Universally Composable Anonymous Broadcast Protocols

Clever Student
1234567a

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, $e_{\ell,M}$) to $\mathcal{F}_{ZK}^{R,R'}$
(5)
(6) send $r \oplus e_{\ell,M}$ to Server B using $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$
(7) send r to Server A using $\mathcal{F}_{\mathcal{AEC}}(\{A, B\})$
(8) $count$ += 1
(9) 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) XOR $r \oplus e_{\ell,M}$ into its database
(2) if $count = K$, then
  (a) combine database with Server A's database
  (b) check for collisions
  (c) resolve collisions
  (d) order messages lexicographically as $M_B =< M_1, ..., M_K >$
  (e) broadcast messages to all parties

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

(1) XOR r into its database
(2) if $count = K$, then
  (a) combine database with Server B's database
  (b) check for collisions
  (c) resolve collisions
  (d) order messages lexicographically as $M_A =< M_1, ..., M_K >$
  (e) 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 $\{A, B\} \backslash \{P\}$ 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 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 $\mathcal{F}_{AEC}$
   - 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
   - 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 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

	machine A	machine B
CPU	Intel Core i7-9700 CPU	2x Intel Xeon E5-2630 v3
CPU Frequency	3.00GHz	2.40GHz
RAM	16GB DDR4	128GB
OS	Ubuntu 20.04 LTS	Ubuntu 16.04 LTS
Compiler	GCC 9.3	GCC 7.3
libm	v2.31	v2.23
libomp	v4.5	v4.5

Table 1: This is the table caption.

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