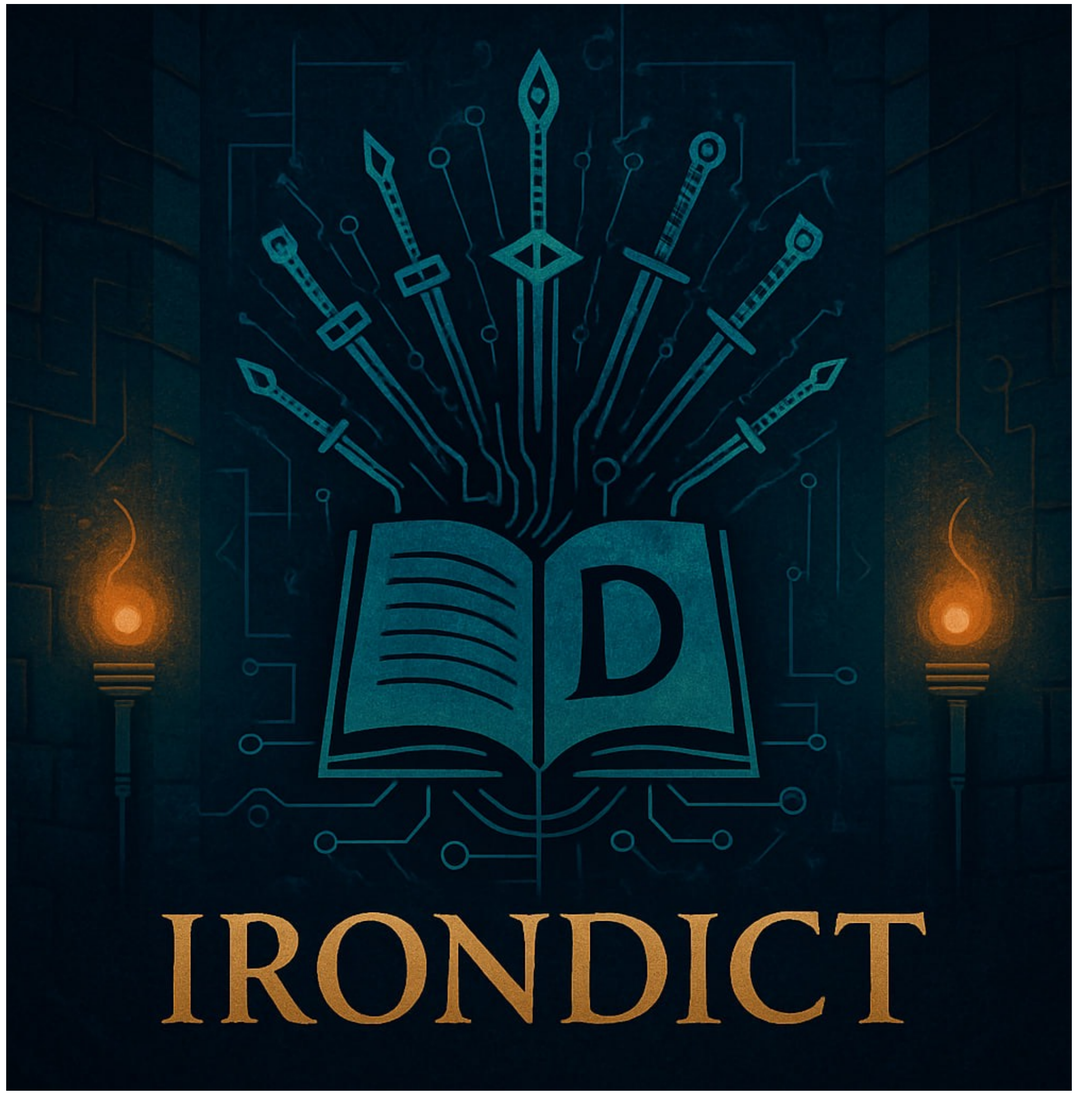


Hossein Hafezi Alireza Shirzad Benedikt Bünz Joseph Bonneau

Secure public keys exchange in secure messaging

John Snow

Request public key of

Secure public keys exchange in secure messaging

John Snow

Request public key of

Secure public keys exchange in secure messaging

John Snow

Request public key of

Public key of


YOU KNOW NOTHING,
JON SNOW


Transparency

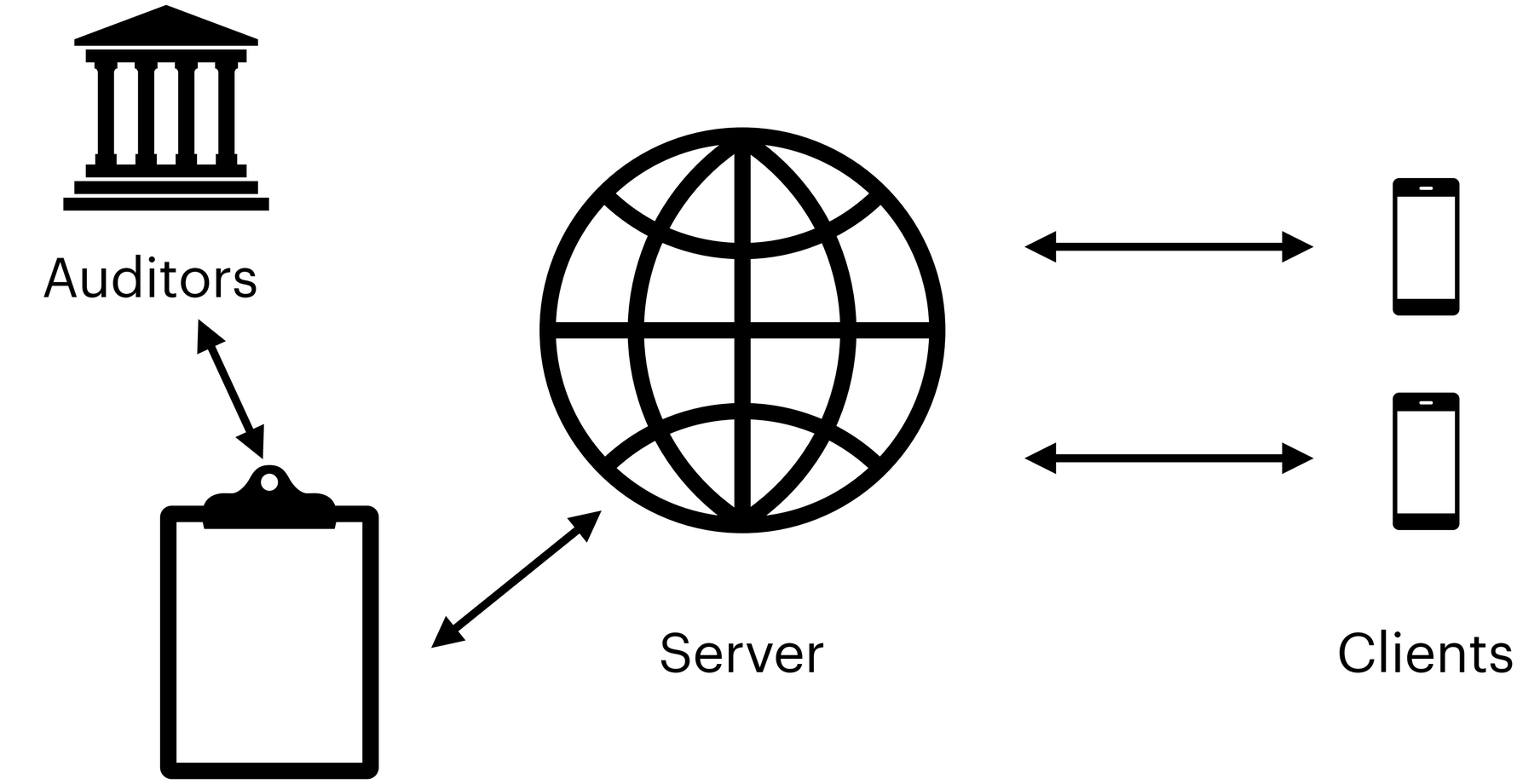
Goal of transparency

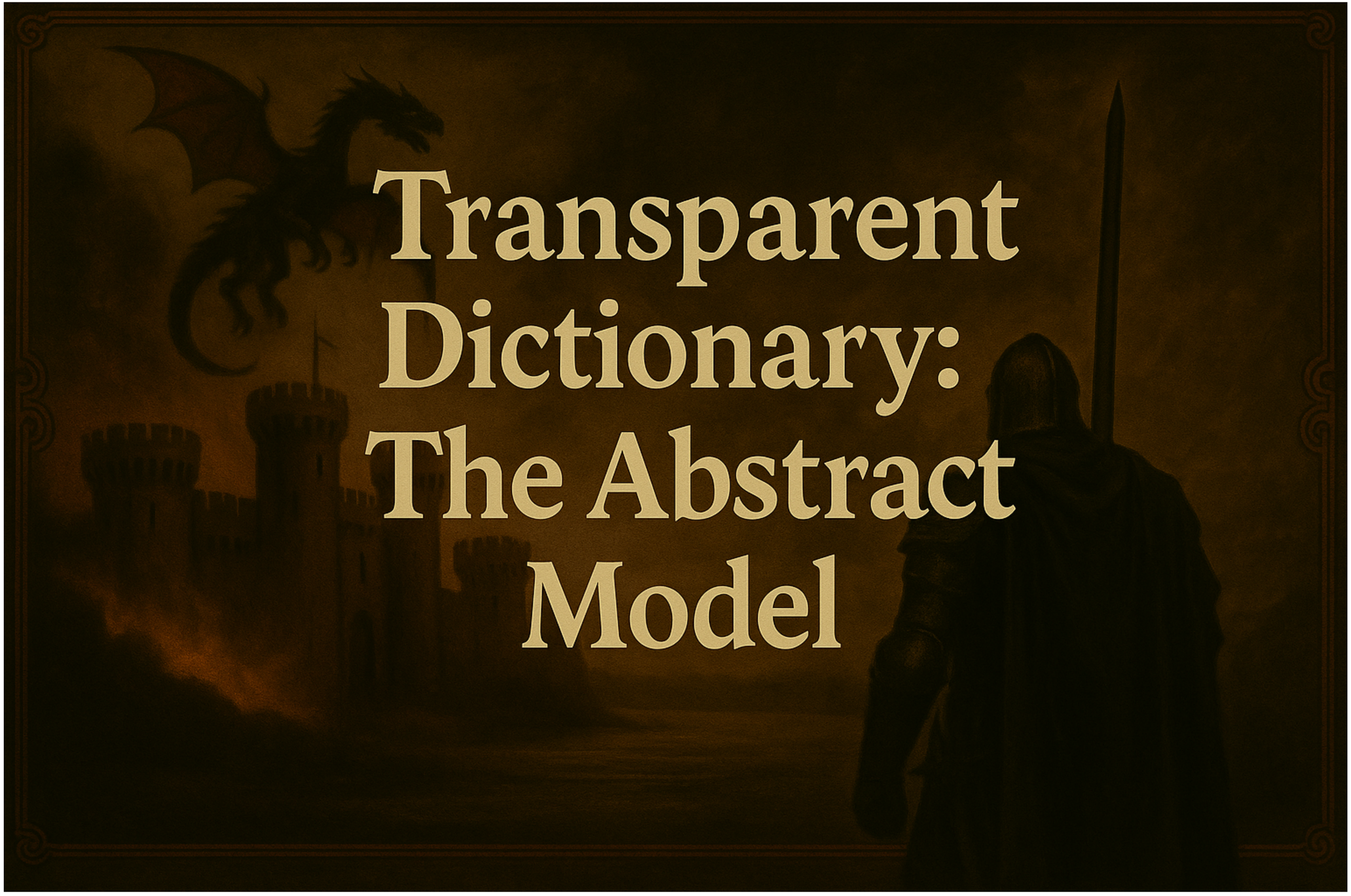
- Same view of the server state for all.
- Server replies back consistently with its state.
- Responses are consistent for everyone.

Checks her pk is correct

Gets the same checked pk

Transparency model




Transparent
Dictionary:
The Abstract
Model

General model of transparent dictionaries

- Dynamic dictionary mapping **labels** to **values**.
- Dictionary has different states in different *epochs*.
- Generic concept ==> Different application

Different applications of TD

public key directory

+(98)9123730591	**pk_1**
+(98)9153004764	**pk_2**
+(98)9333805288	**pk_3**

software distribution

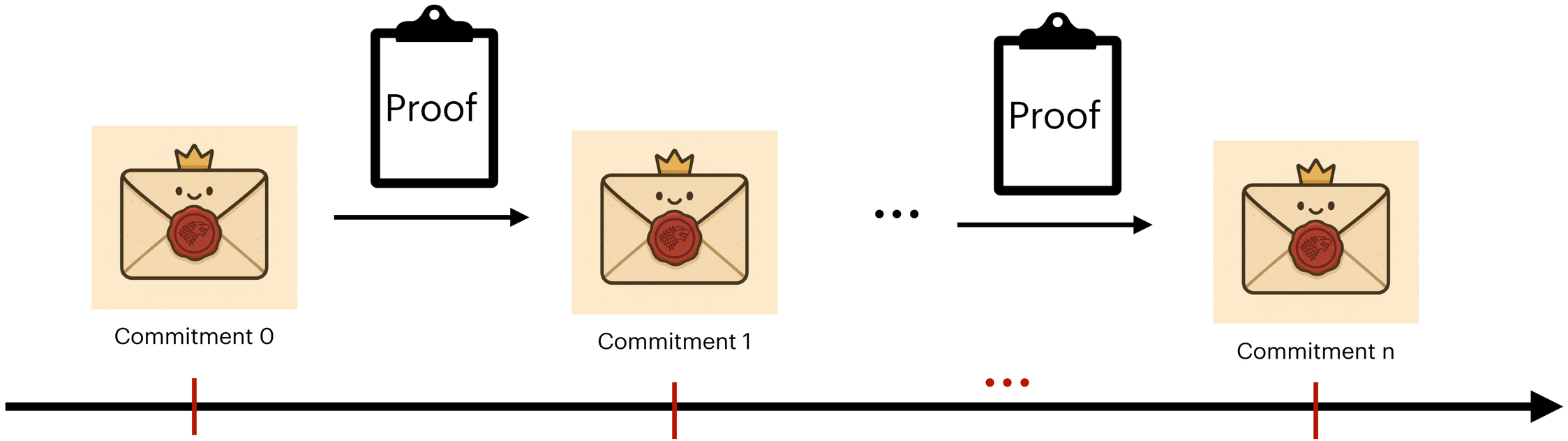
Slack 1.1	**SHA256(slack.exe)**
Telegram 5.2	**SHA256(telegram.exe)**
Signal	**SHA256(signal.exe)**

Important application: public key directory

- Public Key Directories
  - As large as 2 billion ==> WhatsApp users’ size in 2020
  - Requires throughput of 200 updates/s
  - Requirement of privacy (not important for software distribution)

How to Make Transparent Dictionary?

- Server publishes commitment to each state on a bulletin board
- Server proves some invariant between consecutive epochs holds



Expected Functionalities from TD

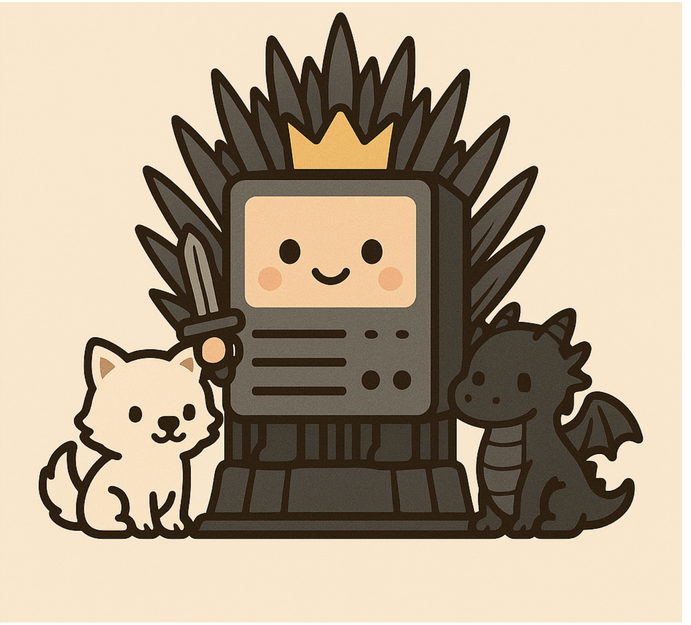
John Snow

(I) what is Ygritte’s latest public key?

Lookup operation

(II) Prove my pk hasn’t changed since last year?

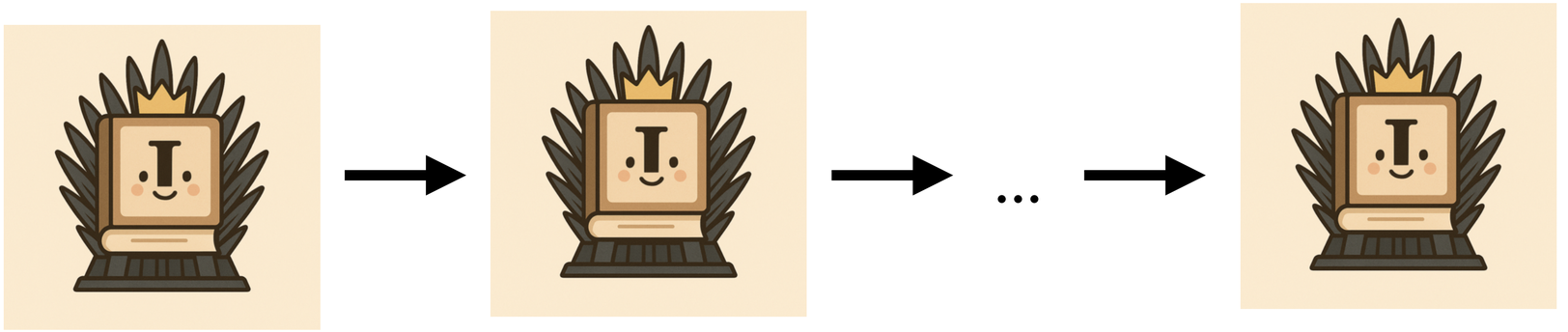
Proof of consistency



Server

Why John Snow wants to know his public key hasn't changed from epoch i to j?

Assume John snow is beyond the wall with no internet from state i+1 to state j-1



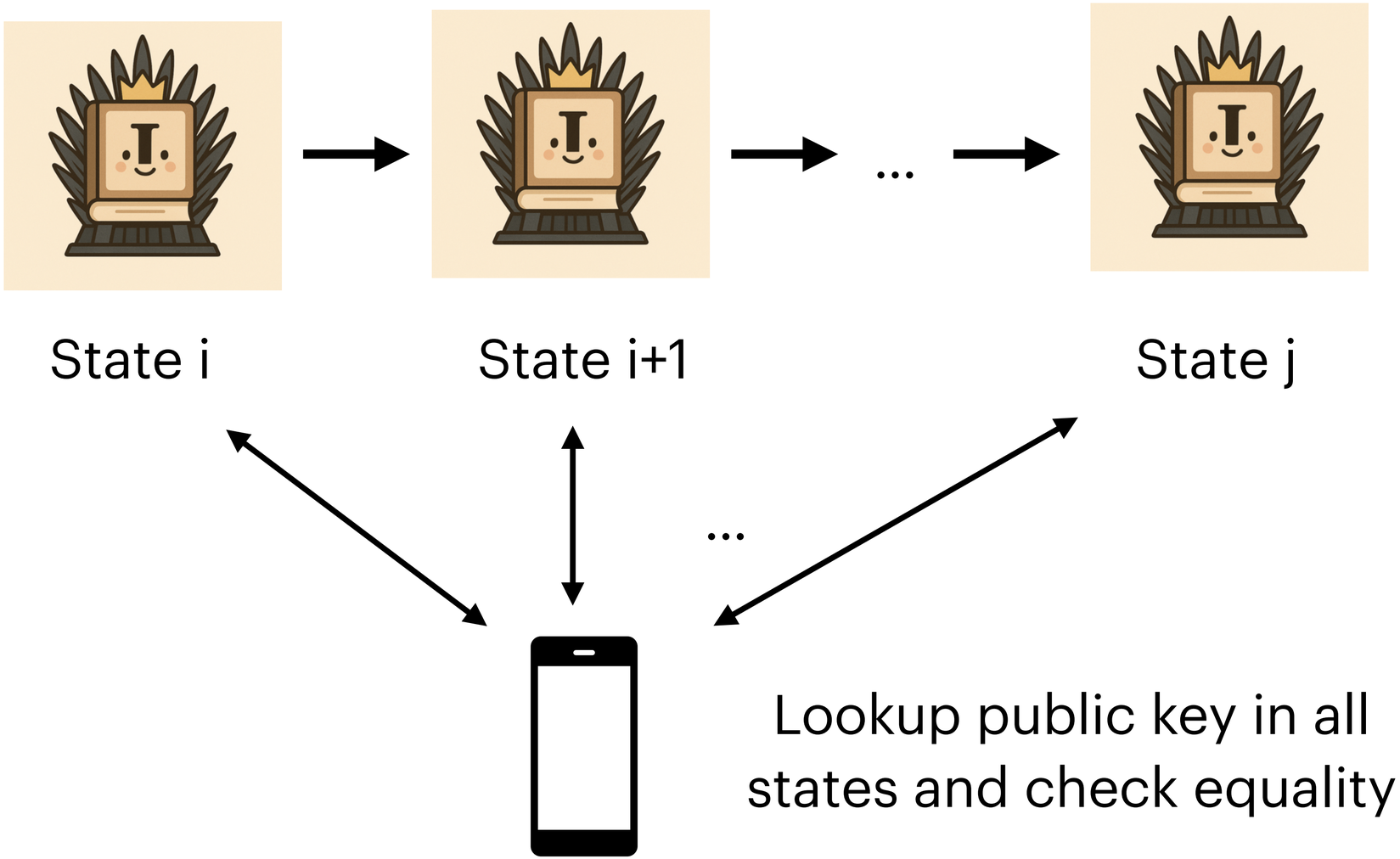
State i

State i+1

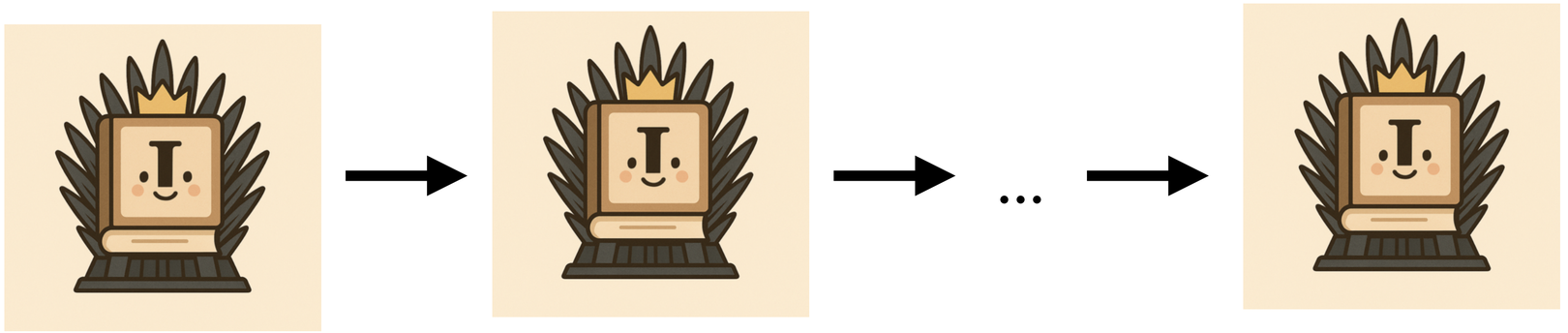
State j


Different
Solutions to
Value
Consistency

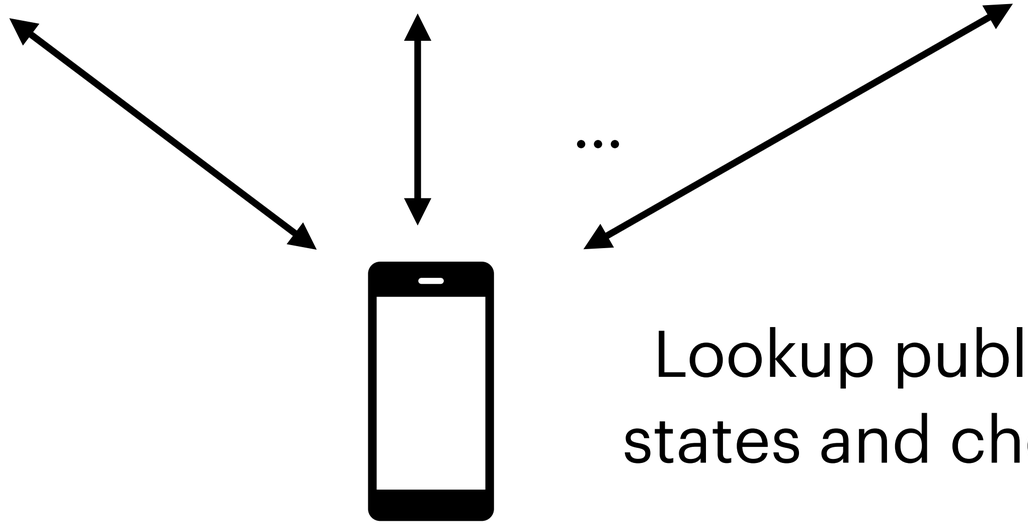
Naive Value Consistency



Naive Value Consistency



State i State i+1 State j

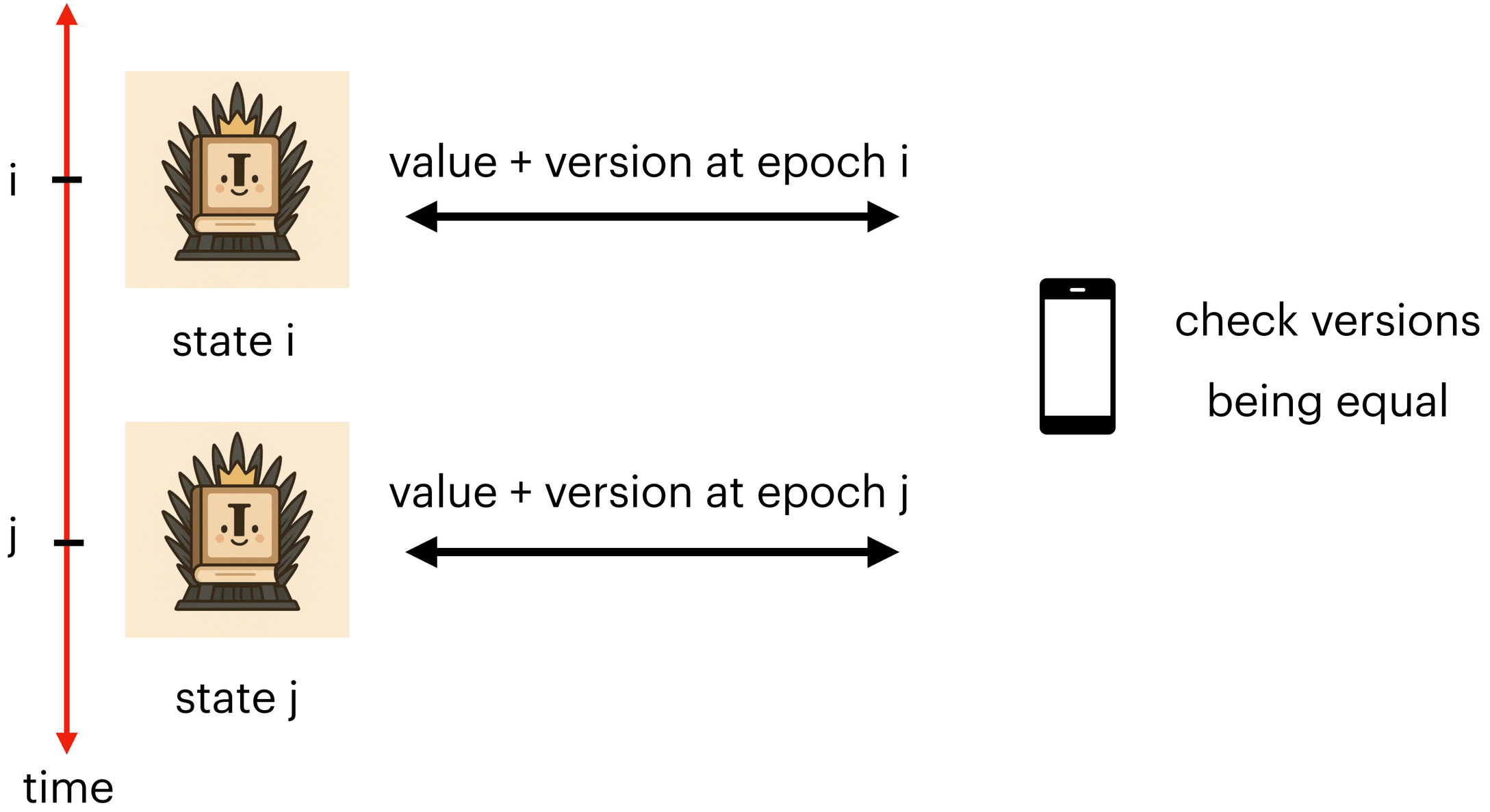


Lookup public key in all states and check equality

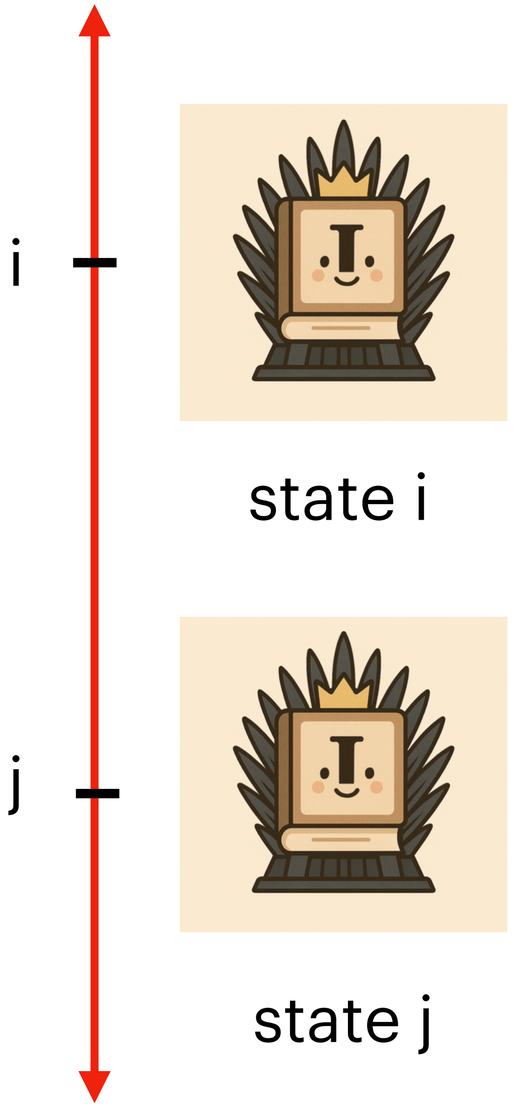
Version Invariant

- Server sets a version for each value in the dictionary
- Initially the version for each value is zero.
- Each time the value is updated, the version increments.
- The server **promises** to increment versions ⟷ the values changes.

Version Invariant



Version Invariant

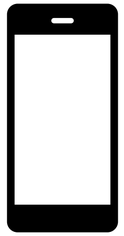


state i

state j

value + version at epoch i

value + version at epoch j

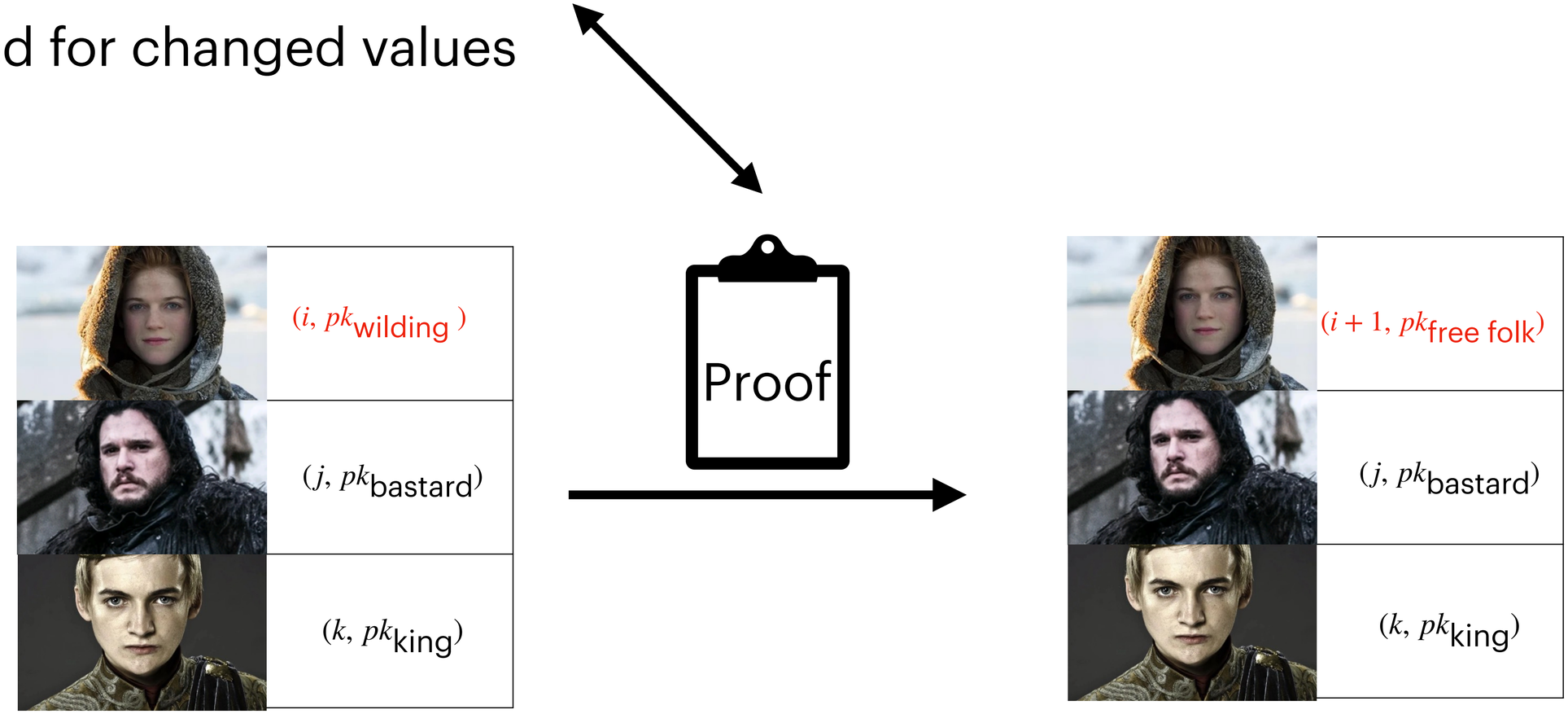


check versions being equal

How to make sure server is not an oathbreaker?

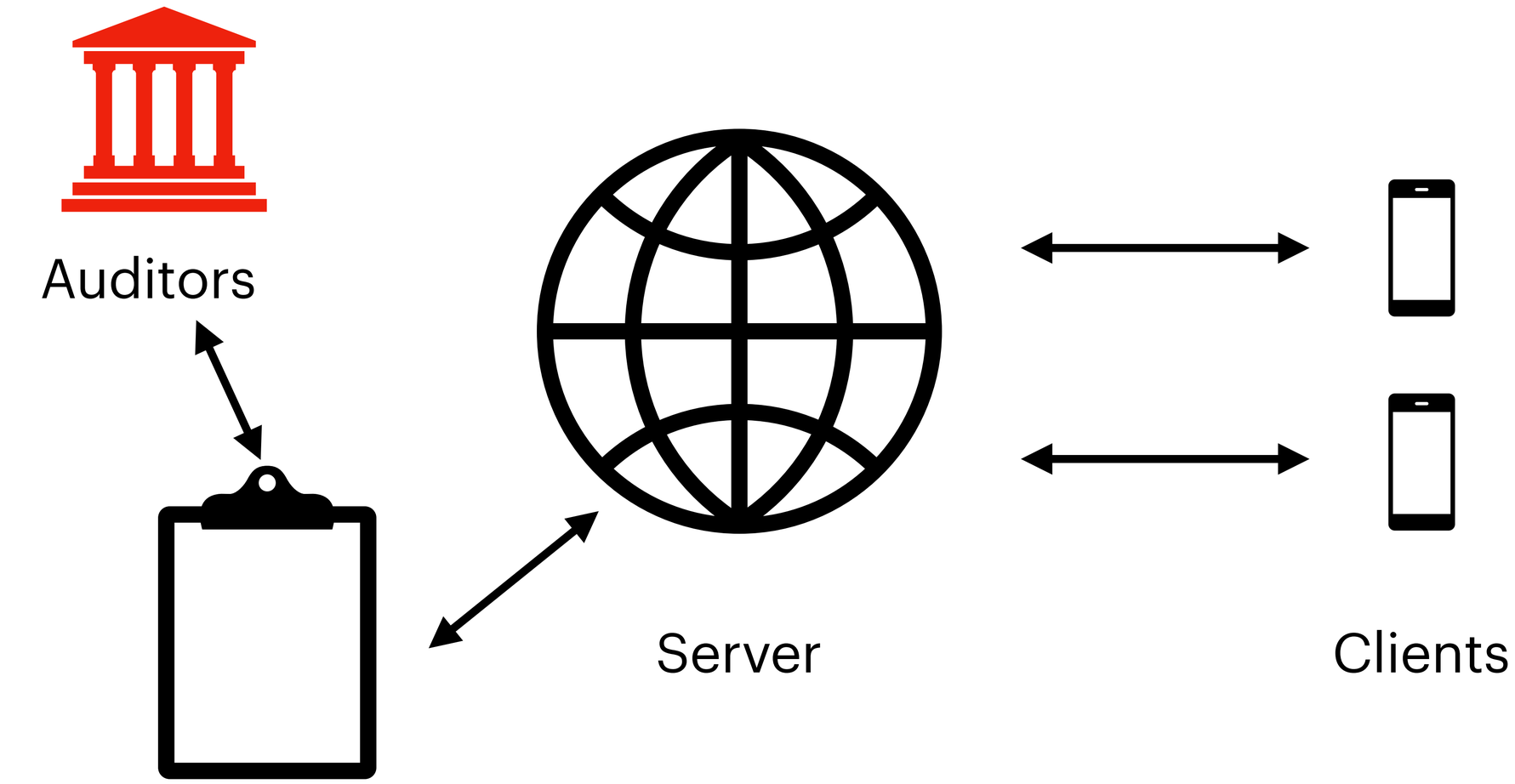
Illustration of Invariance Proof

Invariance proof, e.g. versions are incremented for changed values




What Are
the
Research
Questions?

Who’s the auditor here?



Who are the auditors?

- People who audit theinvariance proofs!
  - If invariance proof is **COSTLY** ==> auditors are servers.
  - If invariance proof is cheap ==> auditors can be everyone.

Who are the auditors?

- People who audit the proof invariance!
  - If invariance proof is **COSTLY** ==> auditors are servers.
  - If invariance proof is cheap ==> auditors can be everyone.

The real question: How expensive are invariance proofs to verify?

Audit proofs in Merkle trees

- Proofs are linear in the number of updates ==>
  - 200MB for WhatsApp every 5min
  - Such proofs are expensive to get succinct with SNARKs
- Hekaton ==> distributed SNARK on a cluster of 4000 machines
  - Throughput of ~10 updates/s ==> SHA is expensive in the circuit

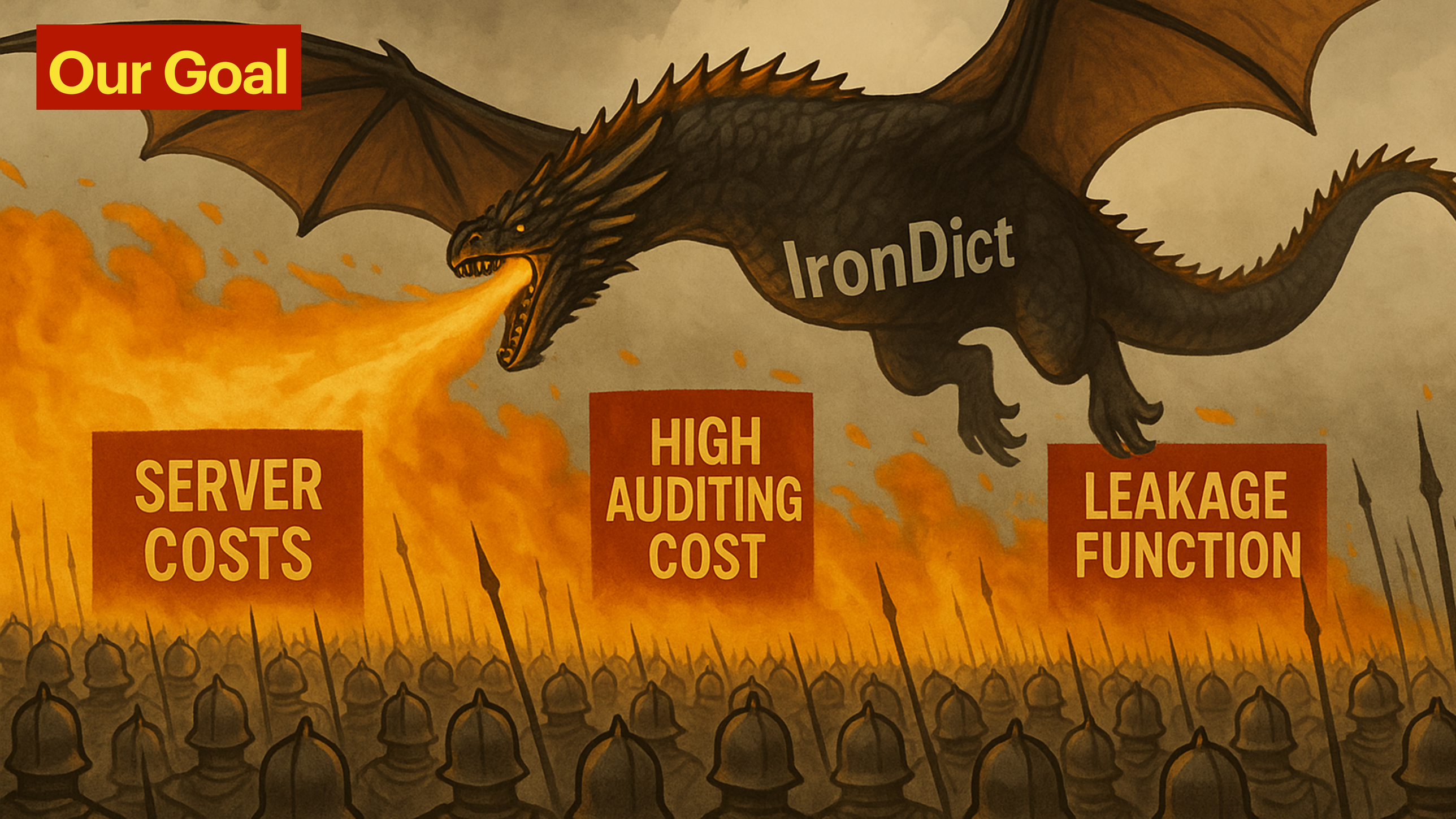
Audit proofs in Merkle trees

- Proofs are linear in the number of updates ==>
  - 200MB for WhatsApp every 5min for a throughput of 200/s
- Such proofs are expensive to get succinct with SNARKs
- Hekaton ==> distributed SNARK on a cluster of 4000 machines
  - Throughput of ~10 updates/s ==> SHA is expensive in the circuit

Can we make auditing cheap enough for everyday users?

Merkle Trees Don’t Give Perfect Privacy

- Privacy in crucial in TP e.g. when used in secure messaging apps:
  - Pattern of changing keys
- Tree construction do not have perfect privacy ==> leak function
  - SEEMLESS ==> each lookup reveal the last time a key was updated


Our Goal
IronDict
SERVER COSTS
HIGH AUDITING COST
LEAKAGE FUNCTION


D
IRONDICT

Our Contributions

- Generic construction of TD based a polynomial commitment scheme
- Perfect privacy ==> first scheme ever!
- Efficient support of fast-forwarding ==> 1000x better than VeRSA!
- Highly efficient: For a dictionary of size **2 billion**
  - Self-auditing ==> proof of size **~10KB** and verified under **125ms**
  - Efficient lookup and proof of consistency both for the server and client
  - Supporting more than **1000 updates/s**
  - Low memory/computation overhead for the server

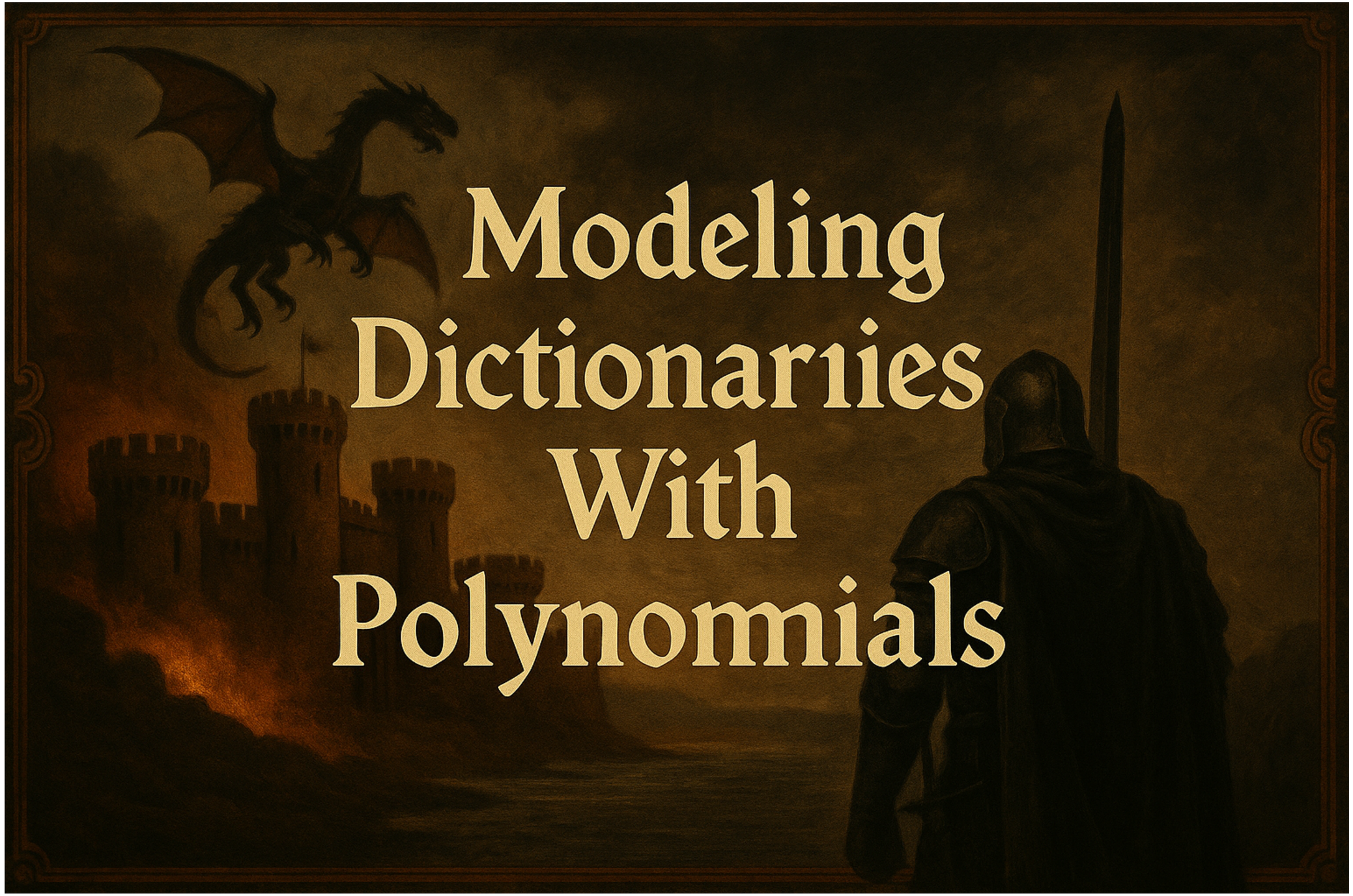
Preliminary

- **Multilinear polynomial:**
  - multivariate polynomial where the degree of each variable is at most 1.
  - e.g. $f(x_1, x_2, x_3) = x_1x_2 + 4x_2x_3 + 5x_3 + x_1x_2x_3$
  - Evaluations of the boolean hypercube give us the polynomial e.g.

$$f(0,0,0) = y_{0,0,0}\,, f(0,0,1) = y_{0,0,1},\ \ldots, f(1,1,1) = y_{1,1,1}$$

Preliminary

- Polynomial commitment scheme (PCS):
  - Backbone of SNARKs ==> so much research on it!
- PCS allows someone to commit to a polynomial f(x) and later prove its evaluation at point x_0, without revealing the whole polynomial.
  - Succinctness: commitment is small.
  - Efficient: verifier time is sublinear in the polynomial size.
  - Binding: the prover cannot change the polynomial later.


Modeling
Dictionaries
With
Polynomials

Model Dictionary With Polynomials

- Consider a very specific type of dictionary:

$$\text{Dict} = [(\ell_i, v_i)]_{i\in[n]} \; : \; \begin{cases} \ell_i \in \{0,1\}^{\log_2 N} \\ v_i \in \mathbb{F}\backslash\{0\} \end{cases}$$

- We can model it with a multilinear polynomial:

$$p(\mathbf{X}) = \begin{cases} v_i & : \mathbf{X} = \ell_i \\ 0 & : \mathbf{X} \in \{0,1\}^{\log_2 N} \backslash \{\ell_i\}_{i\in[m]} \end{cases}$$

Modelling General Dictionaries

- Dictionaries we are interested in:

$$\text{Dict} = [(\ell_i, v_i)]_{i \in [n]} \; : \; \begin{cases} \ell_i \in \{0,1\}^* \\ v_i \in \{0,1\}^* \end{cases}$$

One to One Map from String to Field

- Given hash function collision-resistant and oneway H ==>
  - One-to-one mapping from $\mathbb{F}\backslash\{0\}$ to arbitrarily strings

$$H : \{0,1\}^* \rightarrow \mathbb{F}\backslash\{0\} \ : \ 2^{128} \leq |\mathbb{F}|$$

$$\text{Dict} : \{0,1\}^{\log_2 N} \rightarrow \mathbb{F}\backslash\{0\} \equiv \text{Dict} : \{0,1\}^{\log_2 N} \rightarrow \{0,1\}^*$$

Same Strategy Doesn’t Work Again!

- We can define a similar hash function H, but this time it’s not bijective!!!!

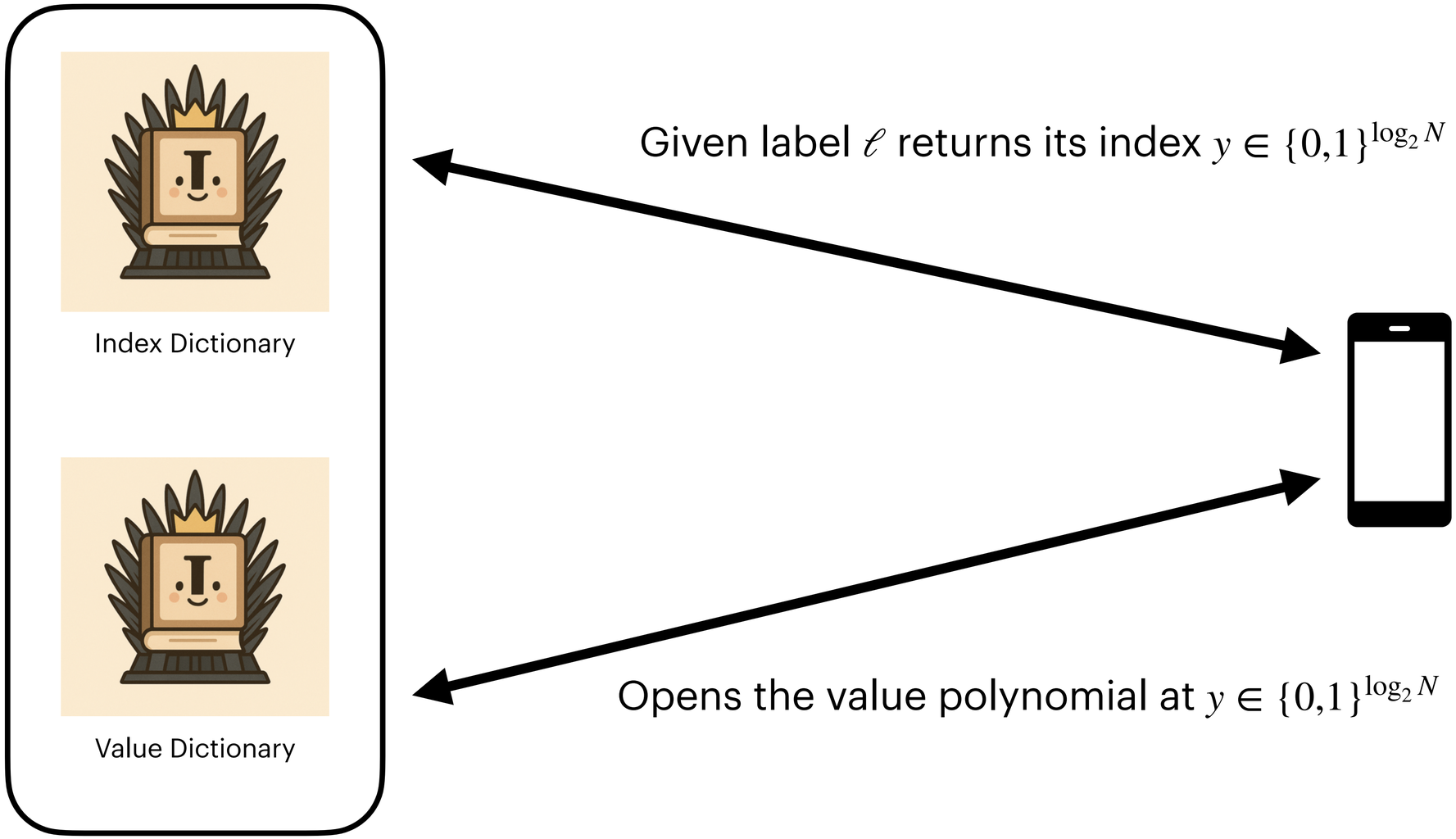
$$H : \{0,1\}^* \rightarrow \{0,1\}^{\log_2 N}$$

Our Strategy: Keep Two Polynomials

- Keep two different dictionaries:

$$\begin{cases} \text{Index} : \{0,1\}^* \to \{0,1\}^{\log_2 N} \\ \text{Value} : \{0,1\}^{\log_2 N} \to \{0,1\}^* \end{cases} : \text{Dict} : \{0,1\}^* \to \{0,1\}^* \equiv (\text{Index}, \text{Value})$$

High Level Description of The Model



What We Want From These Dictionaries

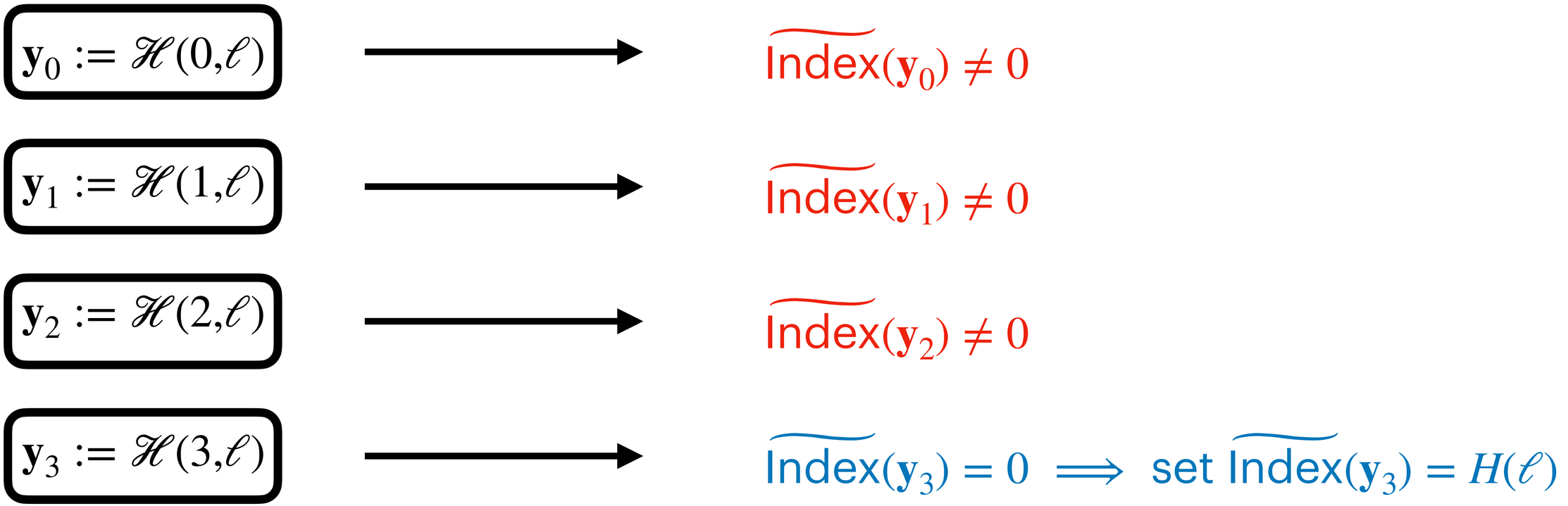
- Index Dictionary: Maps label to an assigned index
  - Append only
- Value Dictionary: Maps indexes to values
  - When values changes, this dictionary changes

Index Dictionary: How to assign indexes

- We cannot assign indexes with one hash evaluation ==> open addressing
- Use a hash function repeatedly until we find a vacant spot.

Open addressing

- Assume $\mathscr{H}$ be a hash function from arbitrary strings to the boolean hypercube.

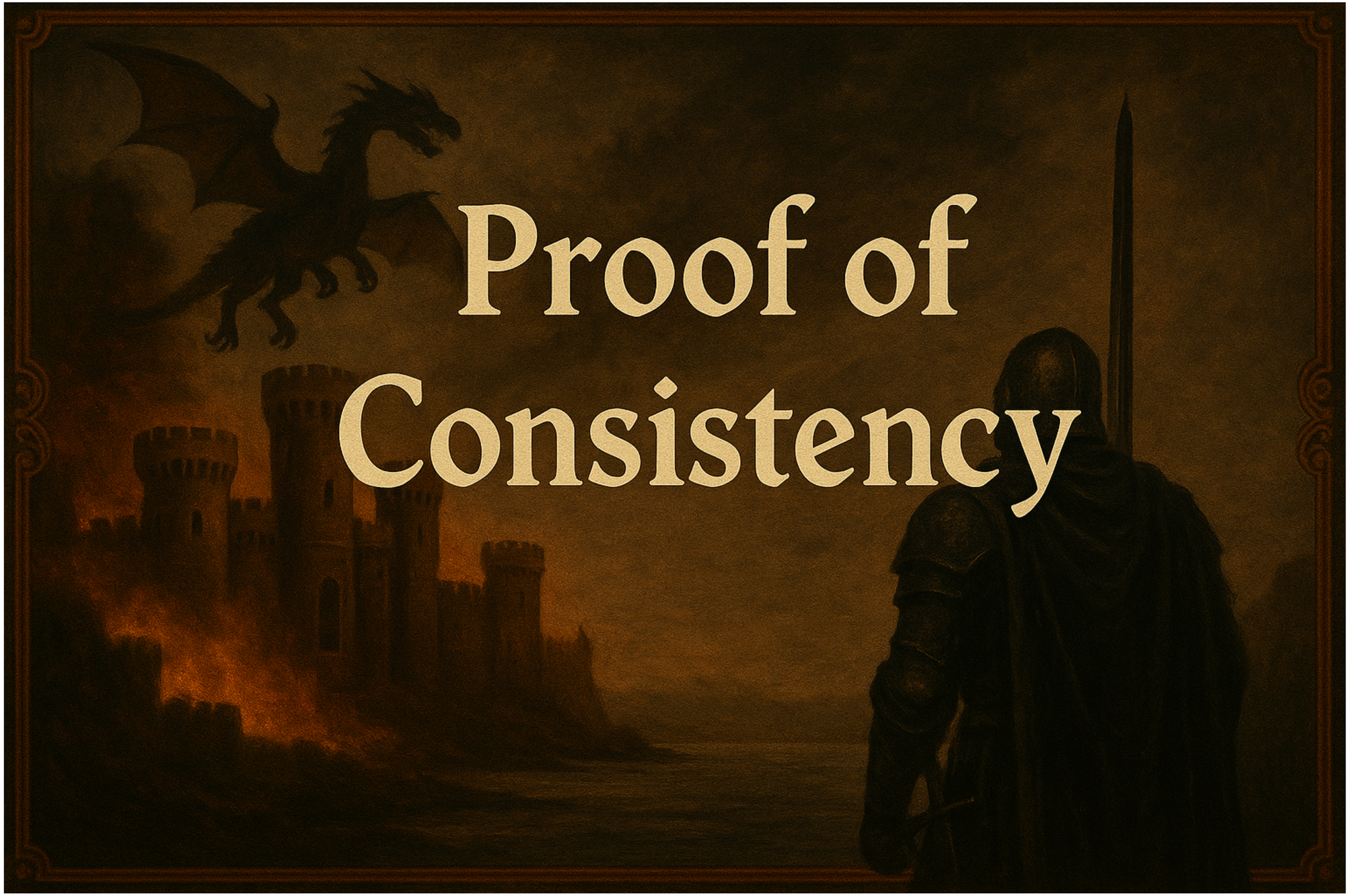


Open addressing

- We require size over provisioning!
- Given a 4x over provisioning, on average it takes 1.3 point to find an empty spot!

Value Dictionary

- We hold no constraint on the value dictionary, e.g. the server can update it as it wishes


Proof of
Consistency

How to Check Consistency?

$$\widetilde{\Delta_i} = \widetilde{\text{Value}}_{i+1} - \widetilde{\text{Value}}_i \implies \widetilde{\text{Value}}_n = \widetilde{\text{Value}}_m + \sum_{m \le i < n} \widetilde{\Delta_i}$$

- Observation: if $\widetilde{\text{Value}}_n(\mathbf{x}) = \widetilde{\text{Value}}_m(\mathbf{x})$ and a ghost key takes place it means the difference polynomials have cancelled each other.

- Use randomness so they cannot cancel out!!!

$$\widetilde{\text{Rand}}_n = \widetilde{\text{Value}}_0 + \sum_{i<n} r_i \cdot \widetilde{\Delta}_i \implies \widetilde{\text{Rand}}_n(\mathbf{y}) = \widetilde{\text{Rand}}_m(\mathbf{y})$$

$$\iff \text{overwelming probability } \widetilde{\text{Value}}_n(\mathbf{y}) = \widetilde{\text{Value}}_{n-1}(\mathbf{y}) = \ldots = \widetilde{\text{Value}}_m(\mathbf{y})$$

- r_i are random values, e.g. derived from Fiat-Shamir

Index Dictionary

Value Dictionary

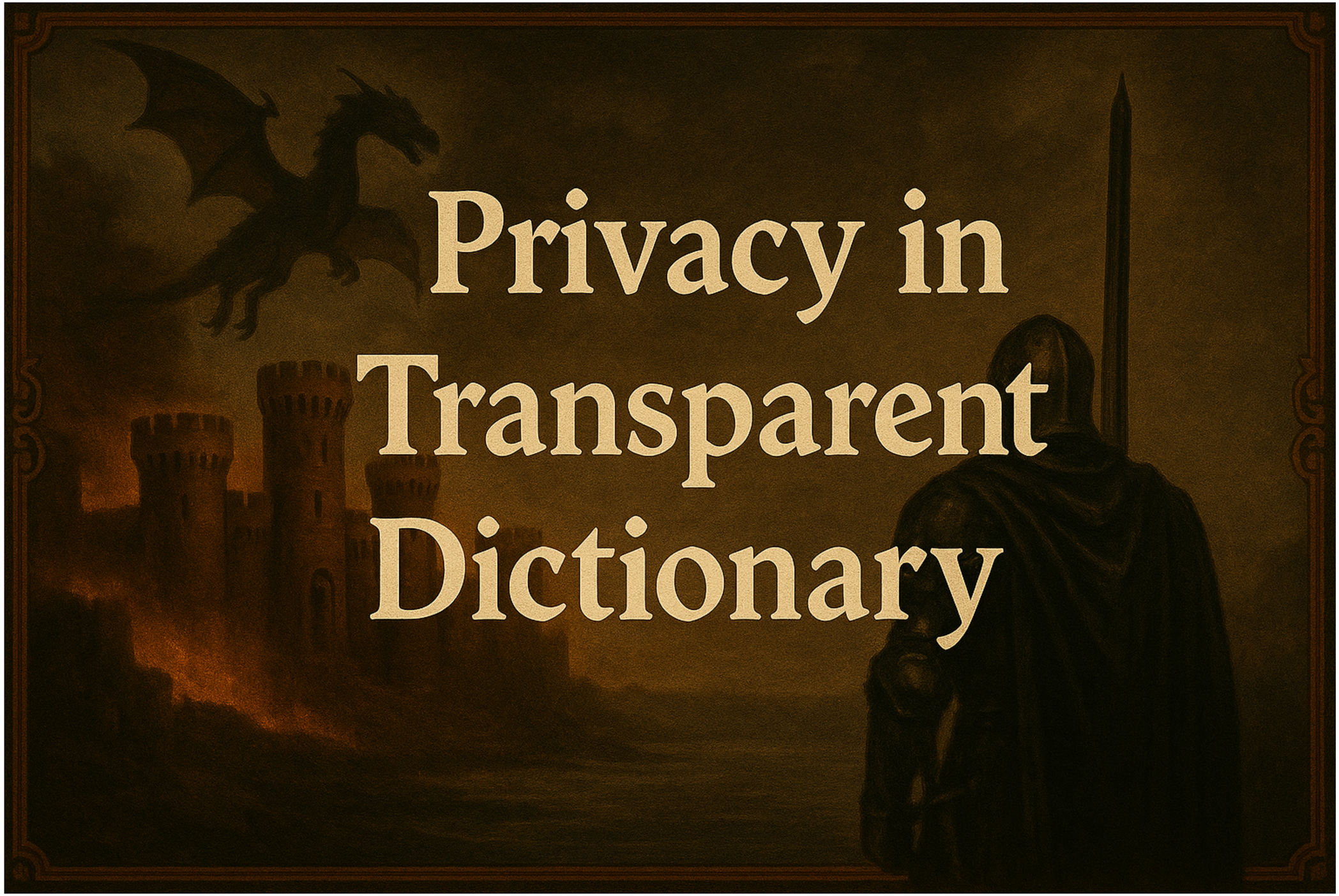
Rand Dictionary

Maps usernames to points on the boolean hypercube

Maps points on the boolean hypercube to public keys

Maps points on the boolean hypercube

to rand combination of values


Privacy in
Transparent
Dictionary

Privacy in IronDict

- Server reveals: sumcheck proof, polynomial commitment and opening
- **MAKE ALL ZK** ==> only reveal what they’re supposed to


Concrete
Instantiation

Initiating The System With KZH

- KZH has sublinear opening time (**FREE** opening for Boolean points)
- Supports efficient zk transformation
- KZH has constant commitment size

Implementation (1 Billion users)

	zk-IronDict (KZH-k)	WhatsApp (SEEMLESS + Parakeet)
Lookup proof / verification	**20KB / 29ms**	**2-4 KB / ~0.5ms**
Lookup time for server	**33ms**	**1ms**
Throughput	**1000**	**1000**
Audit proof / verification	**8KB / 35ms**	**1200MB / 11s**

Fast-forwarding auditing

- Auditor being offline for 1000 epochs has to check 1000 proofs!!
  - IVC ==> 1000x better than VeRSA for the server
  - Checkpoints ==> Only check a sublinear number of epochs.


Thank
You