Multiparty Battleship Game

Communications Security 2024/2025

Ramiro Moldes
ist199313

João Gonçalves
ist199995

Teresa Nogueira
ist1100029

Group 4B

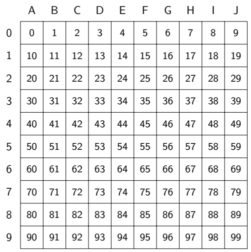
Instituto Superior Técnico

Master in Electrical and Computer Engineering

Presentation Outline

Project Overview

Project Overview



	A	B	C	D	E	F	G	H	I	J
0	0	1	2	3	4	5	6	7	8	9
1	10	11	12	13	14	15	16	17	18	19
2	20	21	22	23	24	25	26	27	28	29
3	30	31	32	33	34	35	36	37	38	39
4	40	41	42	43	44	45	46	47	48	49
5	50	51	52	53	54	55	56	57	58	59
6	60	61	62	63	64	65	66	67	68	69
7	70	71	72	73	74	75	76	77	78	79
8	80	81	82	83	84	85	86	87	88	89
9	90	91	92	93	94	95	96	97	98	99

Figure: 10×10 Battleship board.

- The game is driven by seven actions: `Create`, `Join`, `Fire`, `Report`, `Win`, `Wave` and `Contest`
- Each corresponds to a specific transition (or lack of) in the game's state.

Registered on a blockchain "emulator"

(akin to a bulletin board)

Core Implementation

Core Implementation

The game is setup in three phases:

- **Commitment Phase:** Players commit their fleet layout without revealing it: `Join` and `Create`.
- **Commitment Update Phase:** Players take turns firing at each other and updating their board commitments accordingly: `Fire` and `Report`.
- **Announcement Phase:** A player declares victory, and others may contest it: `Win` and `Contest`.

Commitement Phase

A commitment scheme allows one party to bind themselves to a particular set of data without revealing that data:

- The commitment is implemented as a hash of the player's fleet layout combined with a secret nonce.
- The player runs a **Zero-Knowledge Virtual Machine** program which enforces the fleet layout to be valid.

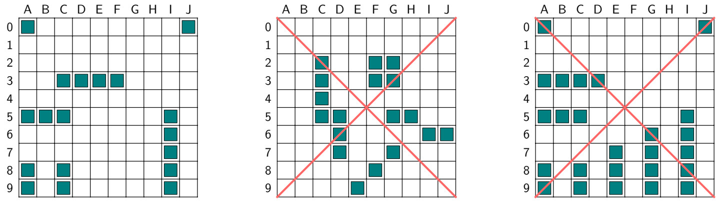
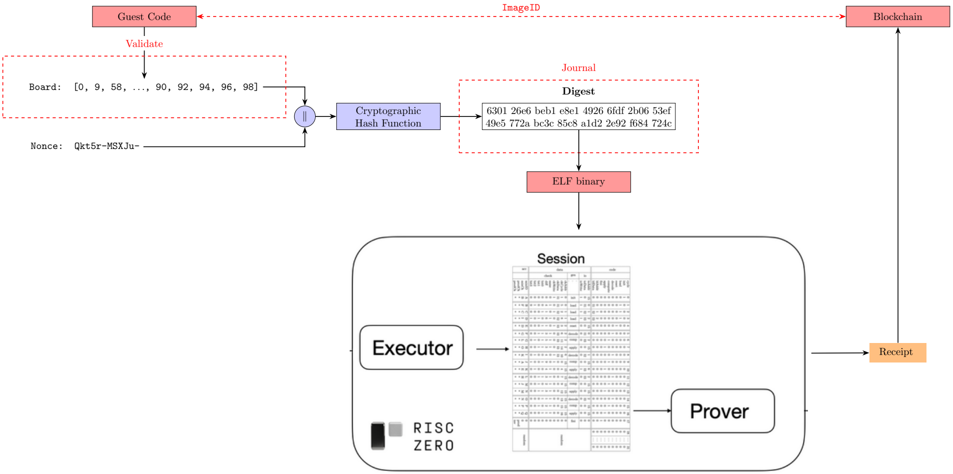


Figure: A set of battlefield grids exemplifying the rules of the game.

Commitement Phase

zkVM application

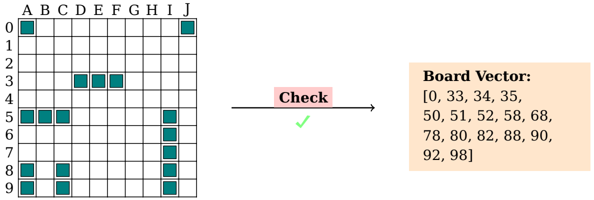


Guest Code
ImageID
Blockchain
Validate
Board: [0, 9, 58, ..., 90, 92, 94, 96, 98]
Nonce: Qkt5r-MSXJu-
||
Cryptographic Hash Function
Journal
Digest
6301 26e6 beb1 e8e1 4926 6fdf 2b06 53ef
49e5 772a bc3c 85c8 a1d2 2e92 f684 724c
ELF binary
Session
Executor
Prover
RISC ZERO
Receipt

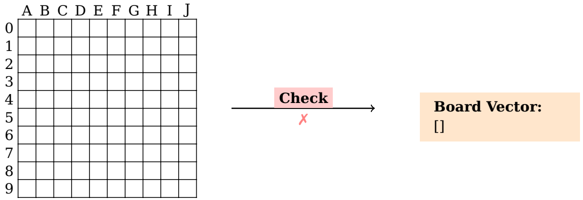
Commitment Update Phase

`Fire` game action - Host's Perspective

The **fire game action** require proof that the fleet is not sunk.



Board Vector:
[0, 33, 34, 35,
50, 51, 52, 58, 68,
78, 80, 82, 88, 90,
92, 98]



Board Vector:
[]

Commitment Update Phase

Fire game action - Blockchain's Perspective

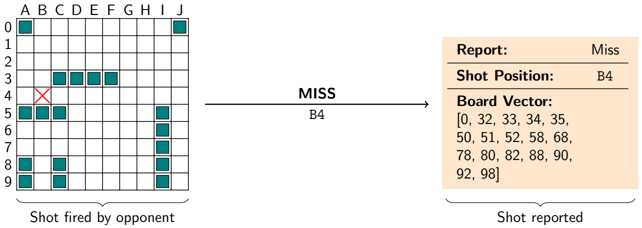
We compare the **committed board digest** stored on-chain with the one **submitted in the journal** during the fire game action:

```
if data.board != player.current_state {
    return format!("Fleet commitment does not
                              match recorded state\n");
}
```

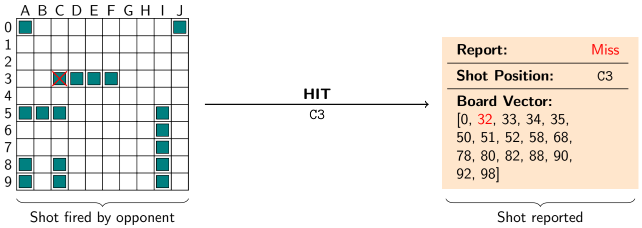
Listing: Validation of commitment hash on the blockchain's side

Commitment Update Phase

`Report` game action - Host's Perspective



Report:	Miss
Shot Position:	`B4`
Board Vector: [0, 32, 33, 34, 35, 50, 51, 52, 58, 68, 78, 80, 82, 88, 90, 92, 98]	

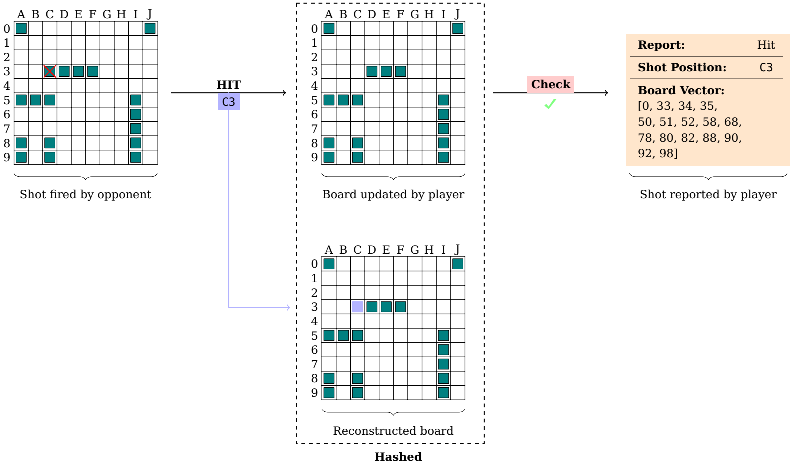


Report:	Miss
Shot Position:	`C3`
Board Vector: [0, 32, 33, 34, 35, 50, 51, 52, 58, 68, 78, 80, 82, 88, 90, 92, 98]	

Figure: Correct Miss report (above) and incorrect Miss report (below).

Commitment Update Phase

`Report` game action - Host's Perspective



Report:	Hit
Shot Position:	C3
Board Vector: [0, 33, 34, 35, 50, 51, 52, 58, 68, 78, 80, 82, 88, 90, 92, 98]	

Figure: Correct Hit report.

Commitment Update Phase

`Report` game action - Blockchain's Perspective

Again, validates that the **board digests** are consistent:

```
if data.board != player.current_state {
    return format!("Reported board does not match
                                                stored commitment");
}
```

Listing: Validation of the commitment.

Makes sure the **shot advertised** by the player **is correct**.

Update game state with new digest submitted in the journal:

```
player.current_state = data.next_board.clone();
```

Listing: Updating the stored board state.

Announcement Phase

`Win` and `Contest` game actions - Host's Perspective

- Both can be executed "out of turn."
- Fleets must not be fully sunk for either action!

```
assert!(
    !input.board.is_empty(),
    "Your fleet is fully sunk..."
);
```

Listing: Asserts that the board vector isn't empty.

Announcement Phase

`Win` and `Contest` game actions - Blockchain's Perspective

Both handlers validate consistency of **board digests** (stored and committed).

- Successful win claim **generates a note**: *who*, *when*, and *what* the final committed board looked like.

```
pub struct PendingWin {
    pub claimant: String, // Fleet ID that claimed win
    pub board: Digest,    // Committed board hash
    pub time: Instant,    // Time when claim was made
}
```

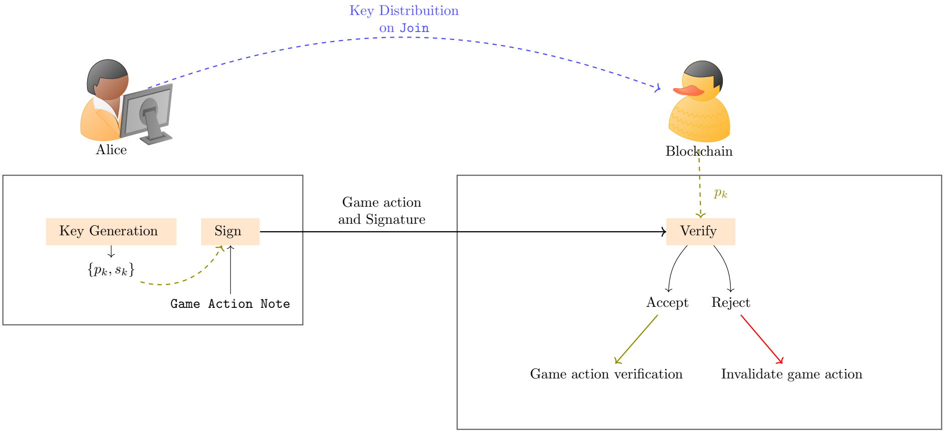
Listing: Structure of note created in a game after a win claim.

- Successful contestation **wipes** `PendingWin` note!
- Win subject to **peer validation**.

Extra Functionalities

Digital Signatures

In Star Trek, *Dilithium* is a rare material that cannot be replicated. Similarly, signatures cannot be replicated or forged: **each one is unique**.

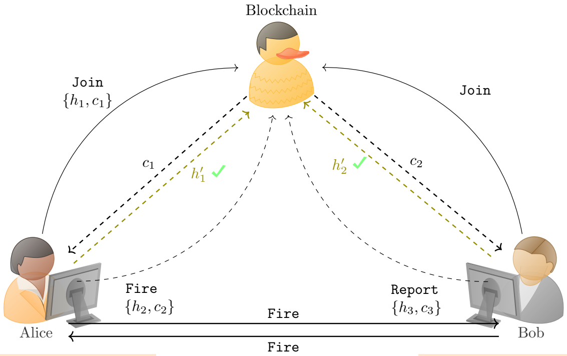


Token-based Turn Management

Game:	Xpto
Players:	{Alice, Bob}
Next Turn:	$\{h_1, c_1\}$ (Alice)

$\longrightarrow$

Game:	Xpto
Players:	{Alice, Bob}
Next Turn:	$\{h_2, c_2\}$ (Bob)



$r_1' :=$ `RSADecryption`$(s_k^{\text{Alice}}, c_1)$

$h_1' :=$ `SHA256`(r_1');

$\downarrow$

$c_2 :=$ `RSAEncryption`(p_k^{Bob}, r_2)

$h_2 :=$ `SHA256`(r_2);

$r' :=$ `RSADecryption`(s_k^{Bob}, c)

$h' :=$ `SHA256`(r');

$\downarrow$

$c_3 :=$ `RSAEncryption`$(p_k^{\text{Alice}}, r_3)$

$h_3 :=$ `SHA256`(r_3);

Token-based Turn Management

```
let token_hash = Sha256::digest(&token);
let enc_token = rsa_pubkey
    .encrypt(&mut OsRng, Pkcs1v15Encrypt, &token).ok()?;
```

Listing: Player with turn creates new token, hashes and encrypts it.

⬇

```
let dec_token = rsa_privkey
    .decrypt(Pkcs1v15Encrypt, &enc_token)
    .map_err(|_| "Decryption failed, not player’s turn?")?;
```

Listing: Target player decrypts the token and passes it to the zkVM.

⬇

Continues…