Accountable Mobile Systems
A draft paper

Ali Shoker

December 23, 2014

Abstract

This draft describes the protocol used in the Accountable Messaging System (AMS). The porpose of the protocol is force any contributing node in message forwarding not to discard forwarding others messages (e.g., to save its own resources like energy, CPU, and network bandwidth). The main idea is to send each message in two rounds where the first hides the destination of the message (using encryption) and the second sends the decryption key. This simplifies tracking rational (selfish) nodes and evicting them. This is made possible by using an encrimental secure log as in PeerReview [2].

1 Introduction

Message forwarding is a fundamental approach in Mobile cooperative systems. Since mobile resources are often scarce, nodes can discard forwarding others messages to save resources. Consequently, nodes should have incentive to forward others messages. Existing approaches (credit-based, TFT-based, reputation-based) do not resolve this problem. Other game theory solutions like Give2Get [4] are limited to closed area (in-door) networks. Accountability solutions that uses secure logging, e.g., PeerReview, can be an alternative solution through instant misbehavior detection. However, PeerReview [2] is not applicable for Mobile networks due to sync assumptions, witness, limited resources, and complexity.

This draft introduces AMS: an Accountable system for Mobile systems against rational nodes with the follwoing features: (1) anonymous messaging as in give2Get, (2) accountability using secure logs, similar to PeerReview, with reduced resource requirements and complexity, (3) generic, no hard assumptions, (4) concrete implementation and experimentation.

2 Protocol

The protocol is composed of two phases: Validation and Communication. Validation phase is used to test the trustworthy of the other nodes. If a node is trusted, the

Communication phase can be launched. Before describing the two phases, we present our system model and some notations we use in this paper.

2.1 System Model

The system is composed of multiple nodes that can interact over a wireless intermittent connection in a mobile network. We do not enforce any kind of synchrony between nodes; however, better synchrony yields a better performance . Nodes have limited resources like processing power, battery power, and storage. We assume that each node can be selfish (trying to save its resources) but neither malicious nor it can collude. To avoid Sybil attacks [1], a trusted authority assigns each node a unique identifier ID (and a corresponding certificate to ensure ID uniqueness). Each node generates a pair of public/private key corresponding to its ID. Public keys can be distributed to all nodes using any method similar to cite.

To enforce accountability against selfish nodes, we provide a solution inspired from [3] and [2]; however, we make many improvements in order to fit our system model (i.e., intermittent connectivity and limited resources). We require that each node has a secure log SL to store its executed operations in a way that allows other nodes to verify log correctness and consistency. A secure log SL is an append-only list of records appended in a chronological order; it is composed of a concatenation of Verified Secure Logs VSL and a single Unverified Secure Log USL, i.e., $SL = VSL_1||...||VSL_i||USL$. VSL_i is a sequence of hash values of the executed operations by the node itself, and appended with a verification certificate from another node. A certificate is a signed hash value of the verified log (i.e., a hash of the hash values). USL is a sequence of hash values of operations executed by the node without being verified yet. In USL, each hash value is associated with an entry $e_k = (s_k, t_k, o_k)$ with a sequence number s_k , an operation type t_k , and an operation o_k (we may change this a bit) . The sequence numbers must be strictly increasing. A hash value h_k is recursively computed from h_{k-1} and e_k as follows: $h_k = H(h_{k-1}||s_k||t_k||H(o_k))$. An authenticator $\alpha_k^X = \sigma^X(s_k, h_k)$ is a signed statement by node X (using its private key) that its log entry e_k^X has hash value h_k^X. In addition, each node keeps a Validation Authenticator hash Table VHT which is used to store the VSLs of the other nodes that were exchanged during validation.

To make the presentation easier, and without loss of generality, for any three nodes X, Y, and Z, we assume that $SL_X = VSL_Y||VSL_Z||USL$. On a fine-grained scale, $SL_X = h_1^X, h_2^X, ..., h_i^X, c_i^{X/Y}, h_{i+1}^X, ..., h_v^X, c_v^{X/Z}, h_{v+1}^X, ..., h_j^X, .., h_n^X$; where h_i^X represents the i^{th} hash value of X, and $c_v^{X/Y}$ represents the verification certificate of X's log from h_i^X to h_v^X delivered by Y. We denote by $h_k^{X/Y}$ the hash value of an entry e_k^X of X computed by node Y. In addition, we assume that communication between nodes is bound by timers (adjusted with care); however, we do not mention timers in the algorithm for simplicity.

2.2 Communication Phase

Assume that the last entry of X's and Y's USL are h_n^X and h_m^Y, respectively.

1. **X encrypts Anonym message to hide its final destination, adds it to its log, and sends it to Y.**

   Additional details. X prepares an Anonym message by encrypting the message payload m and the message destination D_m in the message content $o_{n+1} = Anonym(\sigma^X(< m, D_m >))$ using the private key of X and a randomly generated key K). Then, X creates a hash value h_{n+1}^X for a corresponding entry $e_{n+1}^X = (s_{n+1}, SEND, o_{n+1})$ and appends them to USL. Then, X creates two authenticators a_n^X and a_{n+1}^X corresponding to h_n^X and h_{n+1}^X, respectively. After that, X sends the last entry e_{n+1}^X along with the two authenticators a_n^X and a_{n+1}^X to Y.

   Proof: X will send the encrypted Anonym message correctly with the correct authenticators. X has some message content to deliver to Y or to be forwarded by Y; thus, X has no interest to misbehave in sending Anonym. If Anonym has not encrypted the message, Y might drop it if the message is not destined to it; even more, this free-riding will appear in X's and Y's USLs and they will be accused by other nodes during validation. On the other hand, X must add Anonym to its log and send the correct authenticator otherwise Y will not forward its message and will add X to its blacklist.

2. **Y receives Anonym from X, verifies if X added Anonym to its log, and sends an ASK message to X asking for the key to decrypt Anonym.**

   Additional details. Y receives Anonym from X and computes $h_{n+1}^{X/Y}$ using h_n^X and e_{n+1}^X (after verifying the signatures of the corresponding authenticators). Then Y verifies if X has added Anonym to its log by comparing $h_{n+1}^{X/Y}$ with h_{n+1}^X. If not, Y adds X to its blacklist. Otherwise, Y adds Anonym content to its log in a new entry $e_{m+1}^Y = (s_{m+1}, RECV, o_{m+1})$; where $o_{m+1} = h_{n+1}^X$ (to save storage resources) . Then, it prepares an ASK message to ask for the key of the Anonym message in order to decrypt it. It puts a_{n+1}^X in the message content o_{m+2} and creates a hash value h_{m+2}^Y with a corresponding entry $e_{m+2}^Y = (s_{m+2}, ASK, o_{m+2})$ and appends them to USL. Then, Y creates two authenticators a_{m+1}^Y and a_{m+2}^Y corresponding to h_{m+1}^Y and h_{m+2}^Y, respectively. After that, Y sends the last entry e_{m+2}^Y along with the two authenticators a_{m+1}^Y and a_{m+2}^Y to X.

   Proof: Y wont drop Anonym and will send ASK correctly. Y wont drop Anonym and will send ASK since it is not sure if the message is destined to it or not. In addition, Y will add its authenticator and entries correctly to allow X validate its log otherwise X wont trust Y and send the key.

3. **X receives ASK from Y, verifies if Y has added ASK to its log, and replies with a KEY message.**

   Additional details. X receives ASK from Y and computes $h^{Y/X}_{m+2}$ using h^{Y}_{m+1} and e^{Y}_{m+2} (after verifying the signatures of the corresponding authenticators). Then, X verifies if Y has added ASK to its log by comparing $h^{Y/X}_{m+2}$ with h^{Y}_{m+2}. If not, X adds Y to its blacklist. Otherwise, X adds ASK content to its log in a new entry $e^{X}_{n+3} = (s_{n+3}, RECV, o_{n+3})$; where $o_{n+3} = h^{Y}_{m+2}$ (to save storage resources) . Then, it prepares a KEY message to send the key K of the Anonym message to Y. It puts the key K in the message content o_{n+4} and creates a hash value h^{X}_{n+4} with a corresponding entry $e^{X}_{n+4} = (s_{n+4}, KEY, o_{n+4})$ and appends them to USL. Then, X creates two authenticators a^{X}_{n+3} and a^{X}_{n+4} corresponding to h^{X}_{n+3} and h^{X}_{n+4}, respectively. After that, X sends the last entry e^{X}_{n+4} along with the two authenticators a^{X}_{n+3} and a^{X}_{n+4} to Y.

   Proof: X wont drop ASK and will send the KEY correctly. S wont drop ASK and will send the KEY message correctly to allow Y to read and deliver its previously sent Anonym message. The KEY message should include correct authenticators otherwise Y will detect a misbehavior and will add X to its blacklist.

4. **Y receives Key from X and decrypts Anonym. If the final destination of Anonym is Y, this later uses the messages, otherwise, Y forwards the messages to the final destination (or another forwarder) node. If Y did not receive KEY, it forwards the encrypted Anonym to any N different nodes (to prevent selfishness).**

   Additional details. Y receives the KEY message from X and computes $h^{X/Y}_{n+4}$ using h^{X}_{n+3} and e^{X}_{n+4} (after verifying the signatures of the corresponding authenticators). Then Y verifies if X has added KEY to its log by comparing $h^{X/Y}_{n+4}$ with h^{X}_{n+4}. If not, Y adds X to its blacklist. Otherwise, Y adds KEY content to its log in a new entry $e^{Y}_{m+3} = (s_{m+3}, RECV, o_{m+3})$; where $o_{m+3} = h^{X}_{n+4}$ (to save storage resources). Now, Y used the key K to decrypt the Anonym message. If Y noticed that the message is destined to it, it executes it. Otherwise, Y forwards the message to the destination (or to another bridge node). In case Y did not receive the KEY message of Anonym (whether X has not sent it or the message was lost), Y must keep forwarding the encrypted Anonym message until receiving N ASK messages from different nodes; these messages stand as a proof in Y's log that it is not misbehaving (e.g., by dropping the KEY message from X).

   Proof: Y will not drop Anonym or KEY, and will forward the message. Y wont drop KEY before decrypting Anonym (obvious). After decryption, Y wont

drop neither Anonym nor KEY if it is destined to it (obvious). If the message destination is not Y itself, Y will forward it to the destination in order to avoid sending it to N nodes which is more expensive; otherwise, a misbehavior will be detected during the next validation phase.

2.3 Validation Phase

Assume that the last entry of X's and Y's USL are h_n^X and h_m^Y, respectively.

1. **X sends a Valid message to Y with its VSL, the entries of USL, and the authenticator corresponding to h_{n+1}^X.**

   Additional details. X prepares a validation message $Valid$; then it creates a hash value h_{n+1}^X for the corresponding entry $e_{n+1}^X = (s_{n+1}, SEND, o_{n+1})$, and appends them to USL. The Valid message must be accompanied with the verified secure log VSL, the entries $e_{v+1}^X, .., e_{n+1}^X$ (without their hash values), and the VHT table. In addition, X creates an authenticator a_{n+1}^X corresponding to h_{n+1}^X and sends it along with the Valid message to node Y.

   Proof: X must validate with Y and must send the correct Valid message. (1) X needs to trust Y to communicate with it, otherwise it might lose its resources with unsuccessful communication. (2) X can reduce its storage by deleting the recent entries in USL. (3) X can detect if other nodes are trusted or not (i.e., they did not use diff. log versions for diff. nodes) by checking if its previous communication entries (hash values) are in Y's VHT (since Y will send its VHT to X too). (4) X is afraid of being accused by Y if it did not initiate a validation (since it assumes Y is altruistic). (5) Other nodes can notice from X's and Y's SL that X is communicating with nodes without validating its log, and thus it will be accused. On the other hand, X will not send an incorrect Valid message for the same reasons.

2. **Y receives Valid message from X and verifies if X has computed its log hash sequence correctly. The it checks if its previous hash values, if any, are present in X's log or VHT; this is crucial to check for log fragmentation of nodes. Then, Y sends a VACK message to X with its VSL, the entries of USL, and the authenticator corresponding to h_{m+2}^Y.**

   Additional details. Y receives Valid from X and logs a corresponding entry $e_{m+1}^Y = (s_{m+1}, RECV, o_{m+1})$ in its USL after verifying the signature. Then, Y validates S's log in three steps. First, it creates all the hash values of X's USL starting from h_v^X until h_{n+1}^X by using the entries $e_{v+1}^X, .., e_{n+1}^X$. if the calculated hash value $h_{n+1}^{X/Y}$ matches the received hash value h_{n+1}^X (obtained after decrypting a_{n+1}^X), then Y knows that X has not tampered with its USL. Second, Y check if

X is misbehaving by dropping some Anonym messages that are supposed to be forwarded. This is done by checking whether every RECV entry of an Anonym message in X's USL is either followed by a SEND entry of the same message or followed by N different RECV entries of ASK messages corresponding to Anonym (ensuring that X used to forward Anonym messages to N nodes if it had not received their KEYs). Third, Y verifies if the hash values in X's VSL match the corresponding certificates, i.e., if $H(h_1^X, h_2^X, ..., h_i^X)$ matches $h_i^{X/Y}$ in the certificate $c_i^{X/Y}$, and $Hash(h_{i+1}^X, ..., h_v^X)$ matches $h_v^{X/Z}$ in the certificate $c_v^{X/Z}$. Fourth, Y checks whether its hash values of potential previous communication with X in present in X's SL (in our case Y has met X previously as shown in X's VSL). This step ensures that X has not deleted any previous hash values for communications with Y. Then ,Y checks whether its hash values of potential previous communication with those nodes which validated X's log (i.e., Y and Z in our case) are present in X's VHT. If Y noticed missing hash values with some node T, then Y adds T to it black list. This step ensures that other nodes are not using different log version for different nodes. As soon as this validation finishes, Y drops the entries $e_{v+1}^X, .., e_{n+1}^X$ and computes a certificate for X's $c_{n+1}^{X/Y}$ which is a signed message of the verified log hash $H(h_1^X, h_2^X, ..., h_{n+1}^X)$, and adds X's SL to its own VHT (it overrides the old one if exists). Now, Y prepares a validation acknowledgment message VACK similar to the Valid message in with entry $e_{m+2}^Y = (s_{m+2}, SEND, o_{m+2})$ (where $o_{m+2} = a_{n+1}^X$), adds it to its USL, and sends the ASK message back to X (the details concerning the VACK message are quite similar to step 1 above).

Proof: (i) Y will not drop the Valid message of X and must send the correct VACK message; (ii) Y will correctly validate X's log. Regarding (i), X might hold some message to Y, and thus Y should validate with X to start the communication. The rest of proving (i) is similar to step 1 above. As for (ii), first, Y must compute the hash values from h_v^X until h_{n+1}^X since it must create a certificate for X's $c_{n+1}^{X/Y}$ which is a signed message of the verified log hash $H(h_1^X, h_2^X, ..., h_{n+1}^X)$, and adds X's SL to its own VHT (recall that Y did not receive the hash values between h_v^X and h_{n+1}^X); and since X will verify their correctness in the next step, Y could not miss this step. Second, Y will check if possible old hash values are existing in X's log and the logs of other nodes in VHT (i.e., Y and Z in our case) to make sure that these nodes are not misbehaving by using different log versions for different nodes; otherwise, Y will add misbehaving nodes to it black list. Fourth, Y will add the verified log of X to its VHT since any Valid or VACK message in Y's log should correspond to an entry in VHT, and because this will be verified in future validation phases with any node.

3. **X receives VACK message from Y and verifies if Y has computed its log hash sequence correctly. The it checks if its previous hash values, if any, are present in Y's log or VHT; this is crucial to check for log fragmentation of nodes. Then, X starts the communication phase by**

sending an Anonym message to Y with a certificate of Y's log signed by X.

Additional details. X receives VACK from Y and adds a corresponding entry $e^X_{n+2} = (s_{n+2}, RECV, o_{n+2})$ to its USL log. Then, X makes sure that I has correctly validated it's log by matching the hash values received in the VHT of Y from h^X_v until h^X_{n+1}; otherwise, X adds Y to its blacklist. Then, X deletes all its entries $e^X_{v+1}, .., e^X_{n+1}$ (i.e., USL is now empty). After that, X validates Y's log in a similar way to step 2 above. Then, X trusts Y, and starts the communication phase by preparing an *Anonym* message (see Subsection 2.2), and adds the corresponding entry $e^X_{n+3} = (s_{n+3}, SEND, o_{m+3})$ it to its USL, and sends it along with the certificate $c^{Y/X}_{m+2}$, computed using VSL and USL of Y, which proves that X has correctly validated Y's log until the last hash value in Y's USL h^Y_{m+2}.

Proof:(i) X will not drop the VACK message of Y; (ii) X will match its hash values of USL against those received from Y. (iii) X will correctly validate Y's log. The proof of (i) is similar to that of step 1. As for (ii), X is afraid of saving a forged certificate for its log from Y that contains a hash H different to the correct value, which can be detected by other nodes in future validations (as done is step 2). Concerning (iii), since X will send the certificate $c^{Y/X}_{m+2}$ to Y in the Anonym message, then Y can detect, in a similar way, if X has computed the hash value of the certificate correctly (recall that X has not received all the hash values of Y's USL); Y will do this since it is afraid of saving a forged certificate from X too.

4. **Y receives Anonym message from X and verifies if Y has sent a correct certificate for Y's log.**

Additional details. Y receives the Anonym message from X and adds a corresponding entry $e^Y_{m+3} = (s_{m+3}, RECV, o_{m+3})$ it to its USL. Then Y checks whether X has verified correctly its log by checking if the certificate $c^{Y/X}_{m+2}$ sent by X matches the hash value of Y itself. If this is true, Y trusts X and deletes all the entries in its own USL until e^Y_{m+2}, and the communication phase continues; otherwise, Y adds X to its blacklist.

Proof: (i) Y wont drop the Anonym message and (ii) will verify if X has correctly validated its log. The proof of (i) is presented in Subsection 2.2. As for (ii), Y is afraid of adding a forged certificate from X, if this latter misbehaved, since other nodes will detect this in Y's log and will accuse Y (thinking that Y is tampering with its log).

TODO: What happens with Y's certificates if X refused to send the Anonym (it might do so if it is selfish and only wants to validate trying to minimize its storage by deleting log entries)?

TODO: What happens if we truncated the VSL periodically after a certain fixed number of validations (to save storage, time, and bandwidth)?

References

[1] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, UK, 2002. Springer-Verlag.

[2] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 175–188, New York, NY, USA, 2007. ACM.

[3] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *Proceedings of the 11th USENIX Security Symposium*, pages 297–312, Berkeley, CA, USA, 2002. USENIX Association.

[4] Alessandro Mei and Julinda Stefa. Give2get: Forwarding in social mobile wireless networks of selfish individuals. *IEEE Transactions on Dependable and Secure Computing*, 9(4):568–581, 2012.