Universally Composable Anonymous Broadcast Protocols

Kyla Nel
2576953n

Abstract

Anonymous broadcast functionality $\mathcal{F}_R^K$

Initialise:

1. a list of pending messages $L_{pend} \leftarrow []$
2. $status_P \in \{0, 1\} \leftarrow 0$ for party P indicating whether P has sent a message in the current round

■ Upon receiving (**sid**, **WRITE**, M) from honest party P or (**sid**, **WRITE**, M, P) from S on behalf of corrupted party P:
If $status_P = 0$, then

1. set $status_P \leftarrow 1$
2. append M to L_{pend}
3. if $|L_{pend}| = K$, then
   - (a) order the messages lexicographically as $< M_1, ..., M_K >$
   - (b) set $Lpend \leftarrow []$
   - (c) set $status_P \leftarrow 0$ for every P
   - (d) send (**sid**, **BROADCAST**, $< M_1, ...M_K >$ to all parties and (**sid**, **BROADCAST**, $< M_1, ...M_K >, P$) to S
4. else, send (**sid**, **WRITE**, $|M|$, P) to S

Figure 1: Anonymous broadcast ideal functionality.

Riposte UC Protocol

Variables:

- R - number of rows in each database table
- C - length of messages
- $e_{\ell,M}$ - R x C x 2 bitstring containing 0 everywhere except in row l which contains $(M, M^2) \in \mathbb{F}^k$, where M is the message to be sent
- K - message limit in a round

Initialise:

1. $status_P \in \{0, 1\} \leftarrow 0$ for party P indicating whether P has sent a message in the current round
2. $count \in \mathbb{N} \leftarrow 0$ indicating the number of valid write requests received this round

■ Upon receiving (**sid**, **WRITE**, M) from P
If $status_P = 0$, then

1. set $status_P \leftarrow 1$
2. P chooses index $l \xleftarrow{\$} [0, R)$ and generates bitstring e_l
3. generate random R x C x 2 bitstring r
4. send (**prove**, P, e_l) to $\mathcal{F}_{ZK}^{R,R'}$
5. send $r \oplus e_l$ to Server B using $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$
6. send (**prove**, P, $e_{\ell,M}$) to $\mathcal{F}_{ZK}^{R,R'}$
7.
8. send $r \oplus e_{\ell,M}$ to Server B using $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$
9. send r to Server A using $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$
10. $count$ += 1
11. if $count = K$, then
    - (a) set $status_p \leftarrow 0$
    - (b) set $count \leftarrow 0$

■ Upon receiving (**sid**, **BROADCAST**, M_A) from Server A and (**sid**, **BROADCAST**, M_B) from Server B

1. Verify that $M_A = M_B$
2. If $M_A = M_B$, forward to $\mathcal{Z}$

■ Upon receiving (**sid**, **SEND**, $r \oplus e_l$) from P, if P has not executed a write request in this phase, then
Server B executes the following:

1. Upon receiving (**proof**, l(y)) from $\mathcal{F}_{ZK}^{R,R'}$, if received (**sid**, WRITE, M) from P:
2. XOR $r \oplus e_{\ell,M}$ into its database
3. if $count = K$, then
4. (a) XOR $r \oplus e_l$ into its database
   - (b) if $count = K$, then
     - (i) combine database with Server A's database
     - (ii) check for collisions
     - (iii) resolve collisions
     - (iv) order messages lexicographically as $M_B =< M_1, ..., M_K >$
     - (v) broadcast messages to all parties
5. Upon receiving (**sid**, WRITE, M) from P, if received (**proof**, l(y)) from $\mathcal{F}_{ZK}^{R,R'}$:
   - (a) XOR $r \oplus e_l$ into its database
   - (b) if $count = K$, then
     - (i) combine database with Server A's database
     - (ii) check for collisions
     - (iii) resolve collisions
     - (iv) order messages lexicographically as $M_B =< M_1, ..., M_K >$
     - (v) broadcast messages to all parties

Figure 2: Anonymous broadcast protocol.

AE channel functionality $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$

Initialise a list $PendingMsg \leftarrow \emptyset$.
■ Upon receiving (**sid**, **SEND**, M) from P, if P is honest, then:

(1) If $\{A, B\} \setminus \{P\}$ is corrupted, then send (**sid**, **SEND**, M, P) to $\mathcal{S}$.
(2) If $\{A, B\} \setminus \{P\}$ is honest, then
    - Choose a random tag $\xleftarrow{\$} \{0, 1\}^{\lambda}$.
    - Add (**tag**, M, P) to $PendingMsg$
    - Send (**sid**, **SEND**, **tag**, $|M|$, P, {A,B} \ {P}) to $\mathcal{S}$.
(3) Upon receiving (**sid**, **ALLOW**, **tag**) from $\mathcal{S}$, if there is a (**tag**, M, P) in $PendingMsg$, then remove (**tag**, M, P) from $PendingMsg$ and send (**sid**,**SEND**,M) to {A,B}\{P}

Figure 3: Anonymous broadcast ideal functionality.

Zero-knowledge functionality $\mathcal{F}_{ZK}^{R,R'}$

(1) Wait for an input (**prove,** y**,** w) from P such that $(y, w) \in R$ if P is honest, or $y, w \in R'$ if P is corrupt. Send (**prove**, l(y)) to $\mathcal{A}$. Further, wait for a message **ready** from V, and send **ready** to $\mathcal{A}$.
(2) Wait for message **lock** from $\mathcal{A}$.
(3) Upon receiving a message **done** from $\mathcal{A}$, send **done** to P. Further, wait for an input **proof** from $\mathcal{A}$ and send (**proof**, y) to V.

Corruption rules:

- If P gets corrupted after sending (**prove**, y, w) and before Step 2, $\mathcal{A}$ is given (y, w) and is allowed to change this value to any value $(y', w') \in R'$ at any time before Step 2.

Figure 4: Zero-knowledge functionality $\mathcal{F}_{ZK}^{R,R'}$

Broadcast functionality $\mathcal{F}_{BC}$

(1) Wait for an input (**prove,** y**,** w) from P such that $(y, w) \in R$ if P is honest, or $y, w \in R'$ if P is corrupt. Send (**prove**, l(y)) to $\mathcal{A}$. Further, wait for a message **ready** from V, and send **ready** to $\mathcal{A}$.
(2) Wait for message **lock** from $\mathcal{A}$.
(3) Upon receiving a message **done** from $\mathcal{A}$, send **done** to P. Further, wait for an input **proof** from $\mathcal{A}$ and send (**proof**, y) to V.

Corruption rules:

- If P gets corrupted after sending (**prove**, y, w) and before Step 2, $\mathcal{A}$ is given (y, w) and is allowed to change this value to any value $(y', w') \in R'$ at any time before Step 2.

Figure 5: Zero-knowledge functionality $\mathcal{F}_{ZK}^{R,R'}$

1 Introduction

2 Proof

Cases:

(1) U.r. (**sid**, WRITE, $|M|$, P) from functionality:
    - Simulate a WRITE request on behalf of P where M_{Dummy} is all-zeroes
    - Generate $e_{\ell,M}$
    - $\mathcal{F}_{ZK}$ leaks nothing. $\mathcal{F}_{AEC}$ leaks the length of the message $|M|$, so the simulator sends $|M|$ to the adversary
(2) U.r. $< M_1, ..., M_k >$ from the functionality: If Server A is corrupted, then
    - Simulator sends a dummy message containing all zeroes over $\mathcal{F}_{AEC}$
    - Extract the subset of honest messages from the set of all messages.
    - Randomly assign honest messages to honest parties
    - Generate $e_{\ell,M}$ of the corresponding party and send r to Server A, then $e_{\ell,M} \oplus r$ is the share of party P for Server B

    If Server B is corrupted, then
    - Simulator equivocates by sending any r to Server A
    - Extract the subset of honest messages from the set of all messages.
    - Randomly assign honest messages to honest parties
    - Construct a consistent $e_{\ell,M}$
    - Send $e_{\ell,M} \oplus r$ of the corresponding party to Server B
(3) If an adversarial message is received, then
    - Simulator reconstructs M since it controls $\mathcal{F}_{AEC}$ and every P is forced to send messages across the authenticated encrypted channel because all adversaries are passive
    - Tell the functionality to add M to the list of pending messages

Case 1: We first deal with the case where each party P_i is honest. When Z provides input to P_i, S begins running π_R on behalf of the honest party P_i. The protocol execution is easily simulated by S. In the hybrid world, the authenticated encrypted channel leaks the length of messages sent through it. In the ideal world, S emulates this by leaking the length of messages sent through itself when it plays the role of $\mathcal{F}_{AEC}$.

Case 2: Next we deal with the case where Server A is passively corrupted. Here $\mathcal{A}$ requests to corrupt Server A in the hybrid world and so S corrupts Server A in the ideal world. Upon receiving (**sid**, WRITE, M_i) from honest party P_i, S generates e_{l,M_i} and random vector r. S then sends $r \oplus e_{l,M_i}$ to Server B and r to Server A on behalf of P_i. Upon receiving (**sid**, BROADCAST, $< M_1, ..., M_k$) from $\mathcal{F}_{BC}$, S extracts the subset of honest messages from $< M_1, ..., M_k >$ and randomly assigns the honest messages to honest parties. S orders these messages lexicographically and forwards them to Z.

Case 3: The next case to consider is the case where Server B is passively corrupted. Here $\mathcal{A}$ requests to corrupt Server B in the hybrid world and so S corrupts Server A in the ideal world. Upon receiving (**sid**, WRITE, M) from honest party P_i, S equivocates by sending any random vector r to Server A. S then generates e_{l,M_i} and sends $r \oplus e_{l,M_i}$ to Server B. When the two servers combine their shares, A can send any share g instead of e_{l,M_i} on behalf of Server B. However, when shares are combined, $r \oplus e_{l,M_i} \oplus r$ and $g \oplus r$ share the same distribution and are thus indistinguishable. Upon receiving (**sid**, BROADCAST, $< M_1, ..., M_k$) from $\mathcal{F}_{BC}$, S extracts the subset of honest messages from $< M_1, ..., M_k >$ and randomly assigns the honest messages to honest parties. S orders these messages lexicographically and forwards them to Z.

Case 4: Lastly we consider the case where a party P^* is corrupted. A requests to corrupt P^* in the hybrid world so S corrupts P^* in the ideal world. A sends (**sid**, WRITE, M^*) on behalf of P^*. A then continues to execute the protocol, creating shares and sending them to Server A and Server B. Since these shares are sent over $\mathcal{F}_{AEC}$, S can view these as it emulates the functionality. This allows S to reconstruct M^*. S can then signal its own functionality $\mathcal{F}_R^K$ to add M^* to the list of pending messages.

3 Background

Perhaps you want to cite the seminal paper of Turing [3], or prior [2] and concurrent [1] work.

4 My Amazing System

5 Evaluation

5.1 Experimental Setup

5.2 Experimental Analysis

6 Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Acknowledgments

I would like to thank ...

References

[1] Alonzo Church. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics* 58, 2 (1936), 345–363. http://www.jstor.org/stable/2371045

[2] Kurt Gödel. 1931. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik* 38–38, 1 (Dec. 1931), 173—-198. doi:10.1007/bf01700692

[3] Alan M. Turing. 1937. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42, 1 (1937), 230–265. doi:10.1112/plms/s2-42.1.230