Analysis of an Anonymous Fair Exchange Protocol

Adam Barth and Andrew Tappert

We modeled and analyzed the protocol described in Indrakshi and Indrajit Ray's paper "An Anonymous Fair Exchange E-commerce Protocol" [1]. We modeled the protocol in the Reactive Modules language of Mocha [2], and used that tool to test claims made of the protocol by the authors, which we formalized as propositional logic invariants and alternating-time temporal logic (ATL) formulas. In the course of writing the model and specifications we came upon several potential attacks.

The protocol was designed to allow a customer to exchange payment for a merchant's digital product, such that fairness of the exchange is ensured and anonymity of each party is guaranteed. The protocol relies on the presence of a third party during a prelude phase, but it is optimistic in that the third party is not involved in the exchange phase unless a problem occurs. Additionally, the authors claim their protocol guarantees the customer can verify he is obtaining the correct product by comparing what he downloads from the third party in the prelude (the product encrypted under a particular key) with what the merchant sends in message two (the product encrypted under a special "cross-key"). We did not attempt to model or verify this "cross-validation."

In the prelude, the merchant first supplies the third party with the product he wishes to sell, a key K1 and its inverse, and a public key chosen for the transaction (not one associated with the merchant's true identity, if the merchant desires anonymity). A customer who wishes to buy the product downloads the product encrypted under K1 and the merchant's public key from the third party, then generates a temporary public-private key pair for use in the transaction.

For the actual exchange itself, eight messages are involved a trace of the protocol in which nothing goes wrong. These messages do not involve the third party, only the customer, the customer's bank, the merchant, and the merchant's bank. In the first two, the customer notifies the merchant of his desire to buy the product by sending a purchase order and the merchant responds by accepting (signing) the purchase order and sending the product (again encrypted, but now under cross-key K1xK2) and his account information (encrypted with his bank's public key). In the remaining messages, the customer asks his bank for a payment token made out to the merchant's account in the correct amount, and having received it sends it on to the merchant, who deposits it with his bank. All this achieved successfully, the merchant sends the decryption key to the customer. At various stages abort messages may be sent (for example, by the merchant if he does not accept the purchase order sent by the customer, or by the customer if his bank tells him he does not have sufficient funds to produce the payment token needed for the transaction). This brief description should be enough to understand the analysis which follows; pertinent details will be presented as necessary.

Once the first two messages have been exchanged and the customer has obtained a payment token from his bank, he can call upon the third party to force resolution of the transaction by sending him the messages and the payment token. Only the customer can call upon the third party in this protocol, since the merchant only ever sends the decryption key after being assured of payment by his bank. But sending message two, with the signed purchase order, commits the merchant in the eyes of the third party, which leads to an imbalance in favor of the customer, who is then in a position from which he can either force the transaction to occur, by invoking the third party, or to be aborted, by refusing to send payment. This imbalance does not seem to be correctable by any small modification to the protocol.

Balance requirements for our Mocha model can be given with the following ATL:

```
-- cbal: customer balance
atl "cbal" (~(<< n, hcb >> F ((<< n >> F nprod) & ~(<< hm >> F mpay))));

-- mbal: merchant balance
atl "mbal" (~(<< n >> F ((<< n >> F npay) & ~(<< hc, hcb >> F cprod))));
```

Here `n` refers to "the network" which models whichever parties are chosen to be dishonest in a given run. Thus `cbal`, in the context of an `n` with a dishonest customer, reads, "There does not exist a strategy for a dishonest customer to reach a state in which both he has a strategy for obtaining the product and the merchant does not have a strategy for receiving payment." The formula `mbal` states a similar property, with roles interchanged, for merchant balance. We attempted to use Mocha's ATL checker to test these properties, but it has not returned us an answer as of this writing, having run for more than a week.

In fact, we had several problems working with the Mocha model checker. We used the older version, C-Mocha, because the newer version, J-Mocha, oddly lacks the ability of the older to check ATL formulas. C-Mocha, unfortunately, is not very robust, often crashing when presented with models with errors, and never producing useful error messages. We encountered a more serious difficulty when our model to grow large. In the process of constructing our model incrementally, building from a simple model of part of the protocol to a more complete model of the entire thing, we reached a point when Mocha would simply crash with a segmentation fault whenever we attempted a simulation. After long and frustrating debugging efforts, we finally concluded that Mocha, absurdly, had a maximum limit on the combined length of all the identifiers in a model! Ante managed to track down the bug in Mocha and recompile it, but in the mean time, resorting to a minimalist naming convention, we were able to complete a model of the protocol, albeit with a somewhat simplified version of the third party (upon receiving a message from the customer, it immediately resolves the transaction by sending payment to merchant and product to customer, rather than prodding the merchant to resume the protocol first).

A side effect of the protocol's guarantee of anonymity is that a dishonest bank can create a payment token, which it might not honor, and play the role of the customer. The paper's description of a payment token is simply an amount to credit, the account to be credited (encrypted with the public key of the bank at which the account is held), and a nonce to prevent replay attacks, all encrypted under a key shared by all banks. Given this anonymous sort of payment, neither the merchant nor the merchant bank can learn the identity of a malicious bank which creates bad payment tokens. This problem also does not seem correctable in the context of the protocol's objectives, in particular given the necessity of protecting the customer's identity.

Two other attacks we discovered could easily be prevented. The first problem we came upon was the possibility of a man-in-the-middle attack, in which an intruder is able to obtain the product. An intruder can generate his own temporary public-private key pair, as the customer did in the prelude, intercept the customer's messages, relay them to the merchant with his own keys, and forward the responses back to the customer who is unable to discern anything amiss. This attack is possible only because the merchant never confirms public key the customer sends him. In message one, the customer sends his own public key under the merchant's public key. If, for example, in message two the merchant included the customer's public key under his private key, the customer could detect the man-in-the-middle attack.

The last problem we found was in the operation of the third party. The paper does not treat this part of the protocol as thoroughly as the optimistic portion, neglecting to give a message by message description. Nevertheless, a problem was apparent. As mentioned above, the third party becomes involved only by the customer's invocation. The customer sends messages one and two and the payment token he received from his bank. The third party then contacts the merchant, asking him to please send the decryption key to the customer (very politely, just like that). The merchant at this point can comply, ignore the message, or reply to the third party that he has not yet received payment. In either of the former cases, the customer is supplied with a decryption key (the inverse of either K1 or K2, either of which is sufficient). In the latter case, however, the third party forwards the payment token to the merchant, and assumes he will continue the protocol. Clearly, he can choose not to do so, and can continue to deny that he has received payment when the customer invokes the third party again. What we did in our simplified model to cope with the limits of Mocha actually might make be better version of the protocol: dispense with the pointless politeness, and simply send payment to merchant and key to customer whenever the third party is invoked.

[1] Indrakshi Ray and Indrajit Ray. "An Anonymous Fair-Exchange E-Commerce Protocol." *Proceedings of the First International Workshop on Internet Computing and E-Commerce*. San Francisco, CA, April 2001.

http://www.cs.colostate.edu/~iray/research/icec01.pdf

[2] Rajeev Alur, Thomas A. Henzinger, F.Y.C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. "Mocha: Modularity in model checking." *Proceedings of the Tenth International Conference on Computer-aided Verification (CAV 1998)*. Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, pp. 521-525.

http://www-cad.eecs.berkeley.edu/~mocha/