INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Blockchain Technology and Its Application to IoT-Enabled Internet of Drones for Secure Data Delivery and Collection

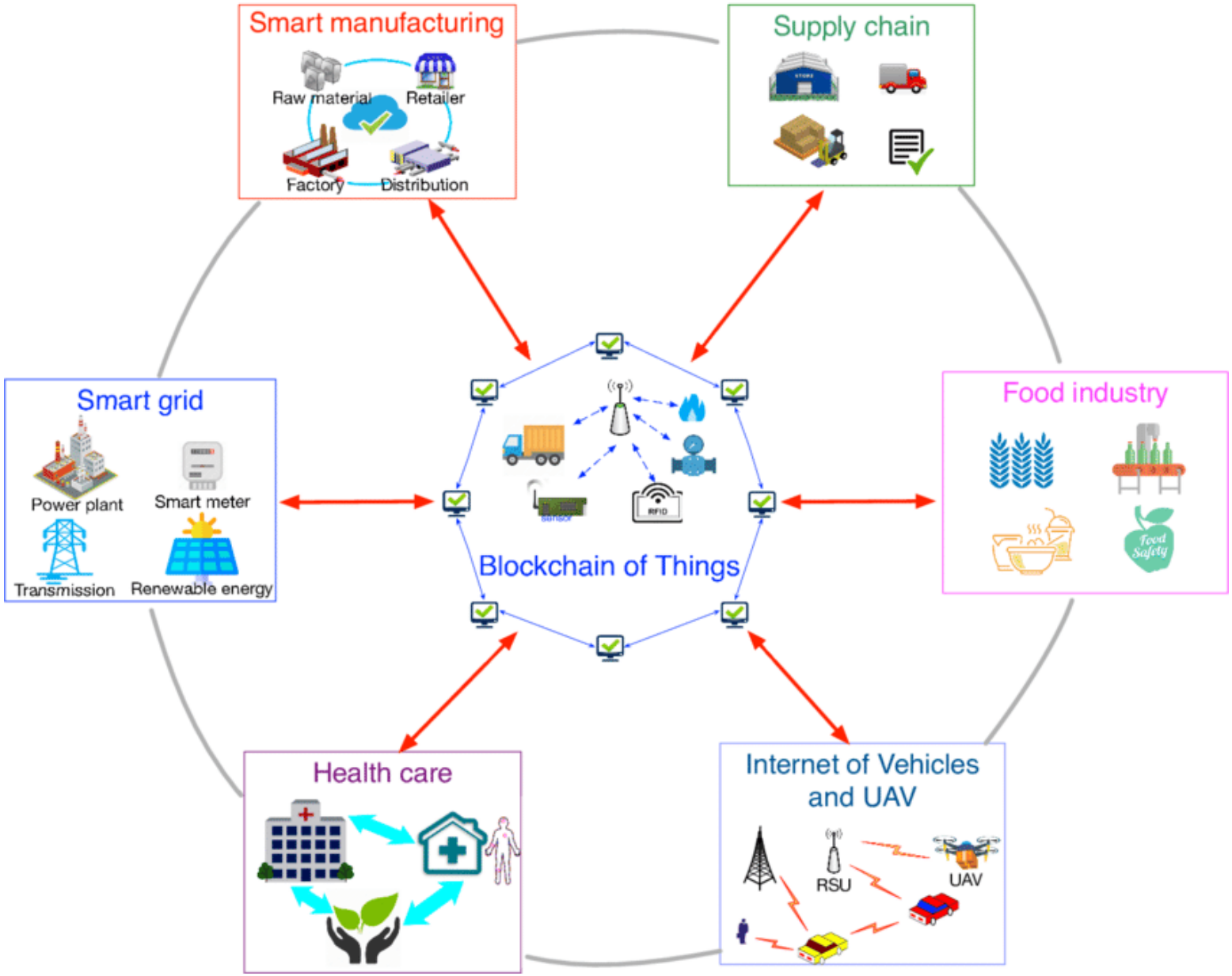
Dr. Ashok Kumar Das

IEEE Senior Member
Associate Professor
Center for Security, Theory and Algorithmic Research
International Institute of Information Technology, Hyderabad

E-mail: *ashok.das@iiit.ac.in*
URL: `http://www.iiit.ac.in/people/faculty/ashokkdas`
`https://sites.google.com/view/iitkgpakdas/`

October 26, 2022

Applications of Blockchain Technology



Important References: Blockchain Applications

- Basudeb Bera, Sourav Saha, Ashok Kumar Das, Neeraj Kumar, Pascal Lorenz, and Mamoun Alazab. "Blockchain-Envisioned Secure Data Delivery and Collection Scheme for 5G-Based IoT-Enabled Internet of Drones Environment," in ***IEEE Transactions on Vehicular Technology***, Vol. 69, No. 8, pp. 9097-9111, 2020, DOI: 10.1109/TVT.2020.3000576.
- Jangirala Srinivas, Ashok Kumar Das, and Athanasios V. Vasilakos. "Designing Secure Lightweight Blockchain-Enabled RFID-Based Authentication Protocol for Supply Chains in 5G Mobile Edge Computing Environment," in ***IEEE Transactions on Industrial Informatics***, Vol. 16, No. 11, pp. 7081-7093, 2020, DOI: 10.1109/TII.2019.2942389.
- Sourav Saha, Durbadal Chattaraj, Basudeb Bera, and Ashok Kumar Das. "Consortium blockchain-enabled access control mechanism in edge computing based generic IoT environment," in ***Transactions on Emerging Telecommunications Technologies (Wiley)***, 2020, DOI: 10.1002/ETT.3995.
- Basudeb Bera, Durbadal Chattaraj, and Ashok Kumar Das. "Designing secure blockchain-based access control scheme in IoT-enabled Internet of Drones deployment," in ***Computer Communications (Elsevier)***, Vol. 153, pp. 229-249, 2020, DOI: 10.1016/j.comcom.2020.02.011

Case Study: Blockchain-Envisioned Secure Data Delivery and Collection Scheme for 5G-Based IoT-Enabled Internet of Drones Environment

Basudeb Bera, Sourav Saha, **Ashok Kumar Das**, Neeraj Kumar, Pascal Lorenz, and Mamoun Alazab. "Blockchain-Envisioned Secure Data Delivery and Collection Scheme for 5G-Based IoT-Enabled Internet of Drones Environment," in ***IEEE Transactions on Vehicular Technology***, Vol. 69, No. 8, pp. 9097-9111, 2020, DOI: 10.1109/TVT.2020.3000576. (2020 SCI Impact Factor: 5.978)

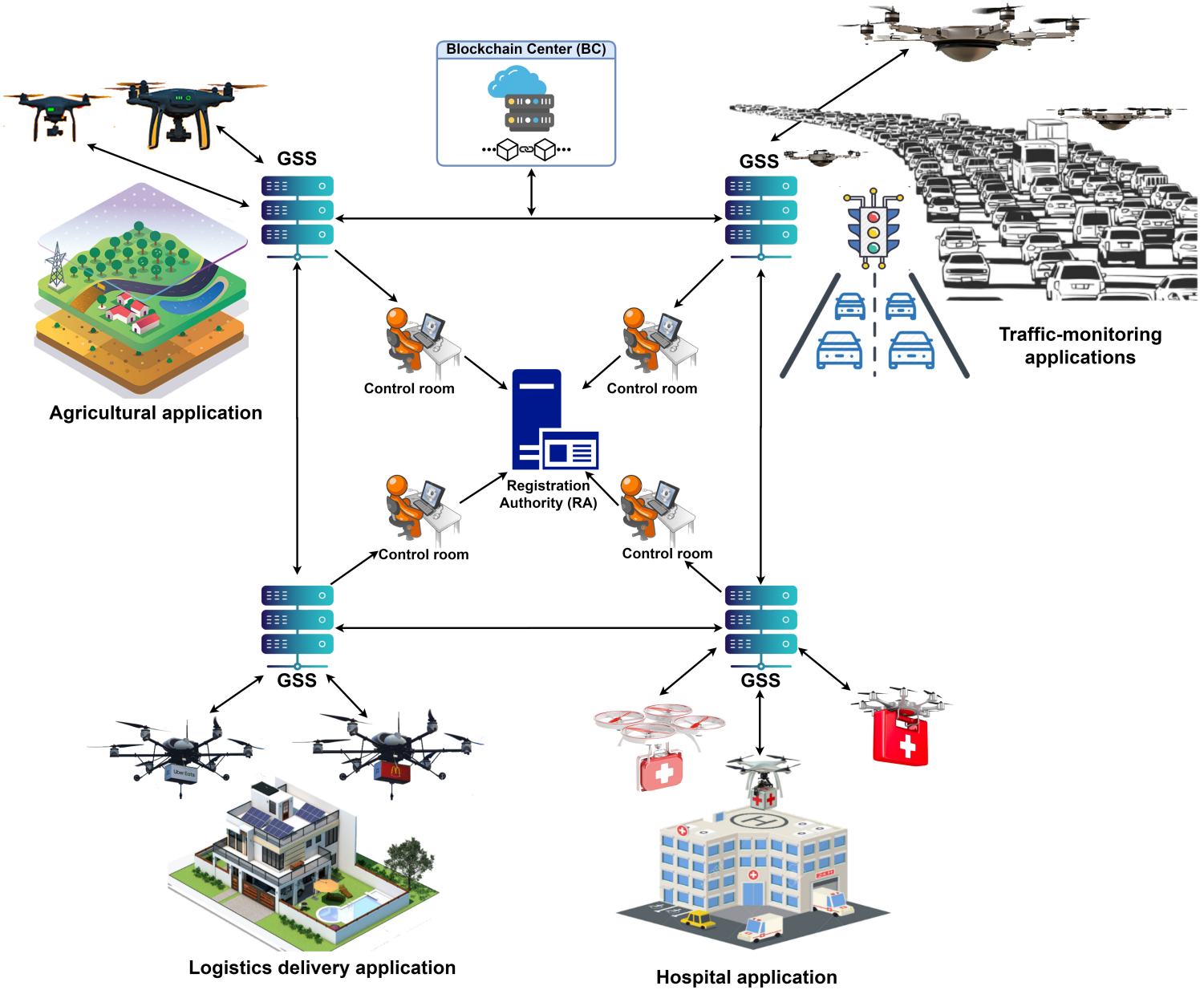
Internet of Drones

- The Internet of Drones (IoD) is a “layered network control architecture”.
- Primarily designed to co-relate access of Unmanned Aerial Vehicles (UAVs) (which are also called *drones*) for controlling airspace and providing support to various navigation activities [[6]].
- Due to increase of commercial drones applications, recent forecasts indicate that there will be a 100 USD billion market opportunity over the coming years based on the drones ([1]).
- Several applications of drones: ranging from military, newsgathering (for example, videography and photography), security, agricultural and logistics deployments, surveillance, medicine to traffic-monitoring applications ([8], [5]).

Secure Data Delivery and Collection: Need of the Hour

- In recent years, a drone delivery service has been established in London due to people demands, which can permit to exchange packages weighing up to 500g. In addition, Germany's express delivery company, Deutsche Post (DHL) also use the drones, called "parcelcopters" for the emergency delivery (for instance, high-priority goods like medicines to remote areas).
- A delivery system in the IoD environment was built upon the trust among the delivery service provider(s) and their customers ([7]). A service provider believes that their customers will not repudiate the delivery confirmation after successfully delivered the items in the proper destination. The customers need to also trust that the service provider will not deliver an item without presence of them or delivered an item in front of the door is stolen.
- In such kind of cases, it is very hard to maintain the responsibility. Due to such reasons, security plays a very important role in the IoD environment. This leads to design a secure data delivery and collection mechanism with the help of deployed drones.
- However, we need to maintain several security requirements, such as confidentiality, authentication, access control, non-repudiation, availability, freshness, etc. Apart from these requirements, we also face various security challenges in an IoD environment including "remote hijacking of drone", "privacy", "replay and man-in-the middle attacks", "impersonation attack", "privileged-insider attack", "physical drone capture attack", etc. as in other networks ([4, 3]).

Blockchain-envisioned 5G-enabled IoD environment

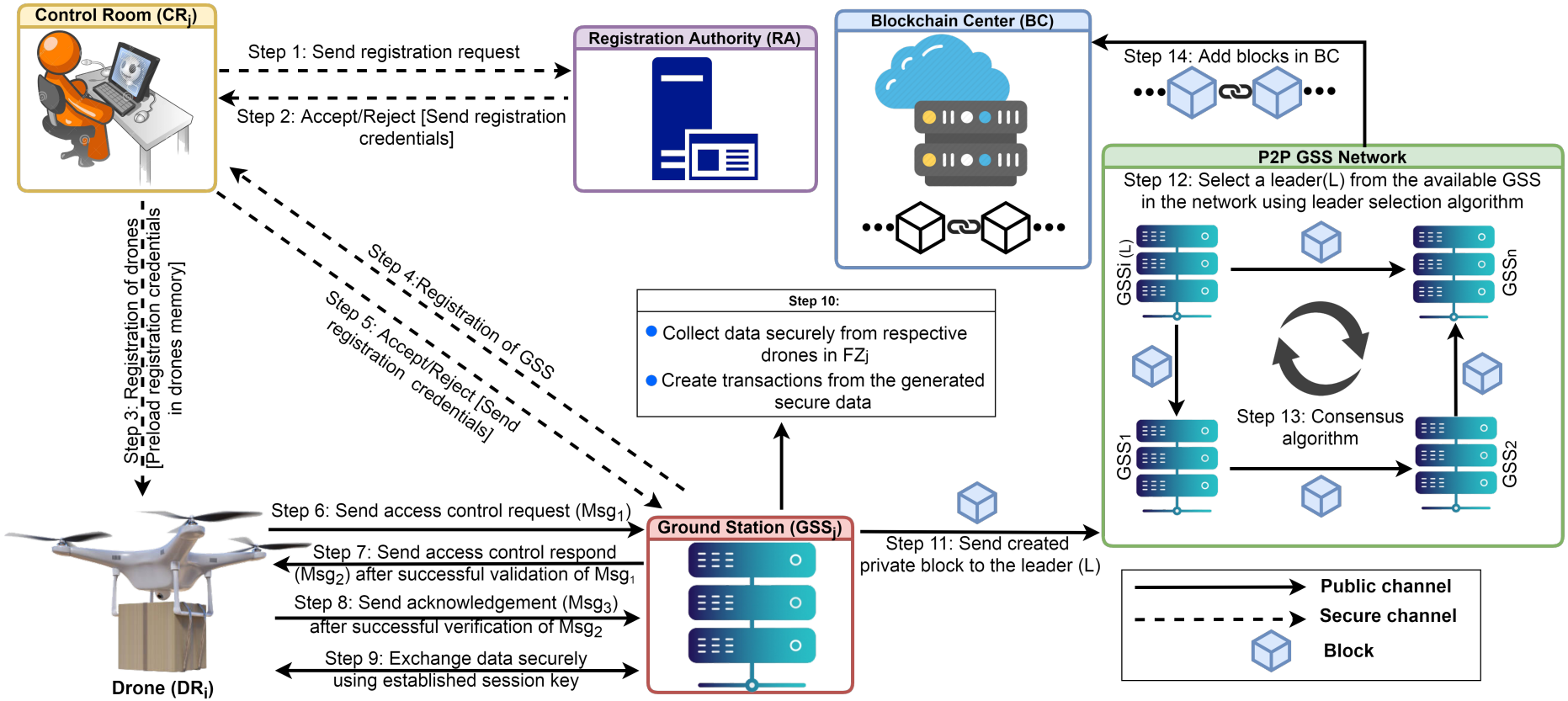


BSD2C-IoD: Blockchain-Based Secure Data Delivery and Collection Scheme

Different Phases:

- system initialization phase
- registration phase
- access control phase
- secure data delivery and collection phase
- block creation, verification and addition in blockchain center phase
- dynamic drones addition phase

Overall process in the proposed BSD2C-IoD



Registration phase for control room, ground station server and drones

(a) Registration of a controller room (CR_j) for an IoT application

Registration Authority (RA)	Control room (CR_j)
• Select $E_p(u,v)$ over $GF(p)$ with base point G, $h(\cdot)$ • Select random private key $r_{CR_j} \in Z_p^*$ and compute $Pub_{CR_j} = r_{CR_j} \cdot G$ for CR_j • Pick its own master (private) key $r_{RA} \in Z_p^*$ and compute public key $Pub_{RA} = r_{RA} \cdot G$ • Select identity ID_{RA}, and identity ID_{CR_j} for each CR_j • Create certificate for each CR_j as $Cert_{CR_j} = r_{CR_j} + h(ID_{CR_j} \| ID_{RA} \| Pub_{RA} \| Pub_{CR_j} \| RTS_{CR_j}) * r_{RA} \pmod p$ • Publish $\{Pub_{RA}, Pub_{CR_j}, E_p(u,v), h(\cdot), G\}$ as public	• Store in each CR_j: $\{ID_{CR_j}, ID_{RA}, Cert_{CR_j}, Pub_{RA}, Pub_{CR_j}, E_p(u,v), h(\cdot), G\}$ • Pick random master key $mk_{CR_j} \in Z_p^*$ and compute public key $Pk_{CR_j} = mk_{CR_j} \cdot G$ • Store $\{mk_{CR_j}, Pk_{CR_j}\}$ in its database and make Pk_{CR_j} as public

(b) Registration of Ground Station Server GSS_j

Control Room CR_j	Ground Station Server GSS_j
• Pick identity ID_{GSS_j} and compute its pseudo-identity $RID_{GSS_j} = h(ID_{GSS_j} \| RTS_{GSS_j} \| mk_{CR_j})$ • Pick random private key $r_{GSS_j} \in Z_p^*$ and compute public key $Pub_{GSS_j} = r_{GSS_j} \cdot G$ • Create certificate for GSS_j as $Cert_{GSS_j} = r_{GSS_j} + h(RID_{GSS_j} \| ID_{CR_j} \| Pub_{CR_j} \| Pub_{GSS_j}) * mk_{CR_j} \pmod p$ • Store RID_{GSS_j} and $Cert_{GSS_j}$ in GSS_j • Make Pub_{GSS_j} as public; and delete ID_{GSS_j} and r_{GSS_j} • Store $\{Pk_{GSS_j}\}$ for each GSS_j in its database	• Select another private key (decryption key) $k_{GSS_j} \in Z_p^*$ and compute public key (encryption key) $Pk_{GSS_j} = k_{GSS_j} \cdot G$ • Pre-load $\{RID_{GSS_j}, ID_{CR_j}, Cert_{GSS_j}, Pub_{CR_j}, Pub_{GSS_j}, (k_{GSS_j}, Pk_{GSS_j}), Pk_{CR_j}, E_p(u,v), h(\cdot), G\}$ in GSS_j

(c) Registration of drones DR_i in a flying zone FZ_j

Control Room CR_j	Drone DR_i
• Pick identity ID_{DR_i} and pseudo-identity $RID_{DR_i} = h(ID_{DR_i} \| ID_{CR_j} \| mk_{CR_j} \| RTS_{DR_i})$ for each drone DR_i • Select private certificate key $r_{DR_i} \in Z_p^*$ and compute public key $Pub_{DR_i} = r_{DR_i} \cdot G$ for each drone DR_i • Select private signature key $k_{DR_i} \in Z_p^*$ and compute public signature key $Pk_{DR_i} = k_{DR_i} \cdot G$ for each drone DR_i • Generate certificate for each DR_i as $Cert_{DR_i} = r_{DR_i} + h(RID_{DR_i} \| Pub_{CR_j} \| Pub_{GSS_j} \| Pub_{DR_i}) * mk_{CR_j} \pmod p$ • Delete ID_{DR_i} and r_{DR_i} from its database	• Store $\{RID_{DR_i}, Cert_{DR_i}, (k_{DR_i}, Pk_{DR_i}), Pk_{CR_j}, E_p(u,v), h(\cdot), G\}$ in DR_i prior to deployment in flying zone FZ_j

Access control phase

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

Access control between a ground station server GSS_j and its drones (DR_i) in a flying zone FZ_j

Drone (DR_i)	Ground Station Server (GSS_j)
available information: $\{RID_{DR_i}, Cert_{DR_i}, (k_{DR_i}, Pk_{DR_i}), Pk_{CR_j}, E_p(u,v), h(\cdot), G\}$	available information: $\{RID_{GSS_j}, ID_{CR_j}, Cert_{GSS_j}, Pub_{CR_j}, Pub_{GSS_j}, (k_{GSS_j}, Pk_{GSS_j}), Pk_{CR_j}, E_p(u,v), h(\cdot), G\}$
• Select random secret $r_1 \in Z_p^*$ & generate current timestamp TS_1 • Compute $r_1' = h(RID_{DR_i} \|\| r_1 \|\| Cert_{DR_i} \|\| k_{DR_i} \|\| TS_1)$, $A_{DR_i} = r_1' \cdot G$ • Generate signature Sig_{DR_i}	
$\xrightarrow{Msg_1 = \{RID_{DR_i}, A_{DR_i}, Cert_{DR_i}, Sig_{DR_i}, TS_1\}}$ (via public channel)	
	• Verify timestamp TS_1, and certificate by $Cert_{DR_i} \cdot G = Pub_{DR_i} + h(RID_{DR_i} \|\| Pub_{CR_j} \|\| Pub_{GSS_j} \|\| Pub_{DR_i}) \cdot Pk_{CR_j}$. • If timestamp and certificate are valid, verify signature by $Sig_{DR_i} \cdot G = A_{DR_i} + h(Pk_{DR_i} \|\| RID_{DR_i} \|\| Pk_{CR_j} \|\| Pub_{GSS_j} \|\| A_{DR_i} \|\| TS_1) \cdot Pk_{DR_i}$ • Generate random number $r_2 \in Z_p^*$ and current timestamp TS_2 • Calculate $r_2' = h(RID_{GSS_j} \|\| ID_{CR_j} \|\| r_2 \|\| Cert_{GSS_j} \|\| k_{GSS_j} \|\| TS_2)$, $B_{GSS_j} = r_2' \cdot G$, $DHK_{GSS_j,DR_i} = r_2' \cdot A_{DR_i} (= (r_2' * r_1') \cdot G)$, session key as $SK_{GSS_j,DR_i} = h(DHK_{GSS_j,DR_i} \|\| RID_{DR_i} \|\| RID_{GSS_j} \|\| Pk_{DR_i} \|\| Pub_{GSS_j})$, and session key verifier $SKV_{GSS_j,DR_i} = h(SK_{GSS_j,DR_i} \|\| RID_{DR_i} \|\| RID_{GSS_j} \|\| B_{GSS_j} \|\| Cert_{GSS_j} \|\| TS_1 \|\| TS_2)$
	$\xleftarrow{Msg_2 = \{RID_{GSS_j}, Cert_{GSS_j}, B_{GSS_j}, SKV_{GSS_j,DR_i}, TS_2\}}$ (via public channel)
• Check validity of TS_2, and certificate as $Cert_{GSS_j} \cdot G = Pub_{GSS_j} + h(RID_{GSS_j} \|\| ID_{CR_j} \|\| Pub_{CR_j} \|\| Pub_{GSS_j}) \cdot Pk_{CR_j}$ • If timestamp and certificate are valid, compute $DHK_{DR_i,GSS_j} = r_1' \cdot B_{GSS_j} (= (r_1' * r_2') \cdot G = DHK_{GSS_j,DR_i})$ and session key as $SK_{DR_i,GSS_j} = h(DHK_{DR_i,GSS_j} \|\| RID_{DR_i} \|\| RID_{GSS_j} \|\| Pk_{DR_i} \|\| Pub_{GSS_j})$ and session key verifier, $SKV_{DR_i,GSS_j} = h(SK_{DR_i,GSS_j} \|\| RID_{DR_i} \|\| RID_{GSS_j} \|\| B_{GSS_j} \|\| Cert_{GSS_j} \|\| TS_1 \|\| TS_2)$ • Check if $SKV_{DR_i,GSS_j} = SKV_{GSS_j,DR_i}$. If so, generate current timestamp TS_3 and compute $ACK_{DR_i,GSS_j} = h(SK_{DR_i,GSS_j} \|\| TS_2 \|\| TS_3)$	
$\xrightarrow{Msg_3 = \{ACK_{DR_i,GSS_j}, TS_3\}}$ (via public channel)	
	• Verify the received timestamp. If timestamp is valid, compute $ACK_{GSS_j,DR_i} = h(SK_{GSS_j,DR_i} \|\| TS_2 \|\| TS_3)$ • Verify if $ACK_{GSS_j,DR_i} = ACK_{DR_i,GSS_j}$. If valid, session key is established

Both DR_i and GSS_j establish the same session key $SK_{DR_i,GSS_j} (= SK_{GSS_j,DR_i})$

Block Creation, Verification and Addition in Blockchain Center Phase

Details of various transactions Tx_i

Type of Transaction	Description
$Tx_{CR-GSS-req}$	It represents a transaction between a control room (CR_j) and its GSS_j. It is the data delivery request from CR_j to GSS_j.
$Tx_{GSS-DR-req}$	It means a transaction between GSS_j and its drones DR_i. It denotes the data delivery request from GSS_j to DR_i in a particular flying zone FZ_j.
$Tx_{DR-GSS-res}$	It denotes a transaction between DR_i and its GSS_j. It is the data delivery/collection response from DR_i to GSS_j.
$Tx_{DR-GSS-data}$	It means a transaction between GSS_j and its drones DR_i. It represents the collection message from GSS_j to DR_i in a particular flying zone FZ_j.

Structure of a block $Block_i$ based on transactions

Block Header	
Block Version	$BVer$
Previous Block Hash	PBH
Merkle Tree Root	MTR
Timestamp	TS
Creator of Block	CB_{ID} (Identity of one of the GSSs, say GSS_j in P2P GSS network)
Public key of Signer GSS_j	Pk_{GSS_j}
Block Payload (Encrypted Transactions)	
List of t_n Encrypted Transactions $\#i$ (Tx_i)	$\{E_{Pk_{GSS_j}}(Tx_i) \mid i = 1, 2, \cdots, t_n\}$
Current Block Hash	$CBHash$
Signature on $CBHash$	$BSign = ECDSA.Sig_{k_{GSS}}(CBHash)$, where $ECDSA.Sig(\cdot)$ and $ECDSA.Ver(\cdot)$ represent ECDSA signature generation and verification algorithms, respectively

Consensus for block verification and addition in blockchain

Algorithm 1 Consensus for block verification and addition in blockchain

Input: A block $Block_i$ as shown in Fig.7 and $n_{f_{GSS}}$: the number of faulty GSS nodes in the P2P GSS network
Output: Commitment and addition of block $Block_i$ into blockchain after its successful validation

1: Assume L, say GSS_l, is selected as the leader and it wants to add $Block_i$ into blockchain
2: L generates current timestamp TS_{GSS_j} for each follower ground station server node GSS_j and performs voting process
3: L encrypts voting request $VotReq$ as $E_{Pk_{GSS_j}}(VotReq, TS_{GSS_j})$ and sends a message containing the same block, and $E_{Pk_{GSS_j}}(VotReq, TS_{GSS_j})$ to each follower node GSS_j, $(j = 1, 2, \cdots, n_{GSS}, \forall j \neq l)$, where $E(\cdot)$ and $D(\cdot)$ are the encryption and decryption functions, respectively
4: Assume the message from L is received by each follower GSS_j in the P2P GSS network at time $TS^*_{GSS_j}$
5: **for** each follower node GSS_j **do**
6: Decrypt the message as $(VotReq', TS_{GSS_j}) = D_{k_{GSS_j}}[E_{Pk_{GSS_j}}(VotReq, TS_{GSS_j})]$
7: Verify timestamp, Merkle tree root, current block hash, and signature on received block $Block_i$
8: If all checks are verified successfully, send the voting reply $VotRep$ and block verification status $BVStatus$ as $\{E_{Pk_L}(VotReq', VotRep, BVStatus)\}$ to L
9: **end for**
10: If $VCnt$ denotes the vote counter, initialize $VCnt \leftarrow 0$
11: **for** each received response message $\{E_{Pk_L}(VotReq', VotRep, BVStatus)\}$ from the responded follower GSS_j **do**
12: Compute $(VotReq', VotRep, BVStatus) = D_{k_L}[E_{Pk_L}(VotReq', VotRep, BVStatus)]$
13: **if** $((VotReq' = VotReq)$ and $((VotRep = valid)$ and $(BVStatus = valid)))$ **then**
14: Set $VCnt = VCnt + 1$
15: **end if**
16: **end for**
17: **if** $(VCnt > 2.n_{f_{GSS}} + 1)$ **then**
18: Send the commit response to all follower nodes
19: Add block $Block_i$ to the blockchain
20: **end if**

Security Analysis

- Replay Attack
- Man-in-the-Middle (MiTM) Attack
- Drone/GSS Impersonation Attacks
- Privileged-Insider Attack
- Physical Drone Capture Attack
- Ephemeral Secret Leakage (ESL) Attack

Formal Security Security Verification using Automated Validation of Internet Security Protocols and Applications (AVISPA) tool

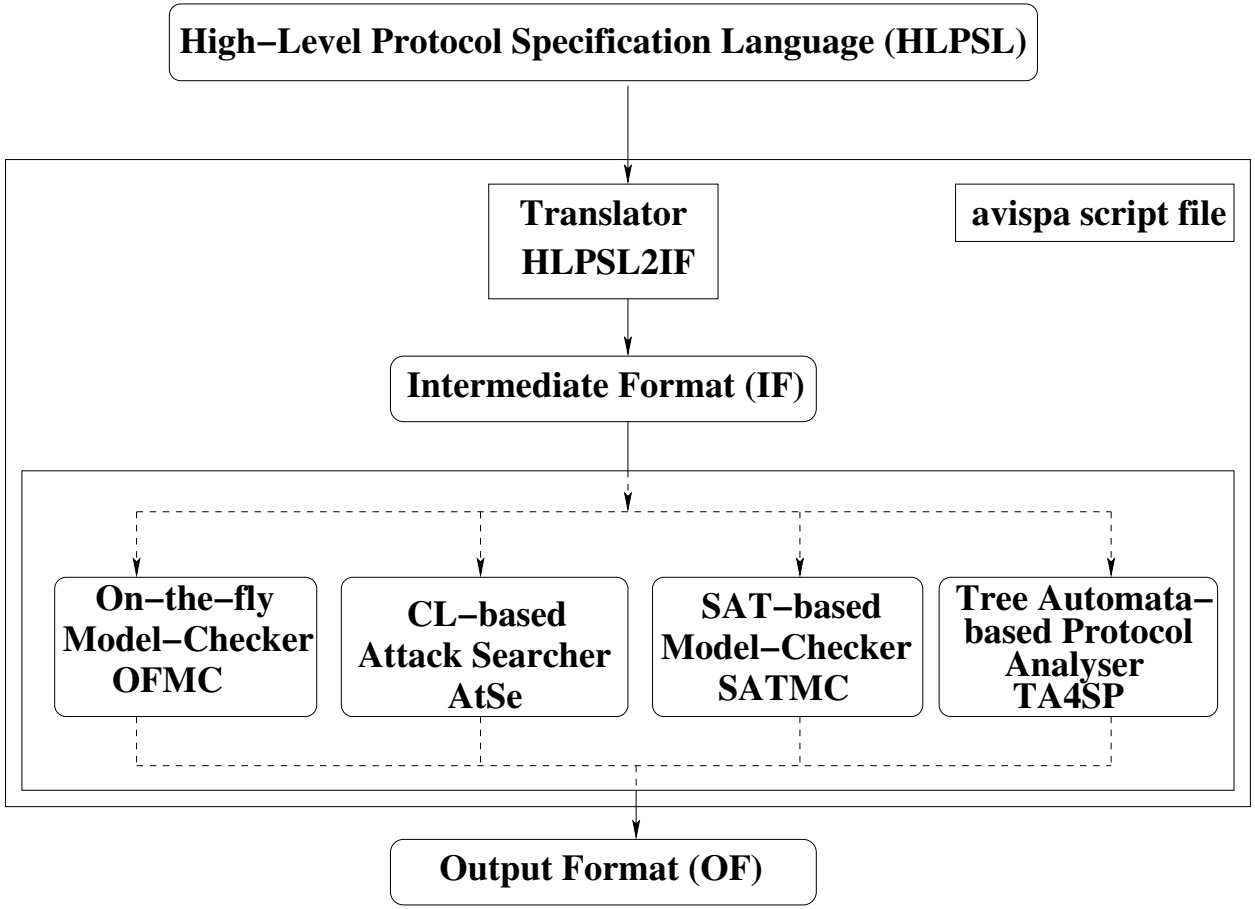


Figure: Architecture of AVISPA (`http://www.avispa-project.org/`)

Simulation results of BSD2C-IoD under OFMC and CL-AtSe backends


```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS

PROTOCOL
 /home/akdas/Desktop/span/
 testsuite/results/IoD-acc.if

GOAL
  as specified

BACKEND
 OFMC

STATISTICS

 TIME 395 ms
 parseTime 0 ms
 visitedNodes: 86 nodes
 depth: 6 plies
```

```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 /home/akdas/Desktop/span/
 testsuite/results/IoD-acc.if

GOAL
 As specified

BACKEND
 CL-AtSe

STATISTICS
 Analysed   : 1 state
 Reachable  : 0 state
 Translation: 0.17 seconds
 Computation: 0.00 seconds
```

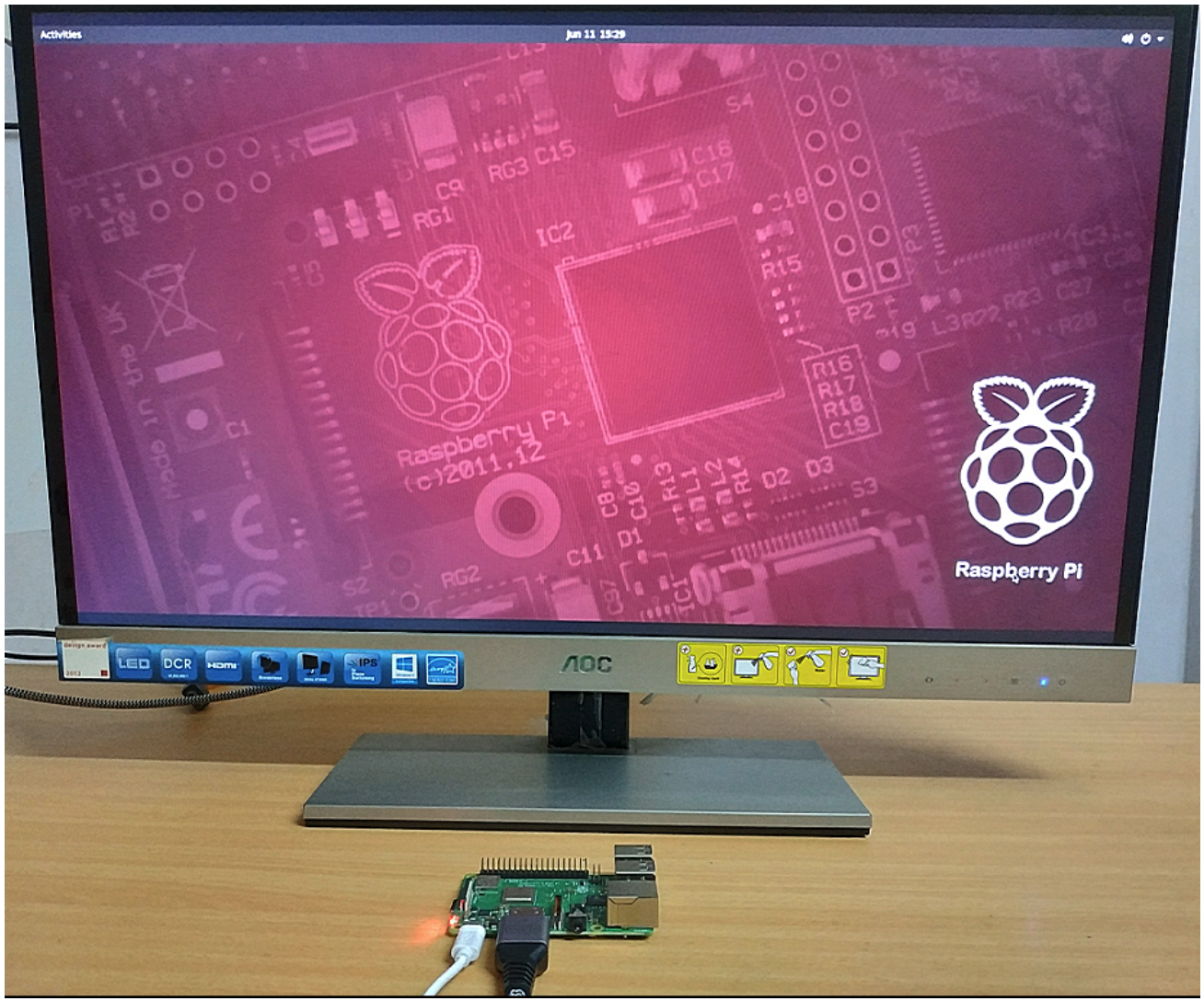
Testbed Experiments using MIRACL under a server

- Multiprecision Integer and Rational Arithmetic Cryptographic Library (MIRACL) [2]
- **Platform #1.** The first platform considered for MIRACL here for a server with the setting as follows: Ubuntu 18.04.4 LTS, with memory: 7.7 GiB, processor: Intel® Core™ i7-8565U CPU @ 1.80GHz $\times$ 8, OS type: 64-bit and disk: 966.1 GB. The experiments on each cryptographic primitive are performed for 100 runs. After that we calculate the maximum, minimum and average time (in milliseconds) for each cryptographic primitive from these 100 runs.

EXECUTION TIME (IN MILLISECONDS) ON A SERVER FOR CRYPTOGRAPHIC PRIMITIVES USING MIRACL

Primitive	Max. time (ms)	Min. time (ms)	Average time (ms)
T_h	0.149	0.024	0.055
T_e	0.248	0.046	0.072
T_{ecm}	2.998	0.284	0.674
T_{eca}	0.002	0.001	0.002
T_{bp}	7.951	4.495	4.716

Experimental setup using Raspberry PI 3



Testbed Experiments using MIRACL under Raspberry PI 3

- **Platform #2.** The second platform that we have considered for MIRACL is as follows: Raspberry PI 3 B+ Rev 1.3, with CPU: 64-bit, Processor: 1.4 GHz Quad-core, 4 cores, Memory (RAM): 1GB, and OS: Ubuntu 20.04 LTS, 64-bit. The experiments on each cryptographic primitive are also executed for 100 runs. We then calculate the maximum, minimum and average run-time (in milliseconds) for each cryptographic primitive from these 100 runs.

EXECUTION TIME (IN MILLISECONDS) ON A RASPBERRY PI 3 FOR CRYPTOGRAPHIC PRIMITIVES USING MIRACL

Primitive	Max. time (ms)	Min. time (ms)	Average time (ms)
T_h	0.643	0.274	0.309
T_e	0.493	0.178	0.228
T_{ecm}	4.532	2.206	2.288
T_{eca}	0.021	0.015	0.016
T_{bp}	32.79	27.606	32.084

Comparative study on communication costs

- Assume that "identity", "random number", "elliptic curve point" $P = (P_x, P_y) \in E_p(u, v)$ where x and y coordinates of P are P_x and P_y respectively, hash output (here SHA-256 hashing algorithm), and timestamp are 160, 160, (160 + 160) = 320, 256 and 32 bits, respectively.
- It is assumed that 160-bit ECC provides the same security level as that for 1024-bit RSA cryptosystem.

Scheme	No. of messages	Total cost (in bits)	Gain
Luo et al. [38]	2	3040	36%
Li et al. [39]	2	3488	56%
Tian et al. [40]	2	$384s$ + 11712	509%
BSD2C-IoD	3	2240	–

Note: s: "no. of pseudonyms of an UAV (drone) in Tian et al.'s scheme" [40] $(s = 5)$

Comparative study on computational costs

- T_h: time required to execute a "one-way cryptographic hash function"
- T_{ecm}: time required to execute an "elliptic curve point multiplication"
- T_{eca}: time required to execute an "elliptic curve point addition"
- T_{bp}: time required to execute a "bilinear pairing"
- T_e: time required to execute a "modular exponentiation"

Scheme	Smart device/drone cost	Gain for a smart device/drone	GSS/server cost	Gain for GSS/server
Luo *et al.* [38]	$T_{bp} + T_h$ ≈ 32.393 ms	194%	$3T_{ecm} + 3T_{bp} + 3T_h + T_{eca} + T_e$ ≈ 16.409 ms	275%
Li *et al.* [39]	$T_{bp} + T_h$ ≈ 32.393 ms	194%	$3T_{ecm} + 4T_{bp} + T_h + 2T_{eca}$ ≈ 20.945 ms	378%
Tian et al. [40]	$8T_e + 9T_h$ ≈ 4.605 ms	–	–	–
BSD2C-IoD	$6T_h + 4T_{ecm} + T_{eca}$ ≈ 11.022 ms	–	$6T_h + 6T_{ecm} + 2T_{eca}$ ≈ 4.378 ms	–

Comparative study on functionality & security attributes

Attribute (A)	Luo *et al.* [38]	Li *et al.* [39]	Tian et al. [40]	BACS-IoD
FSA_1	✓	✓	✓	✓
FSA_2	✓	✓	✓	✓
FSA_3	×	×	×	✓
FSA_4	✓	✓	✓	✓
FSA_5	✓	✓	✓	✓
FSA_6	✓	✓	N/A	✓
FSA_7	✓	✓	✓	✓
FSA_8	×	×	✓	✓
FSA_9	✓	✓	×	✓
FSA_{10}	×	×	×	✓
FSA_{11}	×	×	✓	✓
FSA_{12}	×	×	×	✓

FSA_1: "replay attack"; FSA_2: "man-in-the-middle attack"; FSA_3: "mutual authentication"; FSA_4: "key agreement"; FSA_5: "device/drone impersonation attack"; FSA_6: "GSS/server impersonation attack"; FSA_7: "malicious device deployment attack"; FSA_8: "resilience against drone/device physical capture attack"; FSA_9: "formal security verification using AVISPA tool"; FSA_{10}: "ESL attack under the CK-adversary model"; FSA_{11}: "support dynamic drone/device addition phase"; FSA_{12}: "support blockchain-based solution"

✓: "a scheme is secure or it supports an attribute"; ×: "a scheme is insecure or it does not support an attribute"; N/A: means "not applicable" in a scheme.

Blockchain Implementation

- **Scenario 1:** We assume that the total number of peer-to-peer (P2P) nodes in the CS network is 5. It is worth noticing that the computational time (in seconds) differs for the varied number of blocks mined into a blockchain, where each block contains a fixed number of transactions as 60.

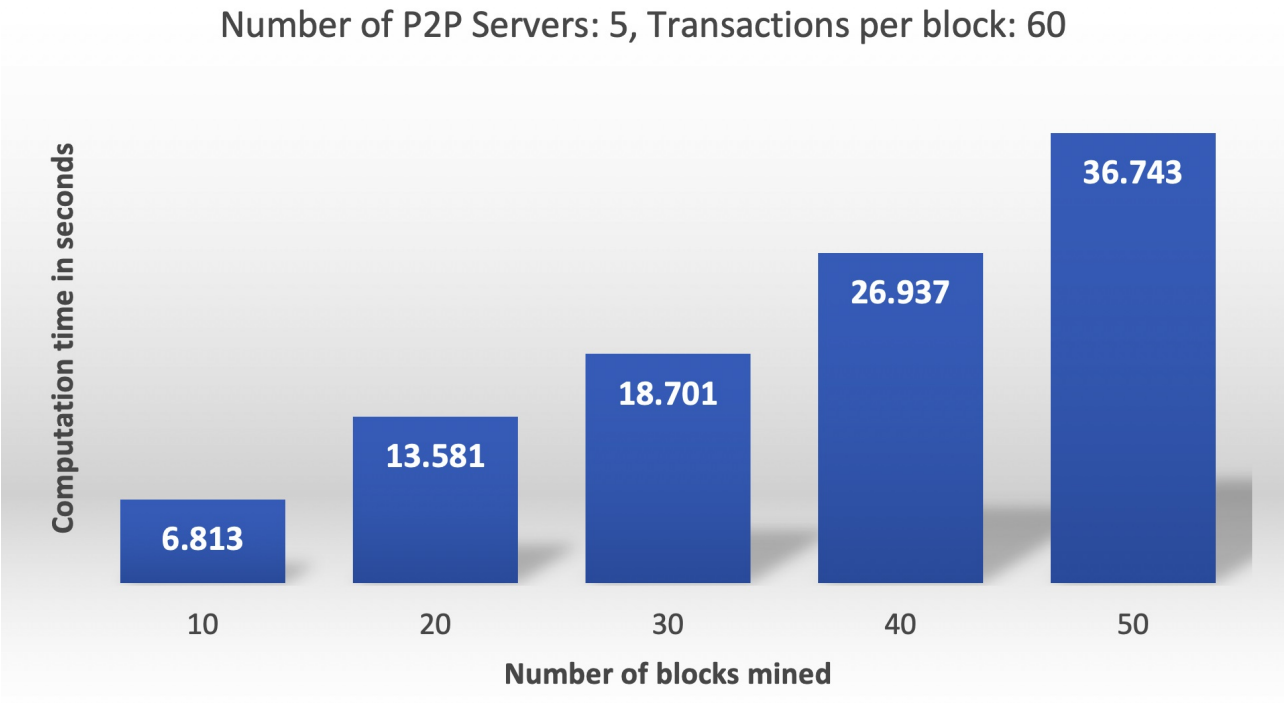


Figure: Blockchain simulation results for Scenario 1

Blockchain Implementation

- **Scenario 2:** In this situation, the total number of peer-to-peer (P2P) nodes in CS network is also considered as 5. We have considered a fixed number of mined blocks as 30. The simulation results show the computational time (in seconds) also differs based on the number of varied transactions pushed per block.

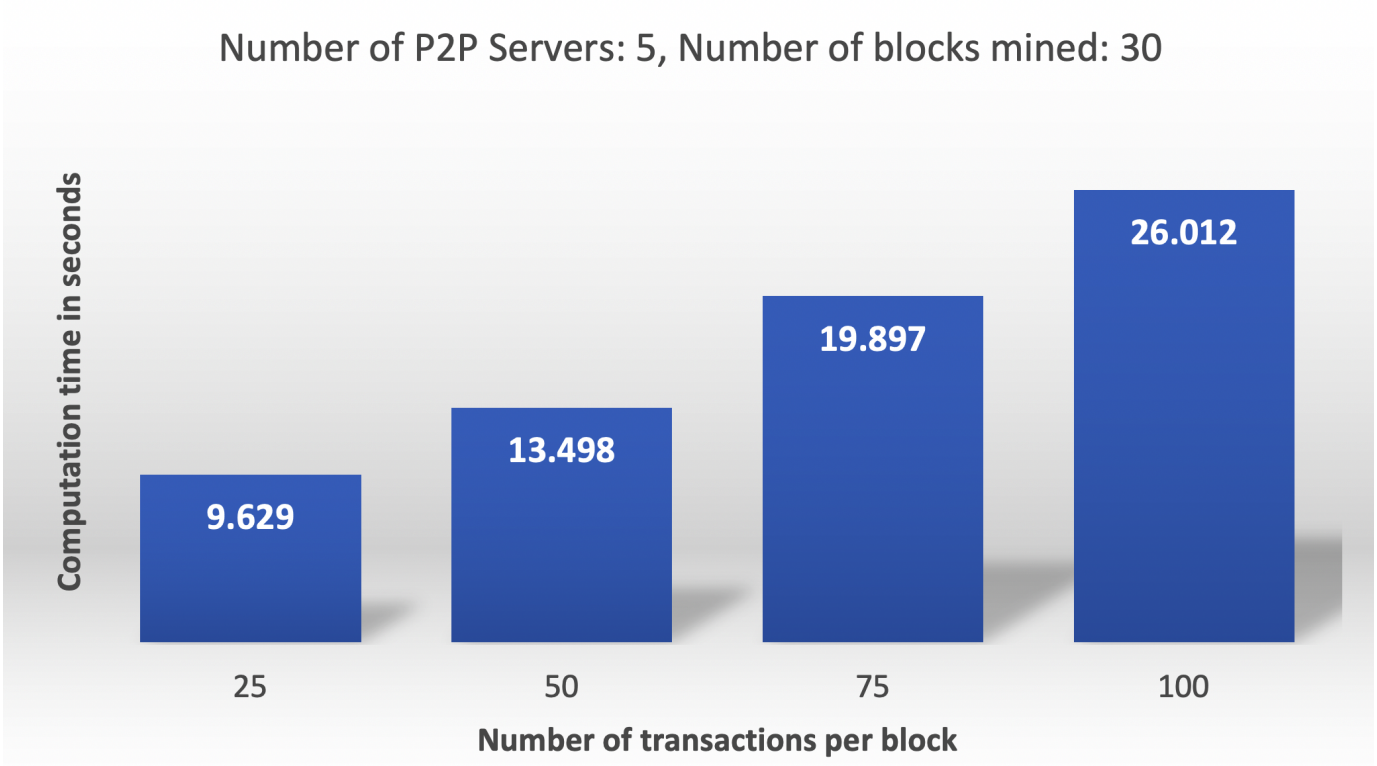


Figure: Blockchain simulation results for Scenario 2

Testbed Experiments

- Under this testbed experiment, we executed the access control scheme.
- The access control mechanism has been implemented in a client-server model paradigm with the socket programming environment in python3 language.
- Each drone *DR* is considered as a client and the ground station server (*GSS*) is assumed to be a server.
- The configuration of each *GSS* is considered as a Laptop configuration under the environment: “Ubuntu 20.04 LTS , 64-bit OS with Intel® Core™ i5-4210U CPU @ 1.70GHz, 4 GB RAM, 130 GB memory partition”.
- Raspberry Pi 3 configuration is “Ubuntu 18.04, Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, 16 GB memory”.

Continuing...

Continuing...

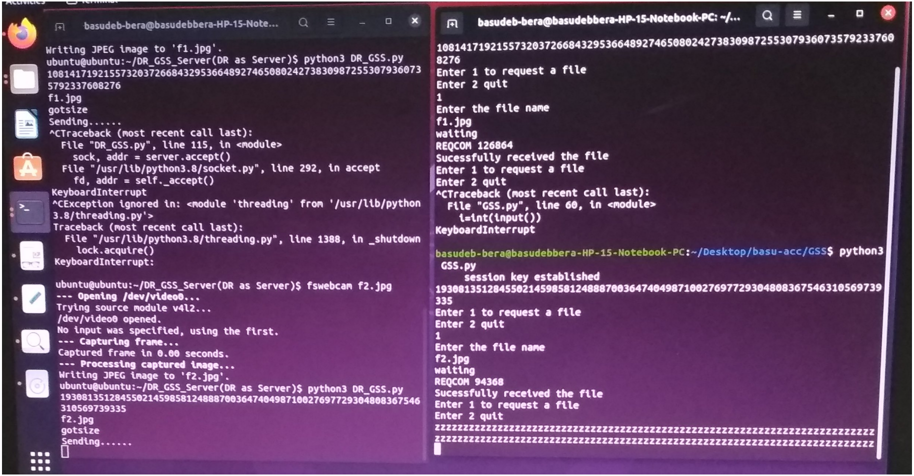


Figure: Testbed experimental results

Continuing...

- It is noticing that the established session key from both sides (*DR* and *GSS*) is the same.
- Once the session key is established between them, *DR* takes a real image (stored in a file, say “f2.jpg”).
- Then sends the image file to the *GSS* by encrypting the image with the same session key.
- The encryption is used here, a symmetric encryption (in this case, we considered the “Advanced Encryption Standard (AES-256)”).

Conclusion and Future Works

Blockchain:

- Blockclain technology and its various applications

Case Study:

- Discussed a novel blockchain-envisioned secure data delivery and collection scheme for 5G-enabled IoD environment (BSD2C-IoD).
- BSD2C-IoD not only provides access control mechanism among the drones and the *GSS* in the flying zones, it also provides secure transactions among the drones, the *GSS*s and the *CR*s in the IoD environment, which are then used to form various blocks by the respective *GSS*.
- Applied ECC-based signature and the security depends on solving ECDLP and collision-resistant one-way hash function.
- Achieves better security, and requires low communication and computational overheads.

Future Research:

- Blockchain-based authentication and access control protocols
- AI-based Big data analytics using blockchain

References


Drones reporting for work.
Goldman Sachs Research, Accessed on January 2020.


MIRACL Cryptographic SDK: Multiprecision Integer and Rational Arithmetic Cryptographic Library, 2020.
Accessed on April 2020.


Debiao He, Neeraj Kumar, and Jong-Hyouk Lee.
Privacy-preserving data aggregation scheme against internal attackers in smart grids.
Wireless Networks, 22(2):491–502, 2016.


Neeraj Kumar, Rahat Iqbal, Sudip Misra, and Joel J.P.C. Rodrigues.
An intelligent approach for building a secure decentralized public key infrastructure in vanet.

Journal of Computer and System Sciences, 81(6):1042–1058, 2015.


Thomas Lagkas, Vasileios Argyriou, Stamatia Bibi, and Panagiotis Sarigiannidis.
UAV IoT Framework Views and Challenges: Towards Protecting Drones as “Things”.
Sensors, 18(11), 2018.


B. Li, Z. Fei, and Y. Zhang.
UAV Communications for 5G and Beyond: Recent Advances and Future Trends.
IEEE Internet of Things Journal, 6(2):2241–2263, 2019.


Seung-Hyun Seo, Jongho Won, Elisa Bertino, You Sung Kang, and Dooho Choi.
A security framework for a drone delivery service.
In *2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet'16)*, pages 29–34, Singapore, 2016.

Thank You
For Your Attention