CS489/689
Privacy, Cryptography, Network and Data Security

PIR, SSE, Blockchains, Bonus Applied Crypto

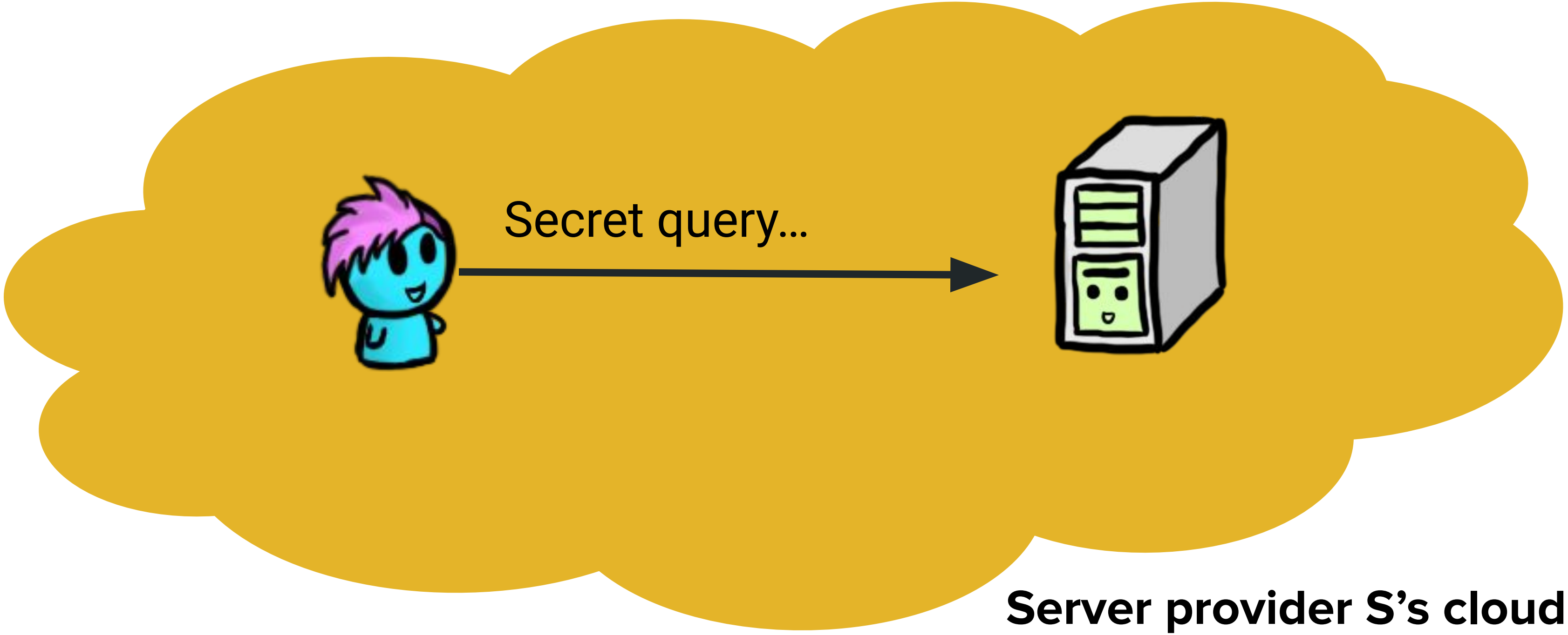
Winter 2023, Tuesday/Thursday 8:30-9:50am

Last two classes!

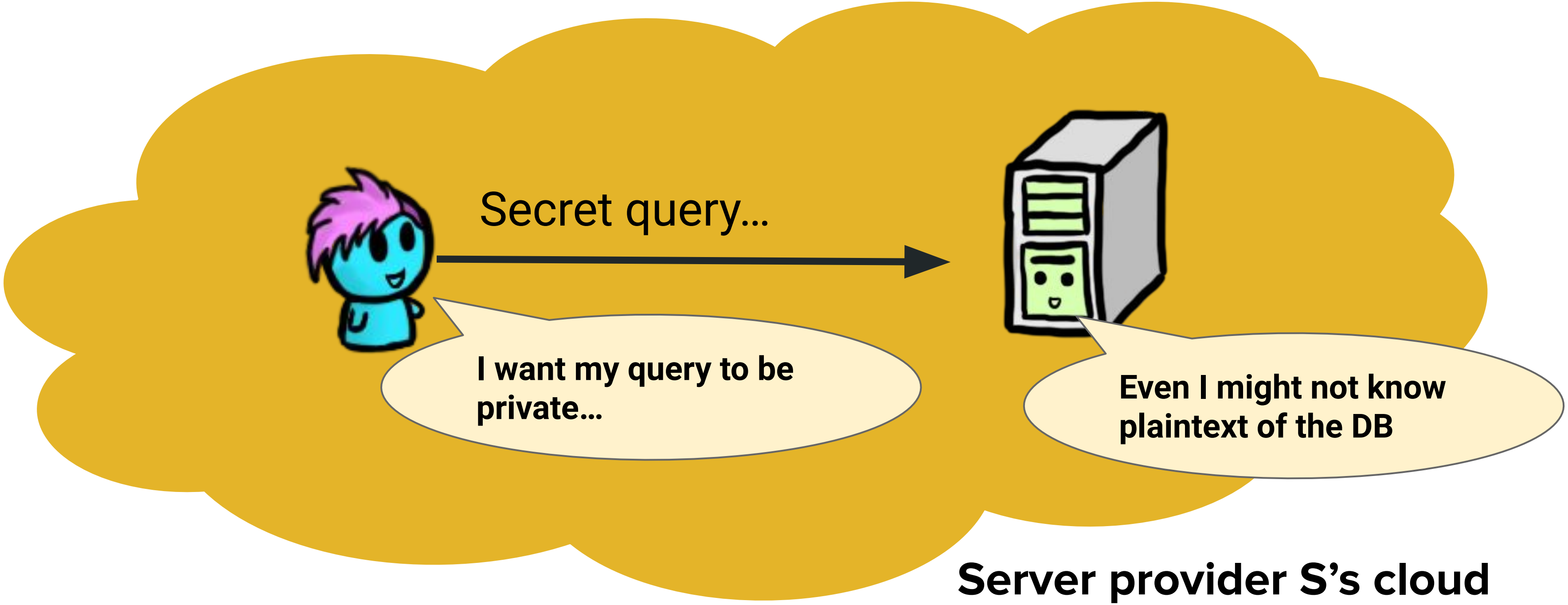
- Private data access (PIR, SSE)
- Blockchains
- Applied cryptography

Private Database Queries

Introduction...



Introduction…



Real-world Example

- The server stores a list of “broken” passwords that appeared on the Internet

Real-world Example

- The server stores a list of “broken” passwords that appeared on the Internet
- The client wants to check whether the password they just created for an Internet site is in that database

Real-world Example

- The server stores a list of “broken” passwords that appeared on the Internet
- The client wants to check whether the password they just created for an Internet site is in that database
- **If it is**, they should not use it
- **If it is not** but revealed to the database, the password should not be used either anymore

Real-world Example

- The server stores a list of “broken” passwords that appeared on the Internet
- The client wants to check whether the password they just created for an Internet site is in that database
- If it is, they should not use it
- If it is not but revealed to the database, the password should not be used either anymore
- Hence, the client needs to query **without revealing the password**

Classification, or Terms for Private Queries

- Server learns matching elements / does not learn matching elements
  - Searchable encryption / Private information retrieval (PIR)

Classification, or Terms for Private Queries

- Server learns matching elements / does not learn matching elements
  - Searchable encryption / Private information retrieval (PIR)
- Keyword / index query
  - Keyword PIR / PIR

PIR

PIR

Carol the client has index i

PIR

Carol the client has index i

Server has DB $d_1, \ldots, d_n$

PIR

Carol the client has index i

Server has DB $d_1,\ldots,d_n$

Goal Correctness: Client learns d_i

PIR

Carol the client has index i

Server has DB $d_1,\ldots,d_n$

Goal Correctness: Client learns d_i

Goal Security: Server does not learn index i

PIR

Carol the client has index i

Server has DB $d_1,\ldots,d_n$

Goal Correctness: Client learns d_i

Goal Security: Server does not learn index i

Catch: There is no security requirement that client only learns d_i

Complexity of PIR

Act.

- Theorem: The server's search complexity for a single query i is O(n)
- Proof:

Q: Why?

Complexity of PIR

- Theorem: The server's search complexity for a single query i is O(n)
- Proof:
  - By contradiction
  - Assume the server does not access (consider) item j for query i
  - The server has just learnt that $i \neq j$

Complexity of PIR

- Theorem: The server's search complexity for a single query i is O(n)
- Proof:
  - By contradiction
  - Assume the server does not access (c … y i
  - The server has just learnt tha…

Caveat: This applies for a <u>single query i</u> but not a set of concurrent queries

Information-Theoretic PIR

- Assume two servers S_1 and S_2 with the same database D

Information-Theoretic PIR

- Assume two servers S_1 and S_2 with the same database D
- The client C constructs an indicator vector $I \in \{ 0, 1 \}^n$
  - $I(i) = 1, I(j) = 0 \ (j \neq i)$

Information-Theoretic PIR

- Assume two servers S_1 and S_2 with the same database D
- The client C constructs an indicator vector $I \in \{ 0, 1 \}^n$
  - $I(i) = 1$, $I(j) = 0$ $(j \neq i)$
- The client chooses a random, binary vector $R \in \{ 0, 1 \}^n$

Information-Theoretic PIR

- Assume two servers S_1 and S_2 with the same database D
- The client C constructs an indicator vector $I \in \{ 0, 1 \}^n$
  - $I(i) = 1, I(j) = 0 \ (j \neq i)$
- The client chooses a random, binary vector $R \in \{ 0, 1 \}^n$
- $C \rightarrow S_1 : Q_1 = R \quad C \rightarrow S_2 : Q_2 = R \oplus I$

Information-Theoretic PIR

- Assume two servers S_1 and S_2 with the same database D
- The client C constructs an indicator vector $I \in \{ 0, 1 \}^n$
  - $I(i) = 1, I(j) = 0 \ (j \neq i)$
- The client chooses a random, binary vector $R \in \{ 0, 1 \}^n$
- $C \rightarrow S_1: Q_1 = R \quad C \rightarrow S_2: Q_2 = R \oplus I$
- Each server S_t computes $s_t = -1^t \sum_{1 \leq j \leq n} Q_t(j)\, d_j$
- $S_t \rightarrow C: s_t$

Information-Theoretic PIR

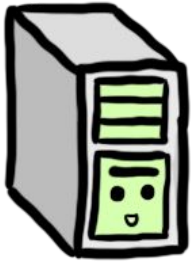
- Assume two servers S_1 and S_2 with the same database D
- The client C constructs an indicator vector $I \in \{ 0, 1 \}^n$
  - $I(i) = 1, I(j) = 0 \ (j \neq i)$
- The client chooses a random, binary vector $R \in \{ 0, 1 \}^n$
- $C \rightarrow S_1: Q_1 = R \quad C \rightarrow S_2: Q_2 = R \oplus I$
- Each server S_t computes $s_t = -1^t \sum_{1 \leq j \leq n} Q_t(j)\, d_j$
- $S_t \rightarrow C: s_t$
- The client computes $d_i = s_1 + s_2$

Notes on IT-PIR

- Requires at least two servers (who cannot collude)
- Communication complexity O(n)

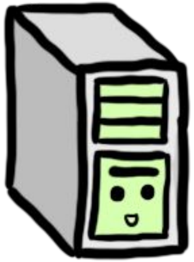
Q: Why?

Computational PIR

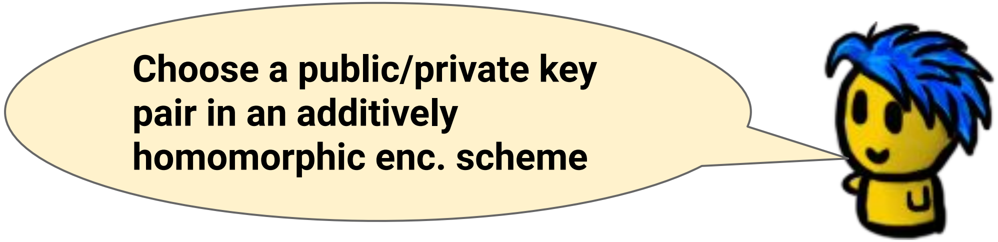


Assume a single server

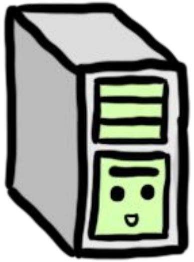
Computational PIR



Assume a single server



Computational PIR



Assume a single server

Choose a public/private key pair in an additively homomorphic enc. scheme

- Let I be the indication vector
- $C \rightarrow S$: $E_C(I(j))$ (for $1 \le j \le n$)
- Server computes $c = \prod_{1 \le j \le n} E_C(I(j))^\wedge(d_j) = E_C(\Sigma_{1 \le j \le n} I(j)\ d_j)$
- $S \rightarrow C$: c
- Client decrypts $d_i = D_C(c)$

Hashing Elements…or Searches?

- Let w be the query keyword
- Let there be a hash function h, such that h(w) = i
- Dealing with collisions:

Hashing Elements…or Searches?

- Let w be the query keyword
- Let there be a hash function h, such that h(w) = i
- Dealing with collisions:
  - Server chooses h, such that there is a “low” number of maximum collisions per bin (lower than a constant) for database D

Hashing Elements…or Searches?

- Let w be the query keyword
- Let there be a hash function h, such that h(w) = i
- Dealing with collisions:
    - Server chooses h, such that there is a “low” number of maximum collisions per bin (lower than a constant) for database D
    - Client downloads h, computes query i = h(w) and submits i to PIR (two-round protocol)

Hashing Elements...or Searches?

- Let w be the query keyword
- Let there be a hash function h, such that h(w) = i
- Dealing with collisions:
  - Server chooses h, such that there is a “low” number of maximum collisions per bin (lower than a constant) for database D
  - Client downloads h, computes query i = h(w) and submits i to PIR (two-round protocol)
  - Query may return constant number of elements as a block (note public database assumption)
  - If D is updated, so may be h

Server computes indicator vector

- Assume fully HE
- Let w_k be the k-th bit of w and $d_{j,k}$ the k-th bit of d_j $(1 \leq k \leq u)$
- $C \longrightarrow S$: $E_C(w_k)$
- For each d_j
  - The server computes $E_C(I(j)) = \prod_{1 \leq k \leq u} \neg(E_C(w_k) \oplus E(d_{j,k}))$
- Caveat: Computation cost is high

Repeated keywords

- So far, each keyword was unique
- How can we deal with repeated keywords?
  - The indicator vector has now multiple 1 entries
  - The size of result set that can be returned in one query is fixed

Searchable Encryption

Searchable Encryption

- Server does learn the matching entries for the query
- Server does not know database

Upload

- Client uploads encrypted database

Upload

- Client uploads encrypted database
- Let H() be a cryptographic hash function and $H_K()$ be a keyed, cryptographic hash function

Upload

- Client uploads encrypted database
- Let H() be a cryptographic hash function and $H_K()$ be a keyed, cryptographic hash function
- Let c_j be a counter for keyword w_j

Upload

- Client uploads encrypted database
- Let H() be a cryptographic hash function and $H_K()$ be a keyed, cryptographic hash function
- Let c_j be a counter for keyword w_j
- Client computes $t_k = H(H_K(w_j)||c_j)$ (|| denotes concatenation)

Upload

- Client uploads encrypted database
- Let H() be a cryptographic hash function and $H_K()$ be a keyed, cryptographic hash function
- Let c_j be a counter for keyword w_j
- Client computes $t_k = H(H_K(w_j)||c_j)$ (|| denotes concatenation)
- Client sorts { t_k } and uploads sorted table T

SSE Query

- Let w be the query keyword
- $C \rightarrow S$: $v = H_K(w)$
- Server computes $t_k = H(v||k)$ $(1 \leq k \leq n)$
- Server returns entries matching t_k in T

Notes on searchable encryption

- One round protocol, independent of result set size
- Server computation complexity $O(\max_j(c_j) \log n)$
- Client communication complexity O(1)
- Server communication complexity $O(\max_j(c_j))$
- Much more efficient than PIR

Efficiency comes at a price

- Server learns which elements match the query

⇒ Same elements, same query

- Assume the server knows which is the most common keyword
  - Abstractly, the server has background knowledge about the frequency of keywords
- Assume the server knows which query is “popular”
  - Abstractly, the server has background knowledge about the frequency of queries
- Etc.

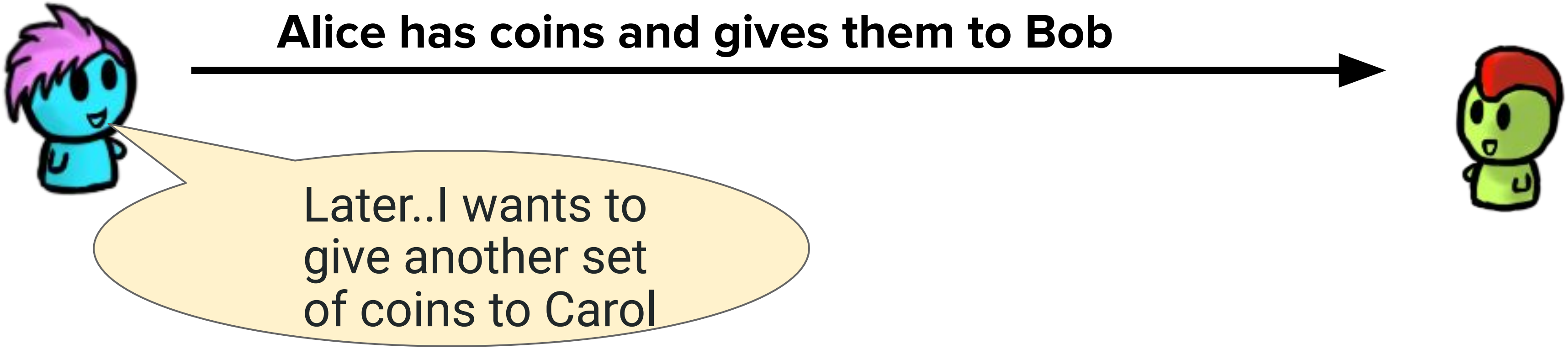
Blockchain

Introduction

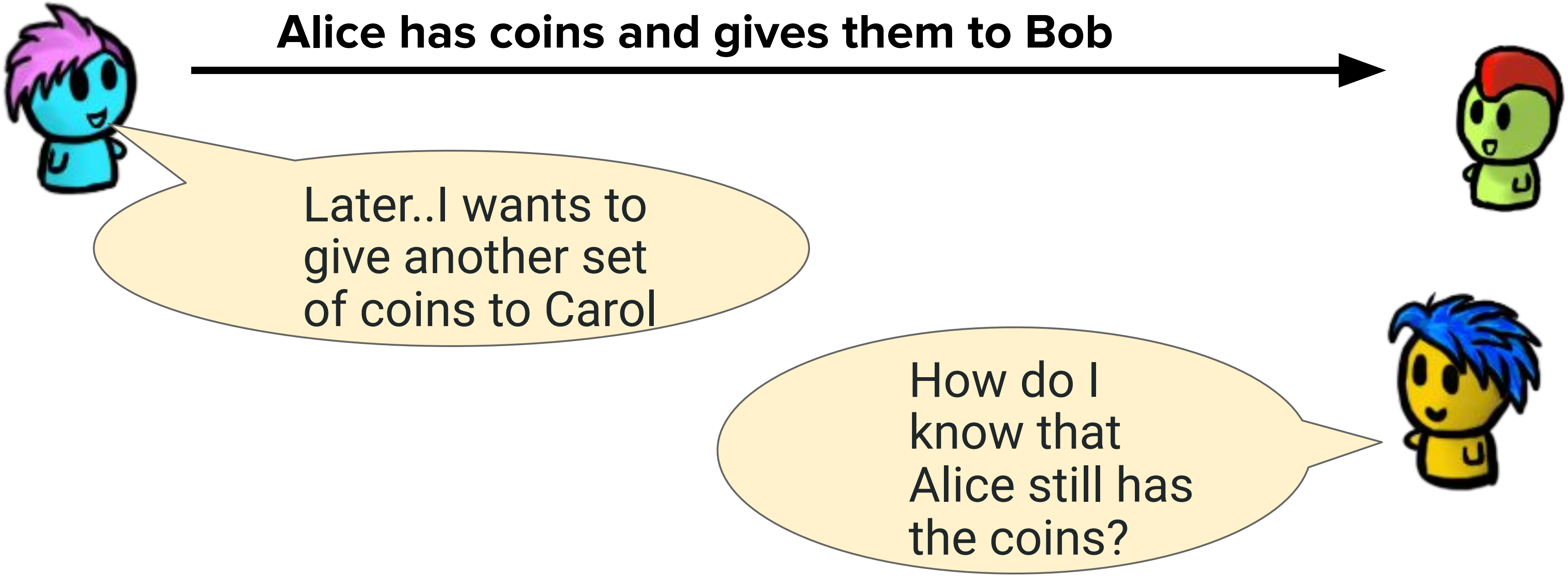
- There is a large and changing number of parties
- There is some state S that all parties share
  - S could be the balance of an account/set of transactions (ledger)
  - S could be the state of a state machine (smart contract)

The double spending problem

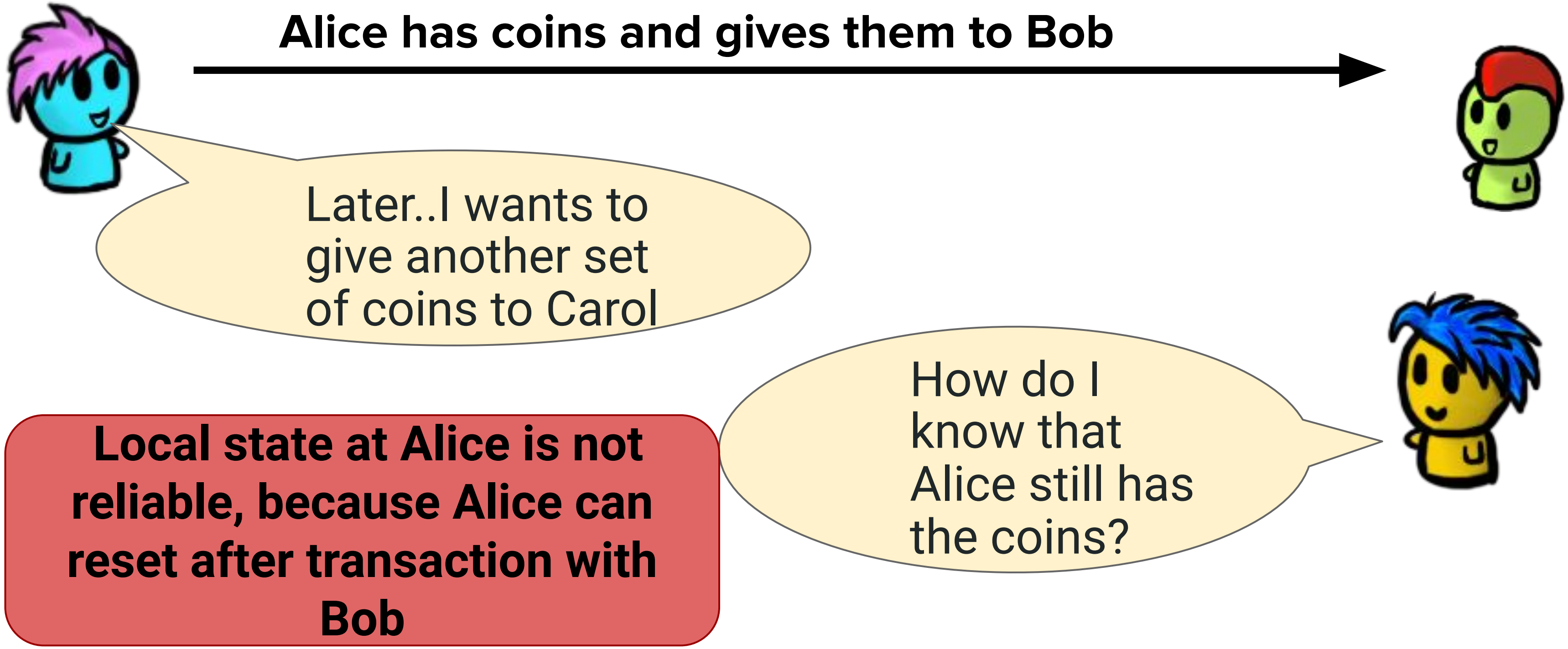
The double spending problem



The double spending problem



The double spending problem



The double spending problem

Alice has coins and gives the

give anot
of c

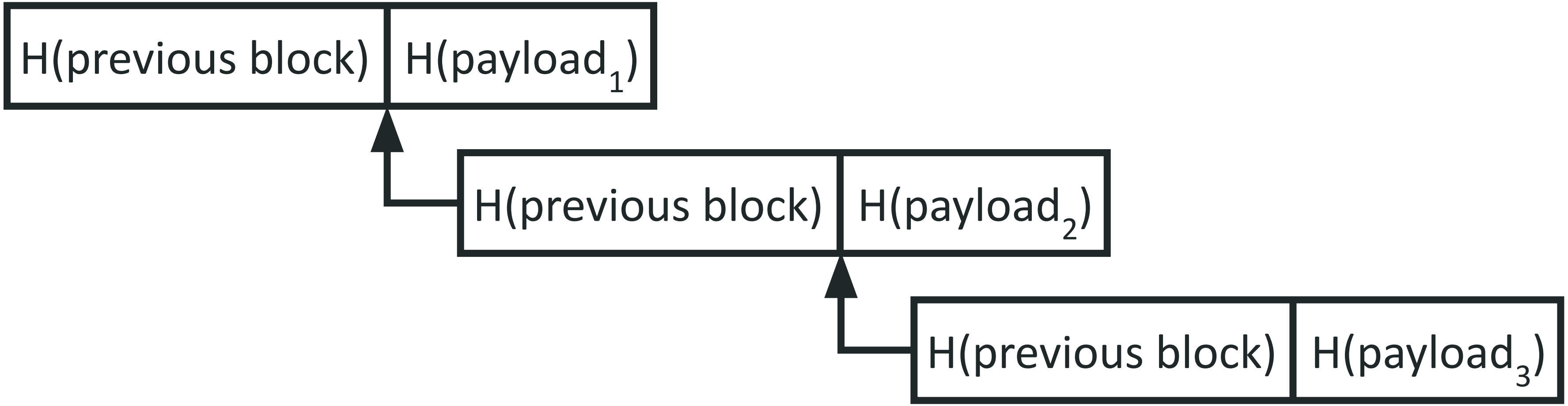
Thus: need a global state
E.g., Trusted party (bank)
Distributed (blockchain)

How do I know that Alice still has the coins?

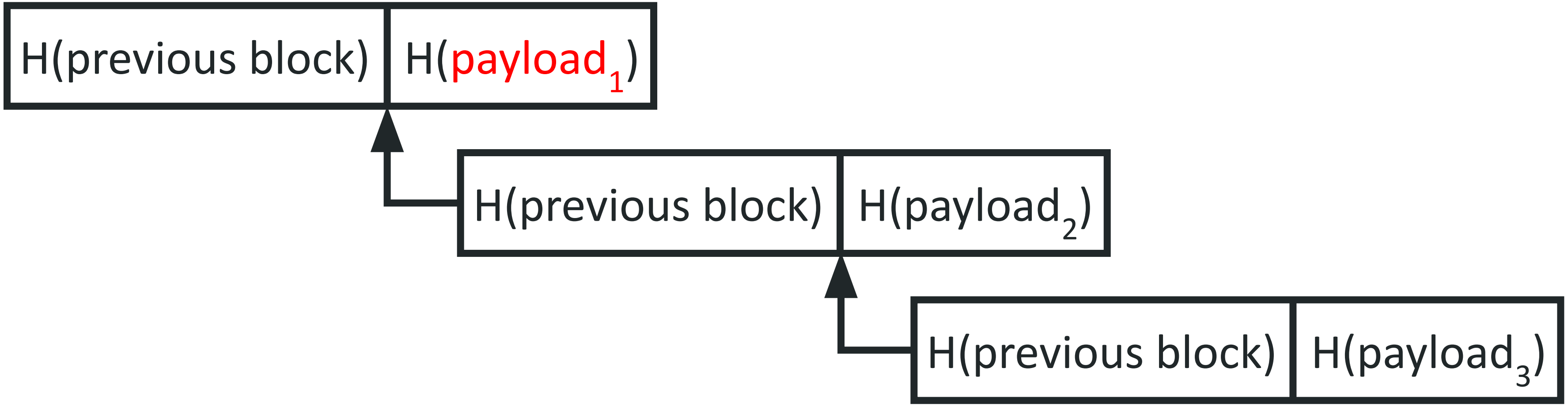
Local state at Alice is not reliable, because Alice can reset after transaction with Bob

What is the state?

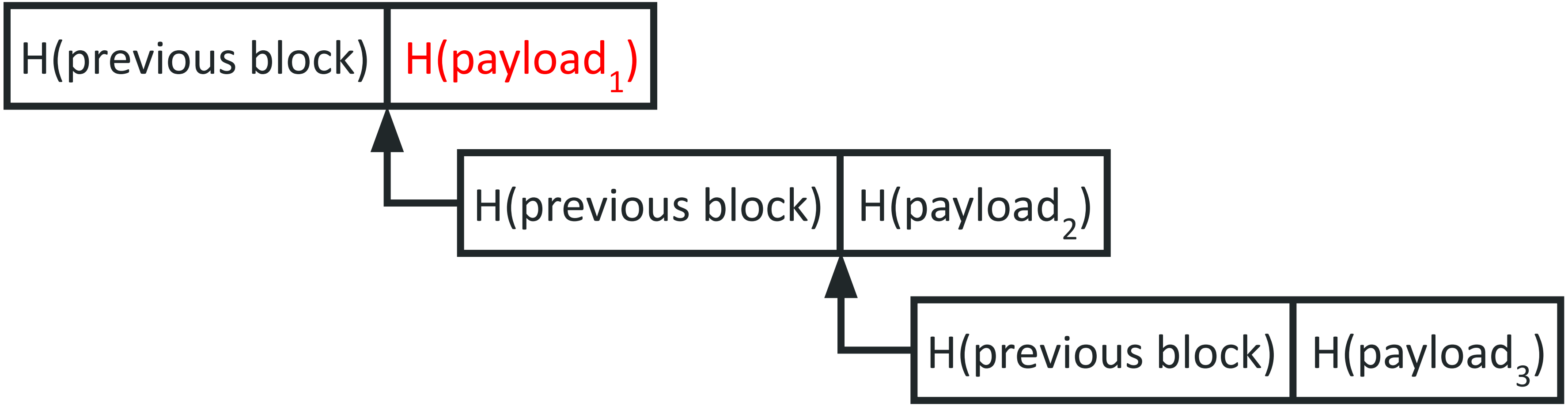
- The state grows (append only): $payload_1$ || $payload_2$ || $payload_3$



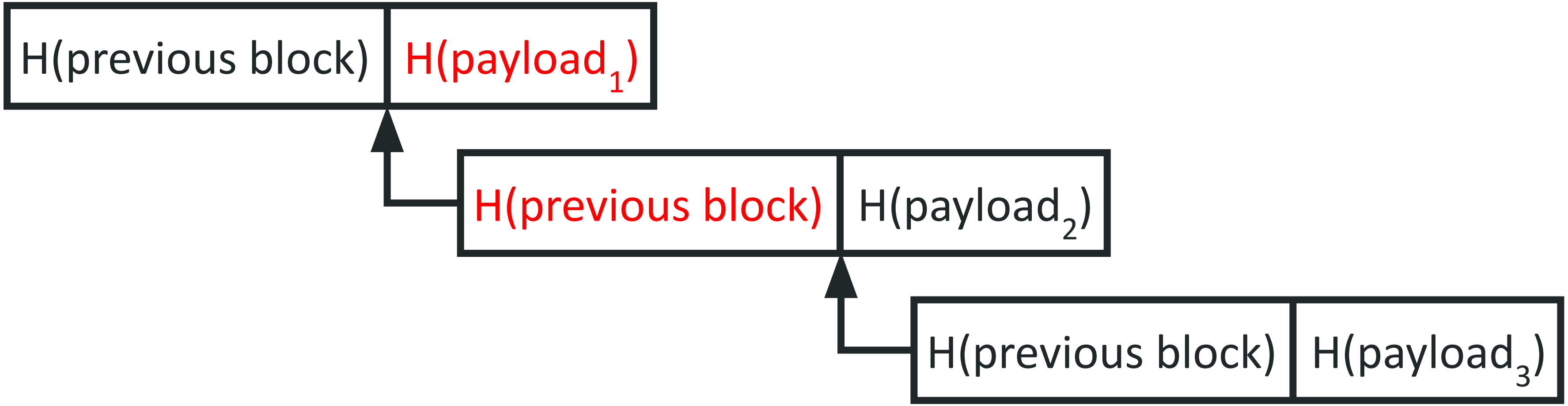
State is append-only



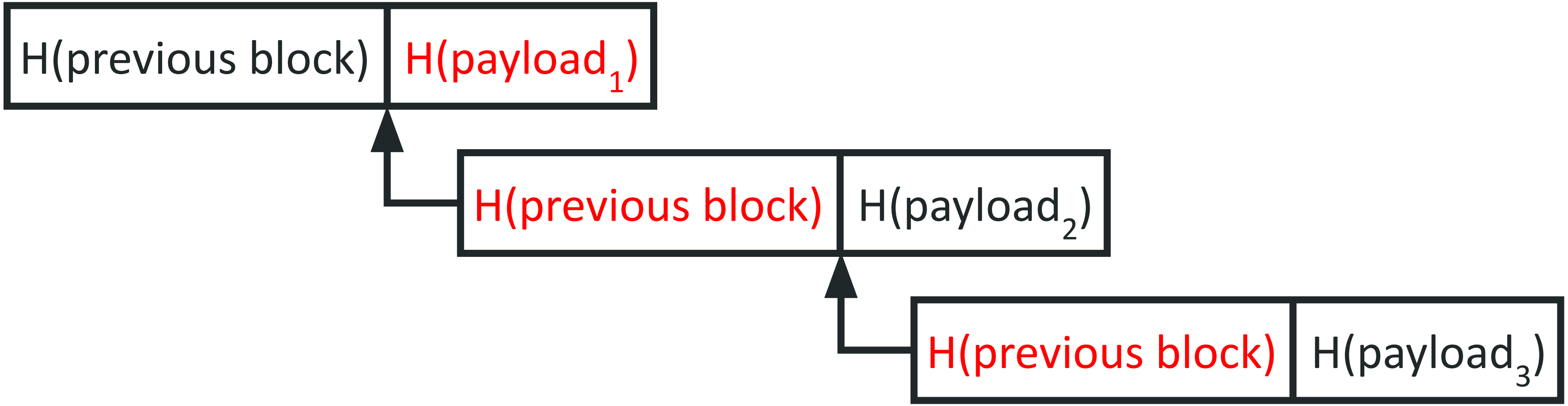
State is append-only



State is append-only



State is append-only



Who can add to the state?

- Key Idea
  - Not everyone can add to the state, but only selected one/few
- Permissioned block
  - Fixed set of signers
- Permissionless block chain
  - Leader election
    - Proof of Work
    - Proof of Stake

Permissioned Blockchain

- Step 1: Signers agree on next state
  - Consensus protocol
    - CS454
- Step 2: Joint signature of block

H_1(previous block)	H_2(payload)	Signature($H_1 \| \| H_2$)

  - Multi-party computation
    - Threshold signature

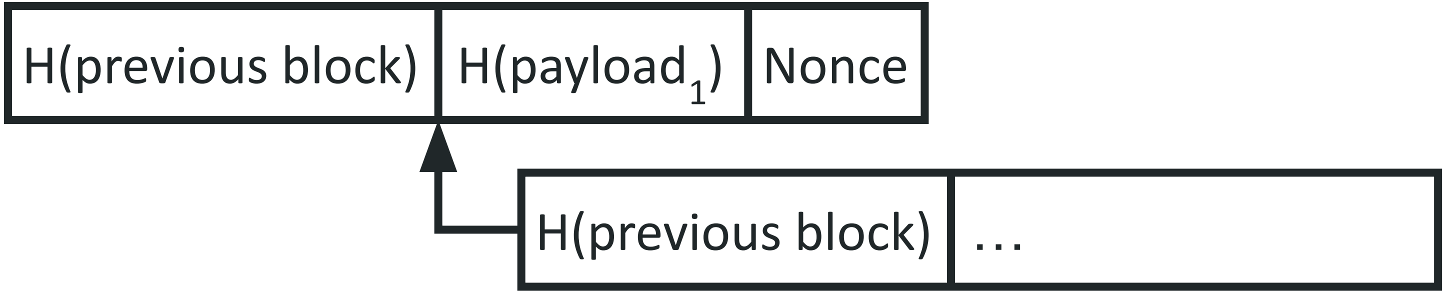
Threshold signature

- Distribution
  - KeyGen: ($pubKey$, $privKey_1$, $privKey_2$, …)
  - $Signer_i$ gets $privKey_i$
- Signature
  - $signature_i$ = Sign(message, $privKey_i$)
  - signature = Reconstruct($signature_1$, $signature_2$, …)
    - At least t out of n: (t, n)-threshold signature
- Verification
  - Verify(signature, message, pubKey) = T / ⊥

Permissionless Blockchain

- Anybody should be able to become signer
  - Open system
- Needs to qualify
  - Computation: Proof-of-Work
  - State-dependent: Proof-of-Stake

Proof of Work



- Requirement:
  - H(previous block) < threshold · No. hash values
    - Starts with leading zeros
- H is a one-way function
  - Need to try different nonce until requirement is fulfilled

Expected number of trials

- Output of hash function is uniform in [0, No. hash values – 1]
- Probabilty of success
  - Pr[H < threshold · No. hash values] = threshold
- Expected number of trials
  - E[H | H < threshold · No. hash values] = 1 / threshold

How to determine threshold

- Constant expected time
  - Bitcoin
- Global threshold:
  - threshold = prev. thres. · (2016 · 10 minutes) / (time since 2016th last block)
- Adjusts to the number of parties in the system (miners)

What if the miner is malicious?

- In permissionless blockchain, step 1 (consensus) of permissioned blockchain is missing
- There is no guarantee that the miner will use agreed payload

Nakamoto Consensus

- Assume other (honest) parties somehow agree on “valid” state
- They will ignore block and continue with previous state

$\Rightarrow$ There can be different chain ends in the system

What is the agreed state

- If an honest miner agrees with more than one state, which one should they append to?
- Answer: The longest

Why?

- The longest chain has used the most work
- If majority of miners is honest, then the longest chain has used the most honest miners’ work
- Therefore if a majority is honest (> 50%), the longest chain will be appended the fastest
- Honest parties outrun malicious parties
  - What if the adversary controls >50% of the computing power?

Careful

- Honest parties not always in the lead
- Need to wait until payload has “settled”
- E.g. in Bitcoin, the recommendation is 6 blocks (60 minutes)

Caveat

- Computing hash values does not provide useful results
- Incentive is by transaction fee (a share of each transaction's value)
- Massive energy loss
  - As much as Argentina (May 2022, https://news.climate.columbia.edu/2022/05/04/cryptocurrency-energy/)
  - 0,5% of all electricity consumed in the world (September 2021, https://www.nytimes.com/interactive/2021/09/03/climate/bitcoin-carbon-footprint-electricity.html)
  - Daily tracking: https://ccaf.io/cbeci/index

Proof of Stake

- Stake is a function of the state
  - E.g. account balance divided by total wealth
- Similar idea to mining
  - Signature (Verifiable Random Function)

  Sign(H(previous block), timestamp) < stake · No. VRF values

What if the miner is malicious?

- Honest miners can punish malicious miner
  - Burn stake (e.g. set account balance to 0)
- With multiple chains, how to determine which is the "valid" one?
  - Ethereum:
    - Punish forker (equivocator)
    - Vote on "valid" chain
      - Punish minority votes
- How to determine "valid" chain, when new to the system?
  - Ethereum: Trusted nodes broadcast last hash

What is in a state?

- Transactions
  - Sender public key
  - Recipient public key
  - Amount
  - Signature by sender public key
- Smart contract
  - Sender public key
  - Opcodes (program)
  - Storage

How to validate transaction?

- Check account balance (from blockchain state)
- Verify signature

Smart contract execution

- Read old storage / program (of blockchain state)
- Execute program
  - Use volatile memory (stack)
- Write updated storage

How to validate smart contract execution?

- Need to execute program and verify updated storage
- What if the program does not hold (or takes very long)?
  - Limit number of computational steps
    - Charge per step
  - Stop if limit has been exceeded
    - Ethereum: Gas limit

Block Chain and Private Data?

Block Chain and Private Data?

Caveat: Blockchain compatibility with sensitive data?

Other exercises

Exercise

- Download data sets D_1, D_2 from XXX
- Encrypt each keyword in D_1 with searchable encryption
- Use D_2 to attack D_1
  - Query for each keyword in $D_1 \cap D_2$
  - Observe the matching entries in D_1 (server's view)
  - Use D_2 to guess the keywords in D_1

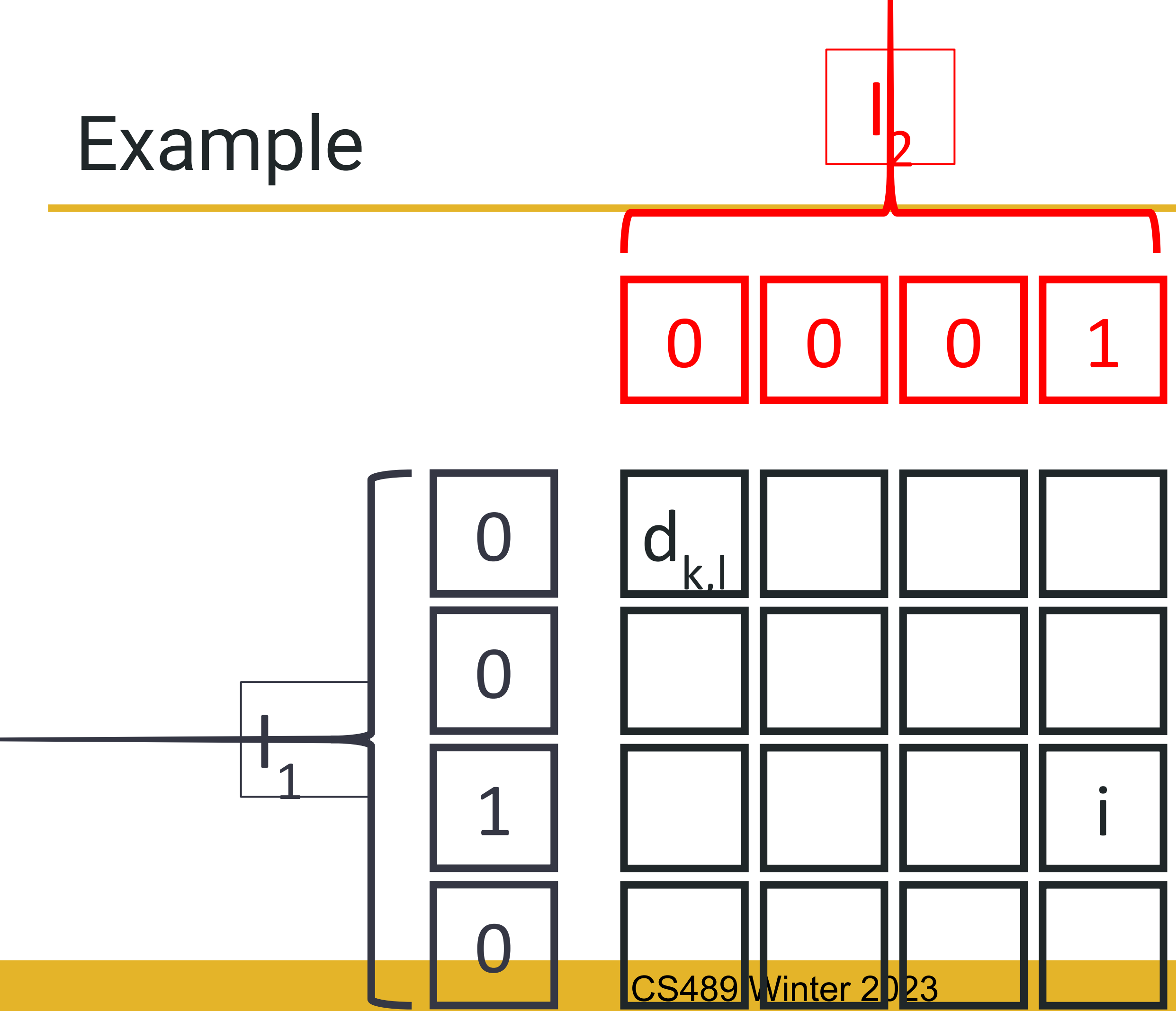
Notes

- So far communication complexity O(n)
- We can do better in C-PIR

Folding

- Server arranges database into a f-dimensional hypercube (we consider f = 2 square)
  - $d_{k,l} = d_i$ ($i = k\sqrt{n} + l$)
- Clients generates f (two) indicator vectors I_1 and I_2
  - $I_1(k) = 1$, $I_2(l) = 1$ (else 0)
- Server computes $c_l = \prod_{1 \le j \le \sqrt{n}} E_C(I1(j)) \wedge (d_{j,l})$ (for $1 \le l \le \sqrt{n}$)

Example



Folding II

- If $E_C()$ is fully homomorphic
- Server computes
  - $e = \sum_{1 \le j \le \sqrt{n}} E_C(I_2(j)) \cdot c_{j,m}$
- Client decrypts decrypts $e = D_C(e)$

Folding III

- If $E_C()$ is additively homomorphic
- Server splits c_l into m "chunks" $c_{l,m}$
  - Note that Paillier ciphertexts are twice as long as their plaintexts
- Server computes
  - $e_m = \Pi_{1 \le j \le \sqrt{n}} E_C(I_2(j)) \wedge (c_{j,m})$
- Client decrypts $c_{l,m} = D_C(e_m)$, reconstructs c_l, decrypts $d_i = D_C(c_l)$

Exercise

- What is the communication complexity of C-PIR for f=2?
- What is the communication complexity of C-PIR for any f?
- What is the lower bound of the communication complexity of folding for C-PIR?

Keyword Queries

- Index queries: Dense, each index returns an element
- Keyword queries: Sparse, very few keywords return an element

$\Rightarrow$ Indicator vector becomes large compared to database size

Notes on C-PIR

- Computational cost of HE is very high
- Additively HE overhead is higher than fully HE overhead
- Additively HE overhead is so high, that downloading the entire database is faster

Exercise

- Create smart contract to accept mined nonce for
  - H(“University of Waterloo rules” || nonce)
  - With at most 2^{20} hashes
- https://ethereum.org/en/developers/

Returning the first result

- Let I be the indicator vector of length n
- Compute prefix $P_1 = \sum_{1 \le k \le j} I(k)$ (for $1 \le j \le n$)
- Compare prefix P_1 to 1^n: $I' = (P_1(j) - 1)$^$(p-1) \bmod p$
- Compute prefix $P_2 = \sum_{1 \le k \le j} I'(k)$ (for $1 \le j \le n$)
- Compare prefix P_2 to 1^n: $I'' = P_2(j) - 1)$^$(p-1) \bmod p$
- Caveat: Using clever math this can be optimized to one comparison

Example

I	0	1	0	1	1	1	0	0
P1	0	1	1	2	3	4	4	4
I’	0	1	1	0	0	0	0	0
P2	0	1	2	0	0	0	0	0
I’’	0	1	0	0	0	0	0	0