_{37_L} and I_{37_R}. Each part is a bit-committed packet that Alice can be asked to open and whose proper opening can be instantly verified. Any pair (e.g., I_{37_L} and I_{37_R}, but not I_{37_L} and I_{38_R}), reveals Alice's identity.

2. Alice blinds all n money orders, using a blind signature protocol. She gives them all to the bank.

3. The bank asks Alice to unblind n 1 of the money orders at random and confirms that they are all well formed. The bank checks the amount, the uniqueness string, and asks Alice to reveal all of the identity strings.

4. If the bank is satisfied that Alice did not make any attempts to cheat, it signs the one remaining blinded money order. The bank hands the blinded money order back to Alice and deducts the amount from her account.

5. Alice unblinds the money order and spends it with a merchant.

6. The merchant verifies the bank's signature to make sure the money order is legitimate.

7. The merchant asks Alice to randomly reveal either the left half or the right half of each identity string on the money order. In effect, the merchant gives Alice a random n-bit selector string, b_1, b_2, ... b_n. Alice opens either the left or right half of I_i, depending on whether b_i is a 0 or a 1.

8. Alice complies.

9. The merchant takes the money order to the bank.

10. The bank verifies the signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits the amount to the merchant's account. The bank records the uniqueness string and all of the identity information in a database.

11. If the uniqueness string is in the database, the bank refuses to accept the money order. Then, it compares the identity string on the money order with the one stored in the database. If it is the same, the bank knows that the merchant copied the money order. If it is different, the bank knows that the person who bought the money order photocopied it. Since the second merchant who accepted the money order handed Alice a different selector string than did the first merchant, the bank finds a bit position where one merchant had Alice open the left half and the other merchant had Alice open the right half. The bank XORs the two halves together to reveal Alice's identity.

In the following protocols let Alice (A) be the Buyer, Bob (B) be the Merchant, and Credit Union (C) be the Bank. These protocols assume that the banks has already generated an RSA public/private key pair.

Alice prepares n money orders in the following way:

1. Alice generates her identity string, i
2. Alice generates a unique string, u
3. Alice generates 100 random numbers, r_j, for $j = 0...99$, and secret splits her identity string, $s_j = r_j \oplus i_j$
4. Alice bit commits the values r_j and s_j in the money order using a one-way hash function (as described in [1])
5. Alice sets the desired amount of money value in the money order
6. Alice blinds, b_i, all n money orders, m_i, $b_i = m_i * k^e$ for $i = 0...99$. k is a random number between 1 and the Credit Union's public key modulus which is also relatively prime with the public modulus. e is the Credit Union's public key.

2 Proposed Solution

2.1 Design

We broke the problem out into three separate executables, Bank, Buyer, and Merchant, which share common source code and libraries. The executables talk via socket connections and file transfers.

The Buyer begins the transaction by connecting to the Bank as a client. The Buyer and Bank will exchange money orders, blinding factors, bit committed identity strings, etc resulting in a signed money order that the Buyer writes to a file.

The Buyer then, perhaps at some later time, connects to Merchant as a client. The Buyer sends the money order in exchange for some item, but before the Merchant provides the item the bank's signature is verified, the Merchant requests that the Buyer reveal half of each secret-split identity string, and the Merchant deposits the valid money order.

The Bank needs to be sure that the money order is valid, and that it hasn't seen it before. Once it's determined that, the money order is deposited.

The merchant is then free to provide the item to the Buyer.

2.1.1 Third Party Dependencies

Our implementation relies heavily upon the following third party libraries and tools:

- boost (for unit tests, socket i/o, serialization)
- libgmp (GNU multi-precision library)
- libgcrypt (GNU cryptography library)
- clang (as our compiler)
- scons (as our build tool)

2.2 Implementation

To support our efforts we built up a cryptographic library which we call libcrypto. This contained some of the core cryptographic functions needed to complete the whole protocol. The following functions were included in our library:

- RSA (decipher/encipher) (sign/verify)
- bit commitment
- blind signature
- secret splitting

With the help of libgmp, we implemented all of the above functionality ourselves with the exception of sha256 (used in bit commitment), which we are performing through the using of libgcrypt. Although the digital cash protocol only needs RSA signatures, our implementation of RSA includes encipher/decipher as well.

3 Conclusions

3.1 Testing and Results

Timing Tests Our timing results as performed on an amd64 bit architecture within our unit test framework are listed below:

- RSA Encipher (1024 bit) - 0.000211282s
- RSA Decipher (1024 bit) - 0.0203434s
- RSA Signature (1024 bit) - 0.0199427s
- RSA Verify (1024 bit) - 0.000167771s
- Bit Commitment - 0.0043509s
- Blind Signature - 0.002142s
- Unblind Signature - 0.000041931s
- Prime Generator (512 bit) - 0.00439004s

Unit Tests We unit tested all of the following pieces:

- RSA (encipher/decipher/sign/verify)
- Bit Commitment
- Slind Signatures
- Secret Splitting
- Extended Euclidean Algorithm
- Fast Exponentiation
- Money Order Serialization/Deserialization

All of the unit tests perform as expected.

System Tests The results of our system testing were successful. We can run through the entire protocol for buyer, bank, and merchant; with the transaction occurring normally. Additionally, our boundary cases (wherein there is a cheating party) were tested and we found our implementation of the protocol to be robust, providing the correct mechanisms to defeat and/or deter cheating. Our system tests included testing the following boundary cases:

- Buyer cheats by attempting to spend the money order more than once
- Merchant cheats by attempting to cash the money order more than once

3.2 Future Work

Additional future work includes:

- Using a more secure RSA implementation
- Add a database for the bank to maintain users and account balances

3.3 Conclusion

In conclusion, we have presented our implementation of a digital cash protocol which provides a buyer the ability to spend cash with a level of anonymity. We believe that this is a robust protocol, and that our implementation could be used as a stepping stone for a real digital cash system.

References

[1] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.