Course Announcements

- Project
  - Project due Wednesday 4/22
  - Please send a private Piazza post to the TA/grader overseeing your project
  - Will post a grading rubric for the project later today
- Assignments
  - No homework for the next 2 weeks
  - Reading: Cryptographically Protected Database Search

Lecture 20: Protecting Passwords and Databases in Use

1. Zero knowledge proofs
2. Protecting password checks
3. Protecting database search

Oblivious computing

Images: Facebook, Wikipedia

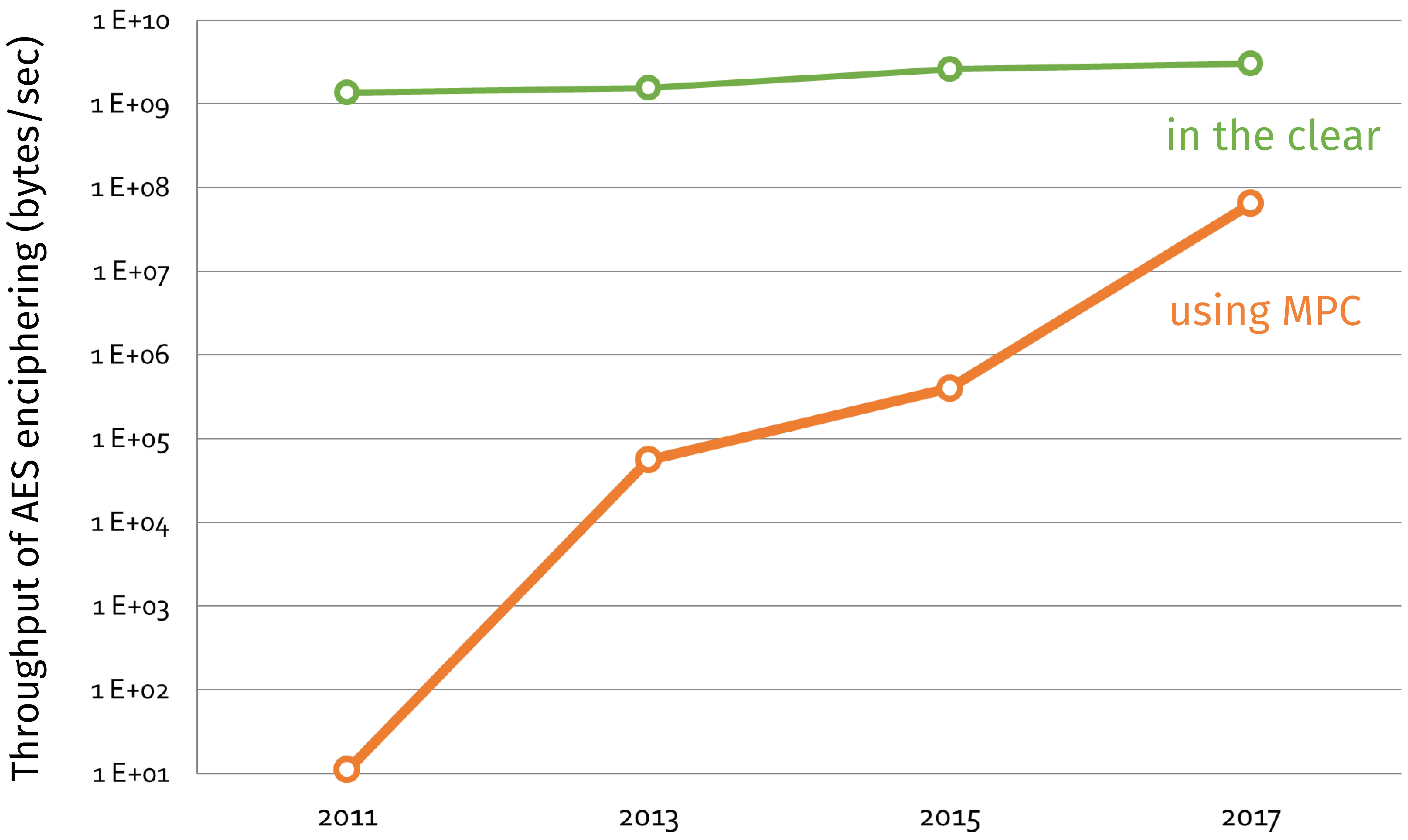
Review: secure computation

- Suppose *m* people have sensitive data $x_1, x_2, \ldots, x_m$
- Want to outsource this data to multiple compute parties $P_1, P_2, \ldots, P_n$
- Parties engage in computing a publicly-known function *f*

$$y = f(x_1, x_2, \ldots, x_m)$$

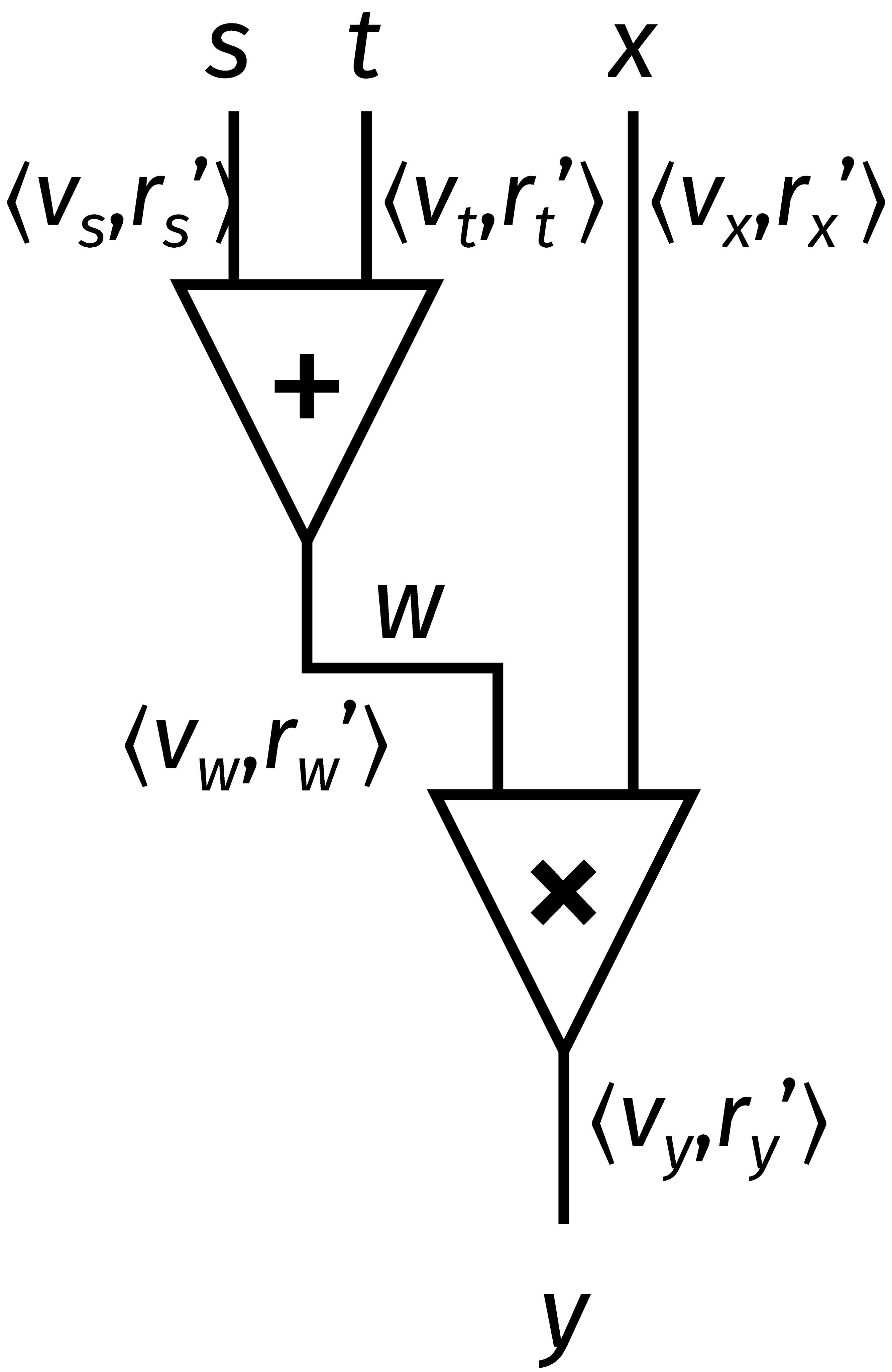
- Assume that at most *t* of the *n* parties are adversarial
  - They might collectively be acting as a passive Eve or an active Mallory
- Want to ensure: nothing is revealed about the inputs beyond what can be inferred from the output *y* (note: for some *f*, inference is bad!)

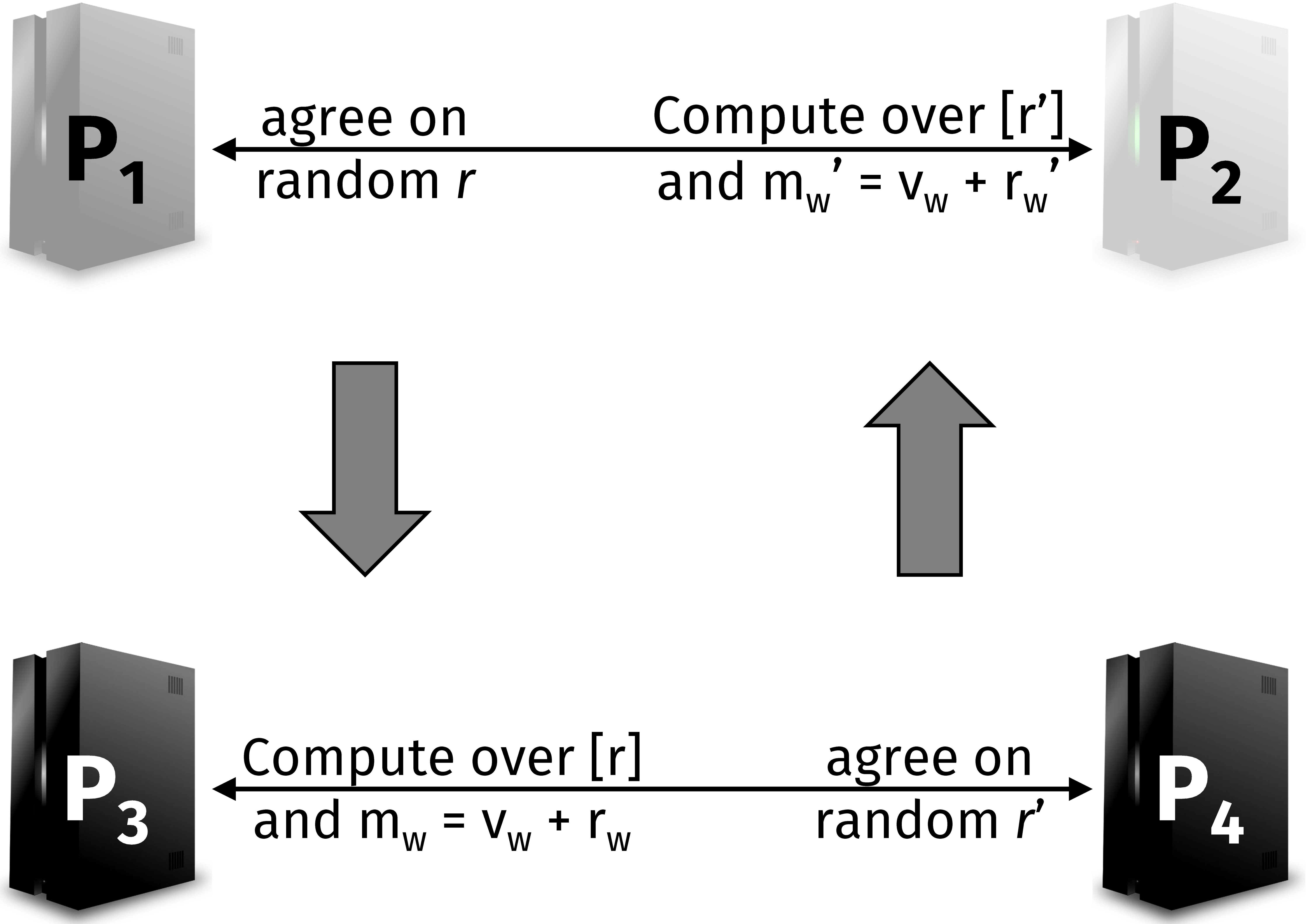
Performance of Generic MPC



1. Zero knowledge proofs

Review: 4-party secure computation vs Mallory





Special case: zero-knowledge proofs

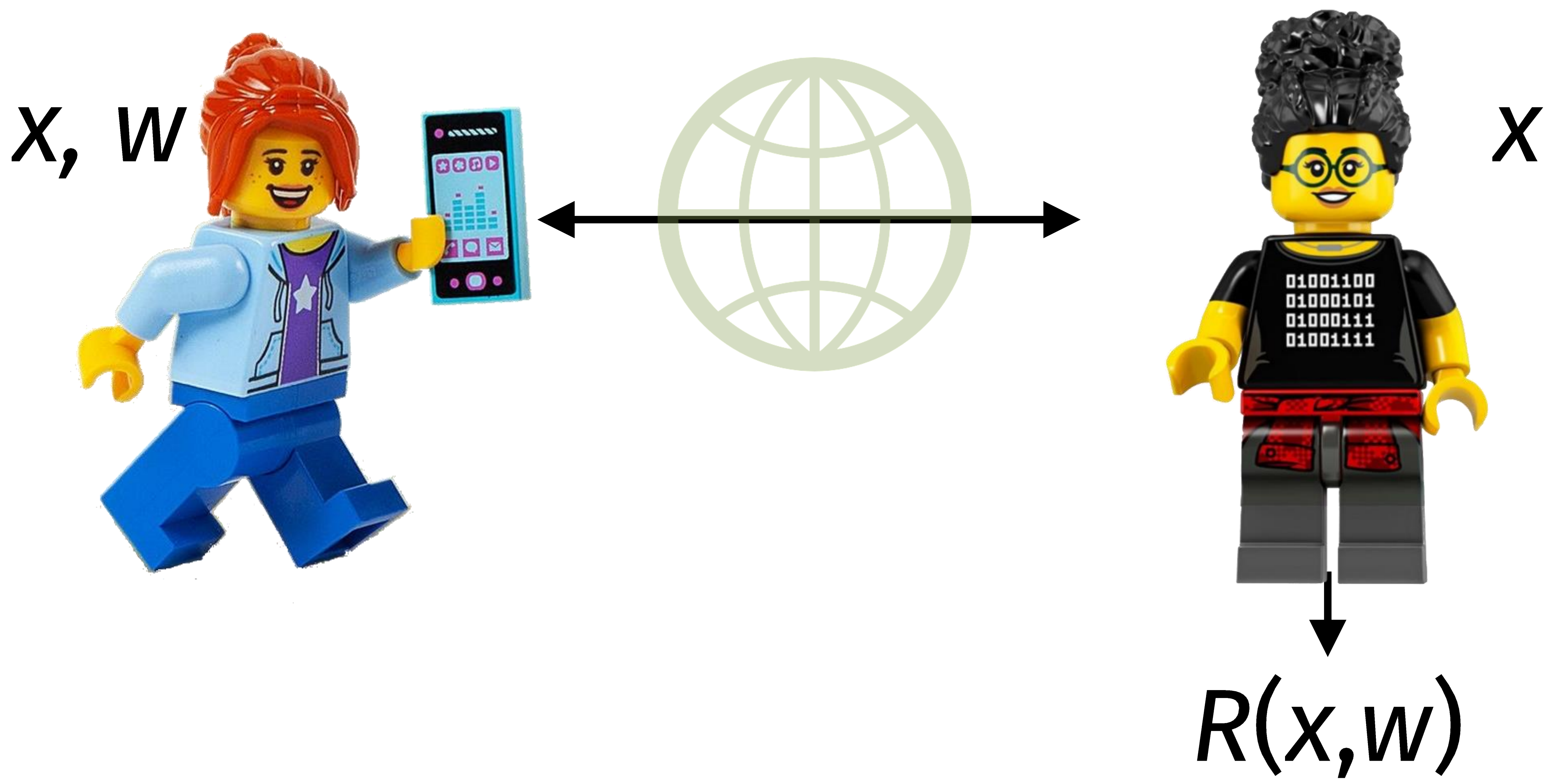
- Consider two parties: a prover P and a verifier V
- There is a public statement x that prover claims is in NP language L
- Prover knows a witness w such that R(x, w) = True

Verify, don't trust

- P wants to convince V that $x \in L$

Verify, don't trust

- P wants to convince V that $x \in L$
  - P doesn't trust V with her data; she doesn't want to reveal w

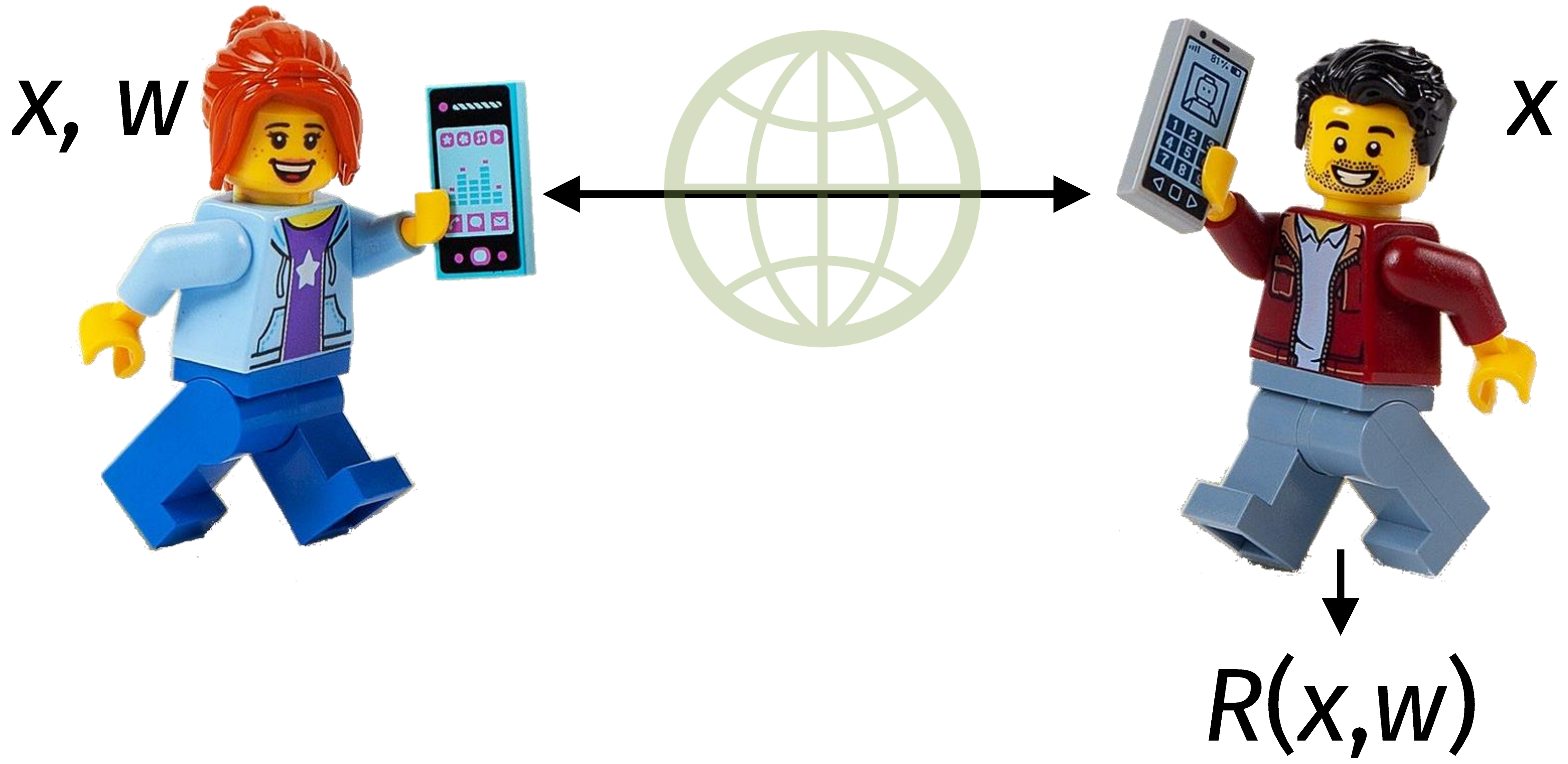


Verify, don't trust

- P wants to convince V that $x \in L$
  - P doesn't trust V with her data; she doesn't want to reveal w
  - V doesn't trust P to tell the truth

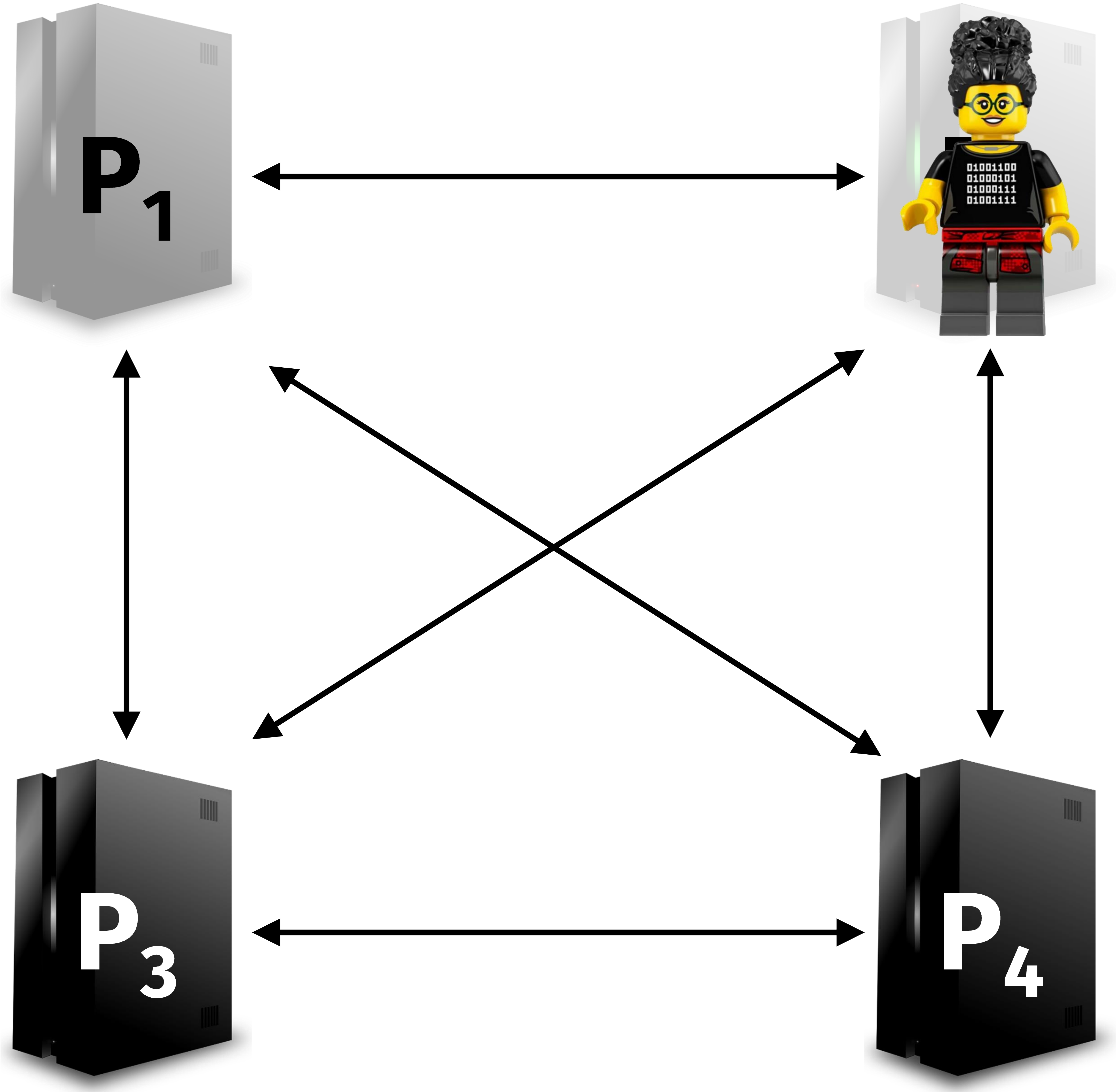
Verify, don't trust

- P wants to convince V that $x \in L$
  - Privacy: P doesn't trust V with her data; she doesn't want to reveal w
  - Correctness: V doesn't trust P to tell the truth, must be convinced
- Breaking the logjam: P and V can compute R(x, w) via 2-party MPC



Zero knowledge via "MPC in the head"

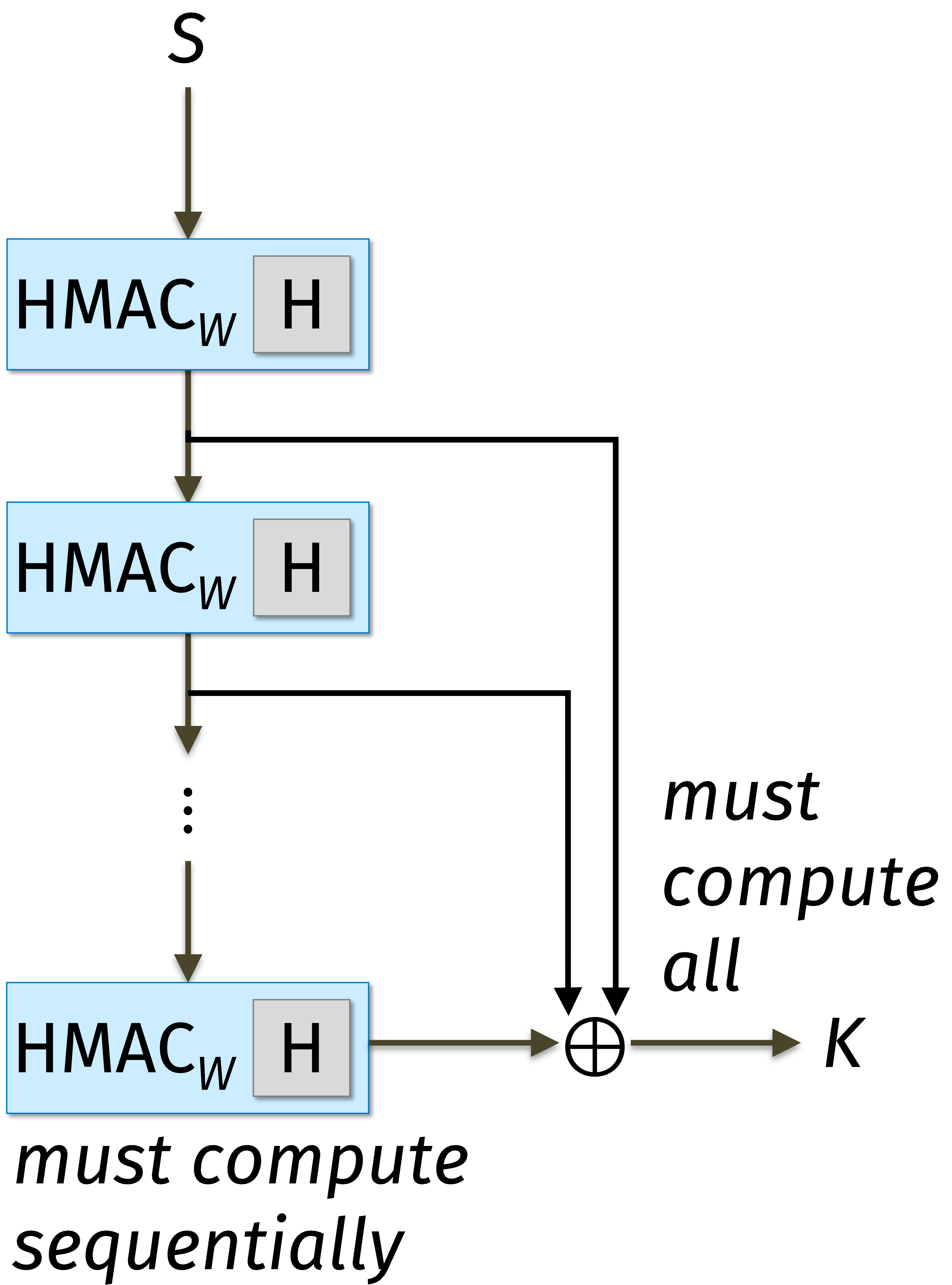
- There is another way
  - Prover securely computes R(x,w)
  - Prover acts as *all* compute parties
- Let the verifier choose *t* parties and receive their complete state
  - Privacy: observing the view of t parties gives V no information
  - Correctness: if P deviates from the protocol, Pr[V catches] = t/n



2. Protecting password checks

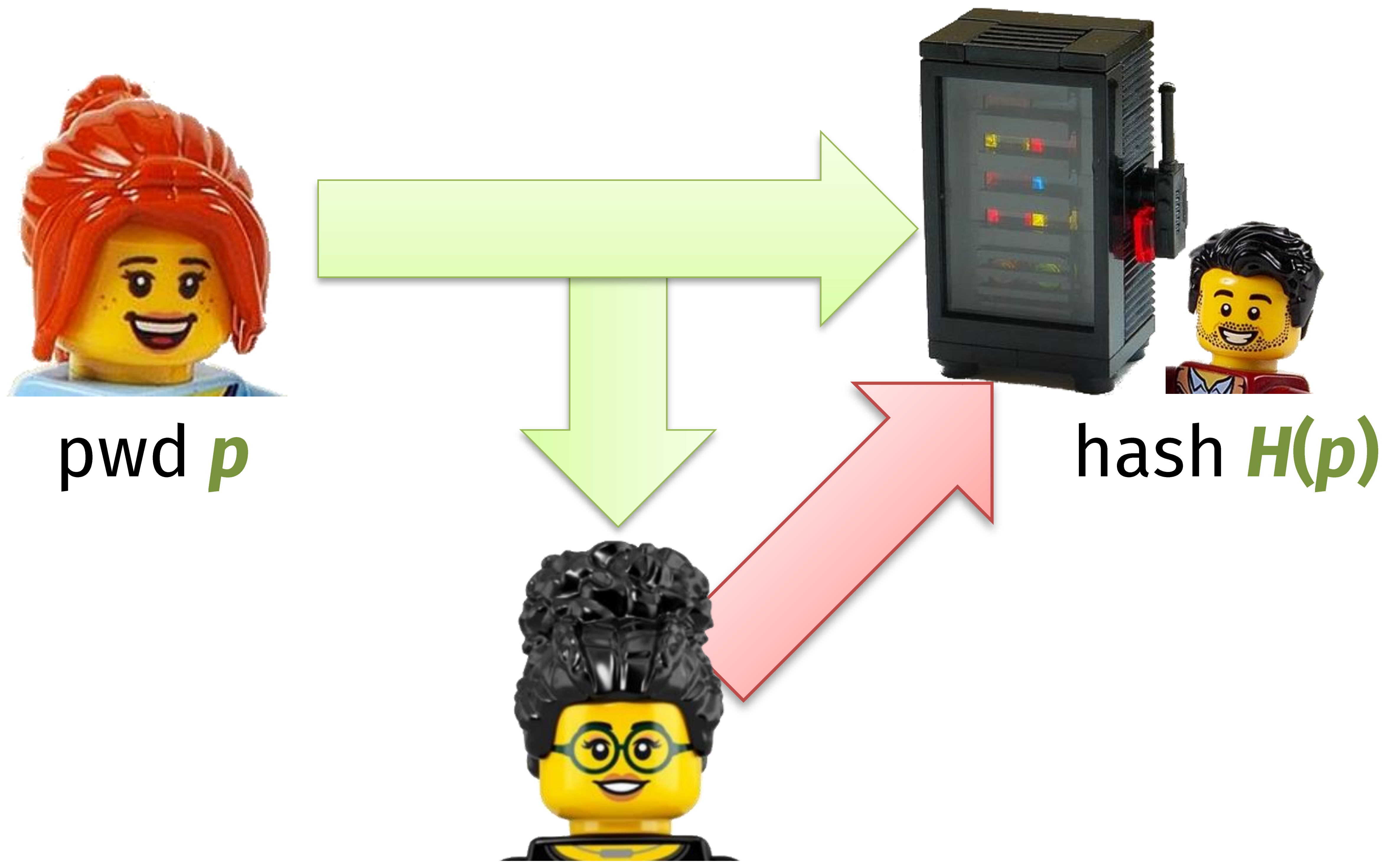
Review: Password-based key derivation

- Approach: derive key from a moderately-strong password
- Threat: powerful attacker who…
  - Obtains your personal phone or organization's /etc/passwd file
  - Brute forces many passwords
- Best option for non-interactive login, ensures that the device itself never stores the password



Offline vs online dictionary attack

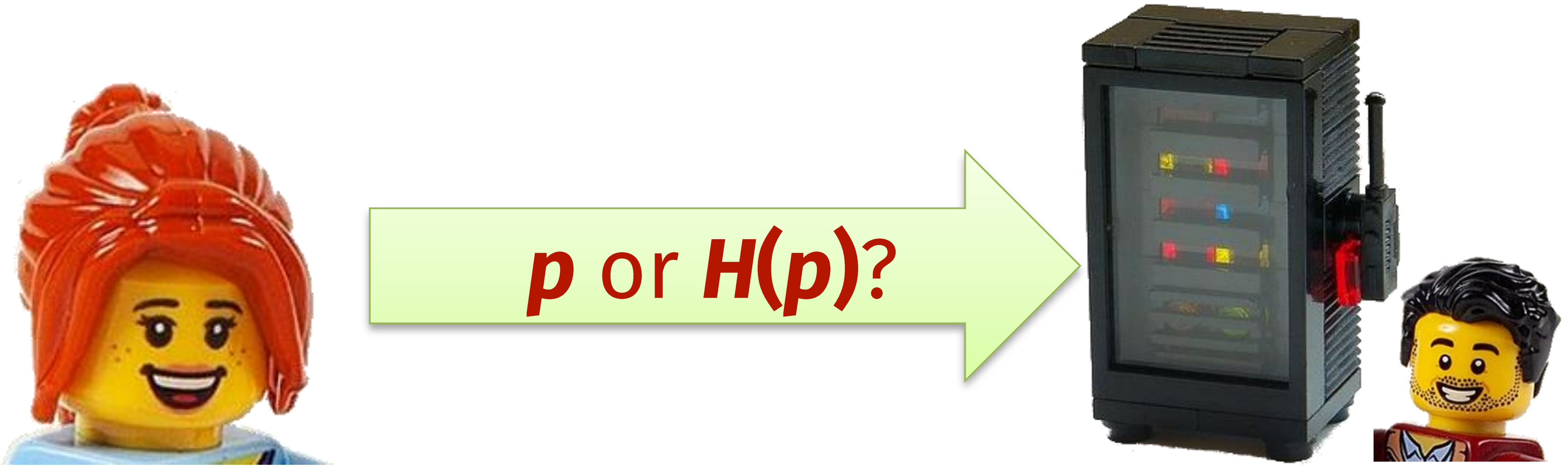
- PBKDF2 vulnerable to an *offline* dictionary attack:
  - Mallory learns salt, count, pbkdf2(pwd, salt, count)
  - Mallory guesses many passwords on her own cluster in parallel
  - Mallory makes only 1 password guess on the target device
- *Online* dictionary attack: Mallory must check guesses with server
  - Opportunity for rate limiting



Password dilemma

Alice wants to authenticate to bob.com. Does she send p or $H(p)$?

- If Alice sends $H(p)$, then the stored hashed database is very sensitive
- If Alice sends p, then transmission is very sensitive (<- done in practice)



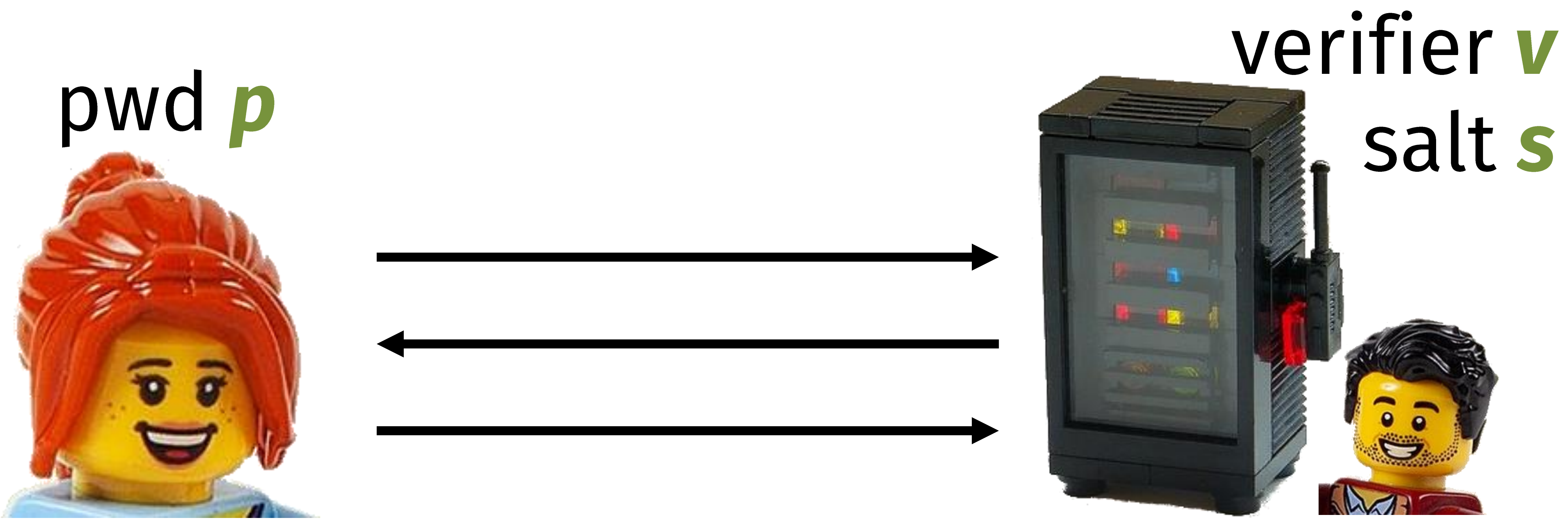
pwd p

database of $H(p)$, where H = pbkdf2, etc

Objective: verify passwords *without* seeing them!

- Alice knows a password *p* but doesn't want to share it with anyone, even bob.com

- If bob.com never sees the password then he cannot accidentally store it



KrebsonSecurity
In-depth security news and investigation

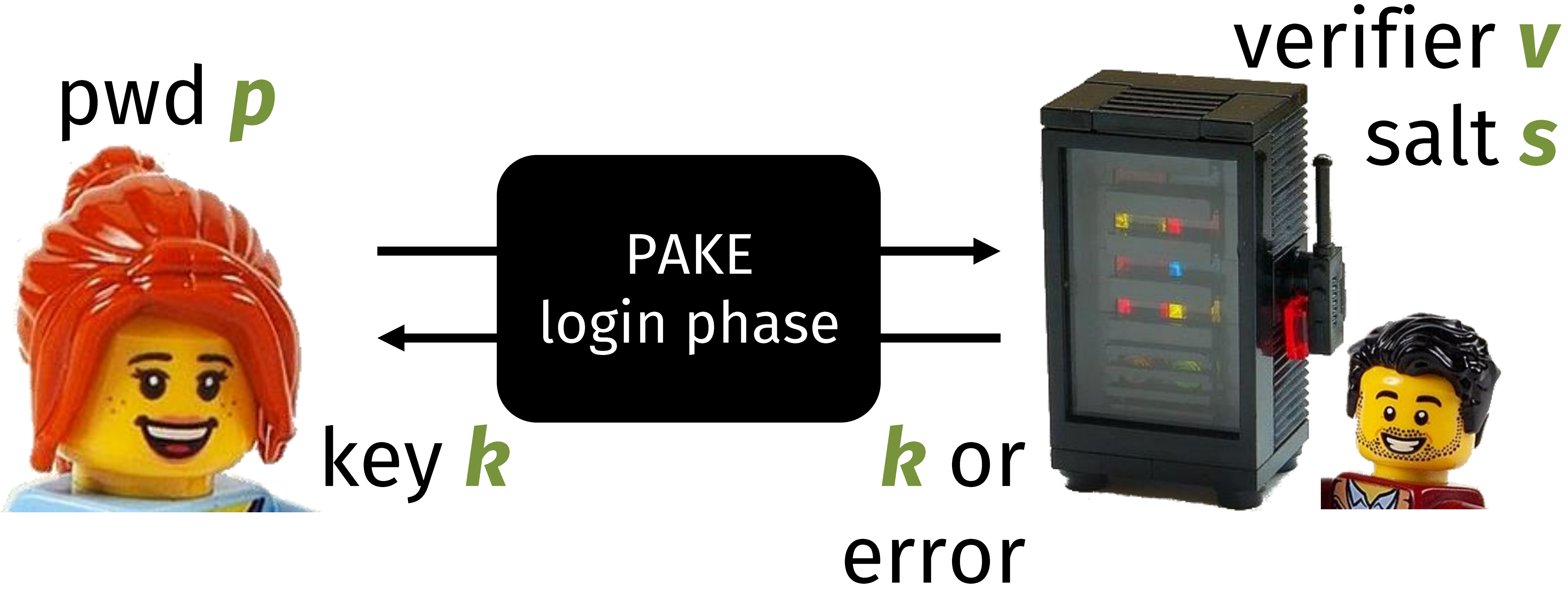
21 MAR 19

Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years

Hundreds of millions of **Facebook** users had their account passwords stored in plain text and searchable by thousands of Facebook employees — in some cases going back to 2012, KrebsOnSecurity has learned. Facebook says an ongoing investigation has so far found no indication that employees have abused access to this data.

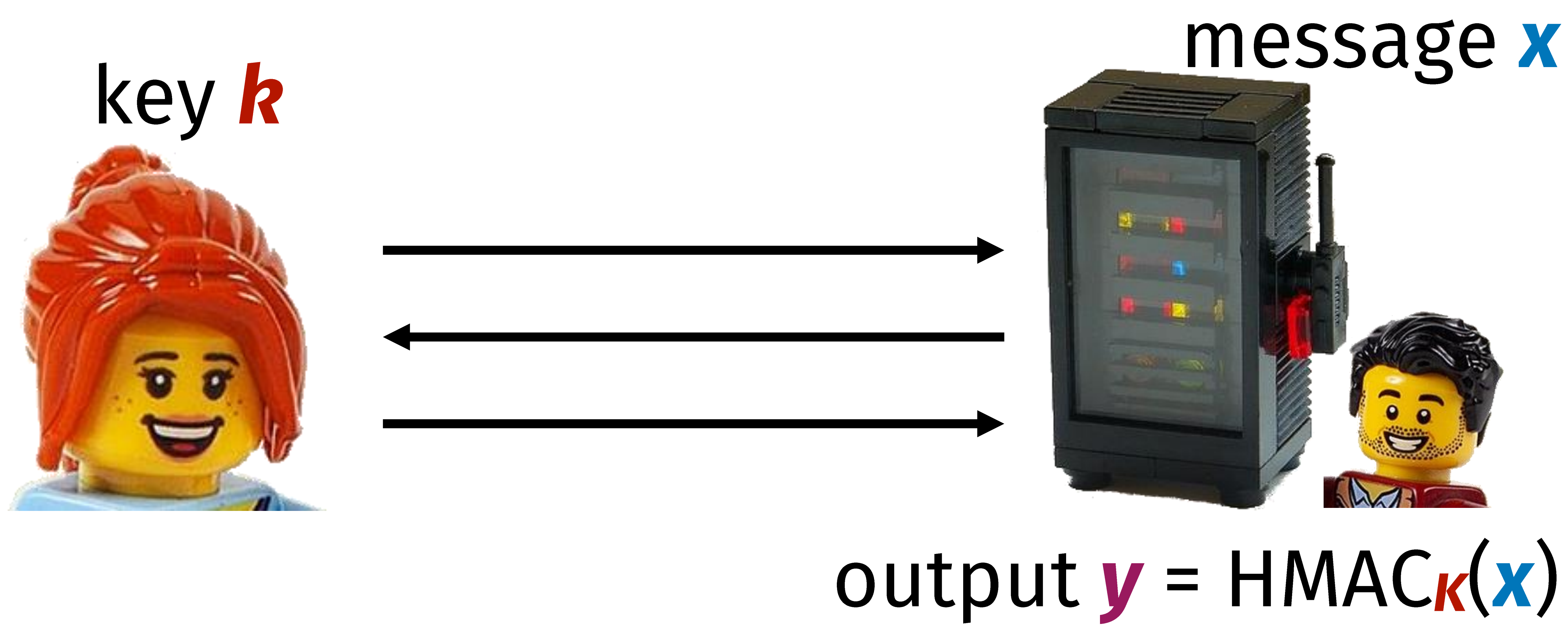
Password authenticated key exchange (PAKE)

- Signup phase
  - Alice provides Bob a verification string *v* to detect if he's talking to someone who knows *p*
- Login phase: the parties interact
  - If Bob is speaking to Alice, both parties get a shared key *k*
  - If Bob is speaking to Mallory, then Bob learns this fact
- Security goals: Bob never learns *p*, ideally Alice never learns *s*



Building block: Oblivious pseudorandom function

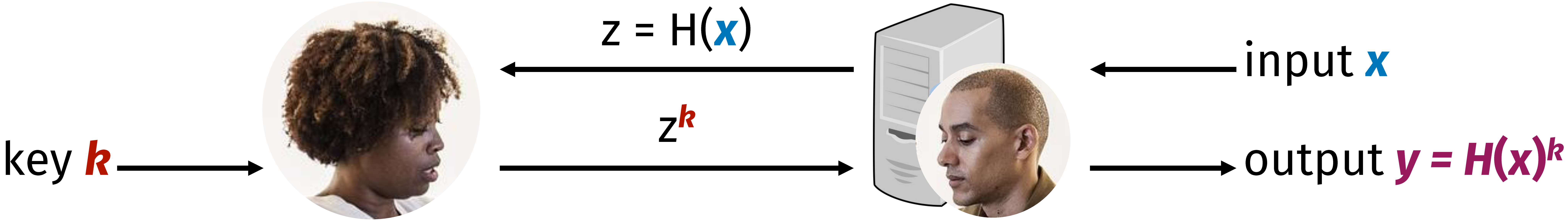
- Let's take a step back, address a different-looking question
  - Alice has a key, Bob has a message
  - Can we compute HMAC on this key & message without sharing them?
- Turns out the answer is **yes!**



Oblivious PRF -> Jointly compute PBKDF2

```
pbkdf2(string password, string salt, int count):
    string result = ‘’
    U₀ = S
    for(j = 1 to count):
        Uⱼ = hmac(password, Uⱼ₋₁)
        result = result ⊕ Uⱼ
    return result
```

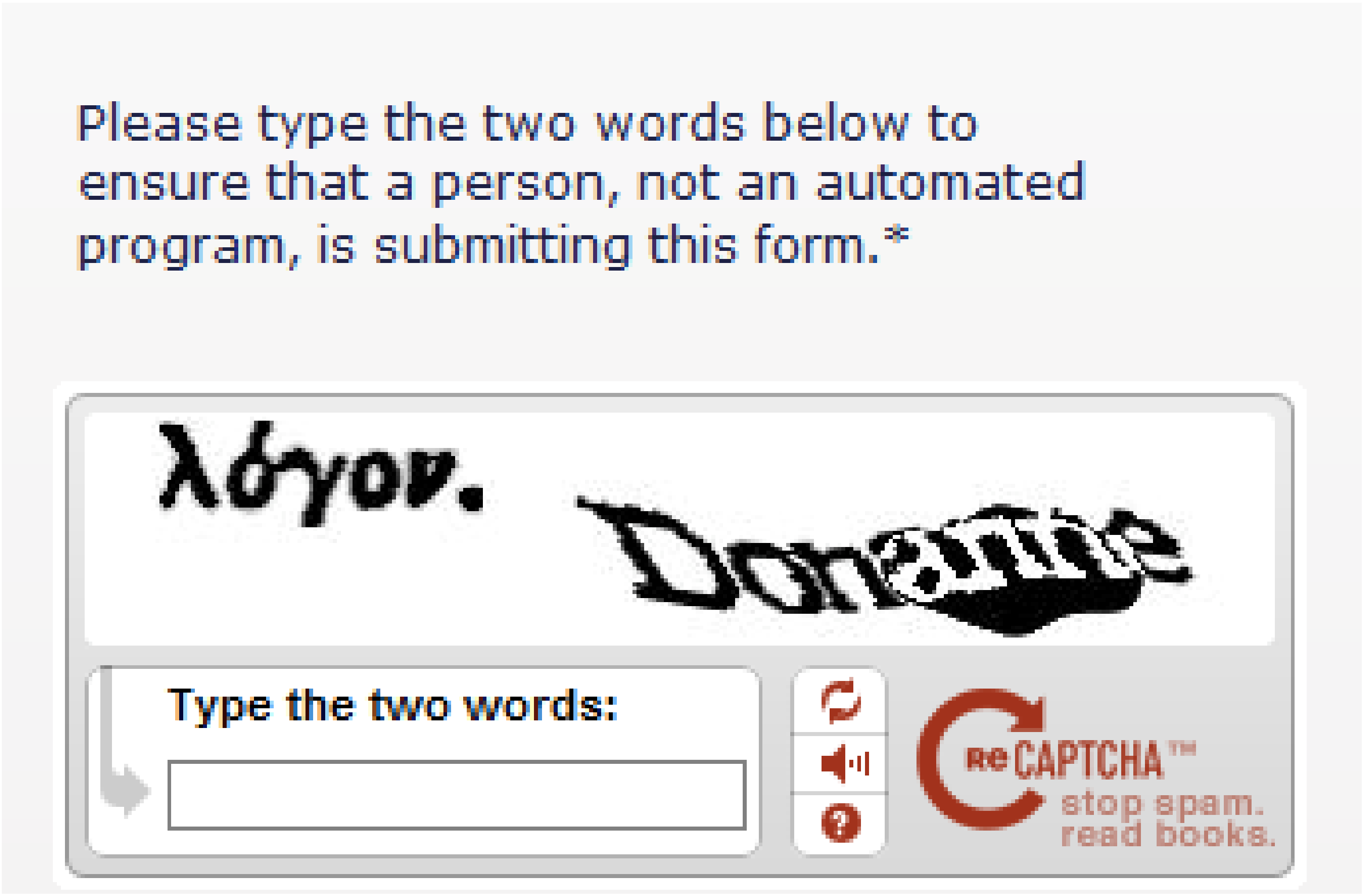
Constructing an Oblivious PRF (but not for HMAC)

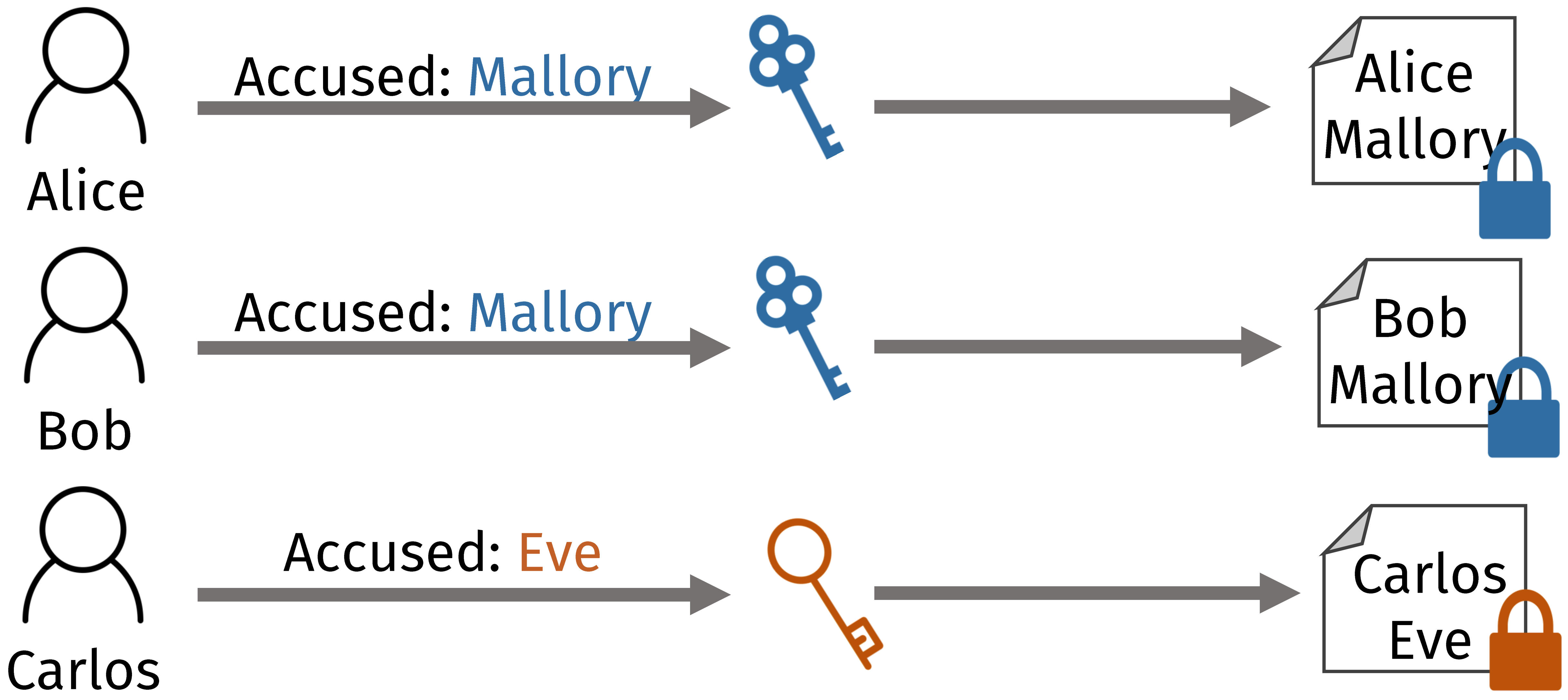


- $B_k(x) = H(x)^k$ is pseudorandom when calculated over a group where discrete logs are hard (e.g., modular arithmetic, elliptic curves)
  - Note: it requires ~milliseconds to compute, rather than ~nanoseconds of AES
- The above protocol is an oblivious method to calculate B
  - Hardness of discrete log prevents Bob from learning k from $H(x)^k$
  - Preimage-resistant hash function H prevents Alice from reversing z to learn x, if Bob chooses x at random (which might be okay in certain circumstances)

Applications of Oblivious PRFs

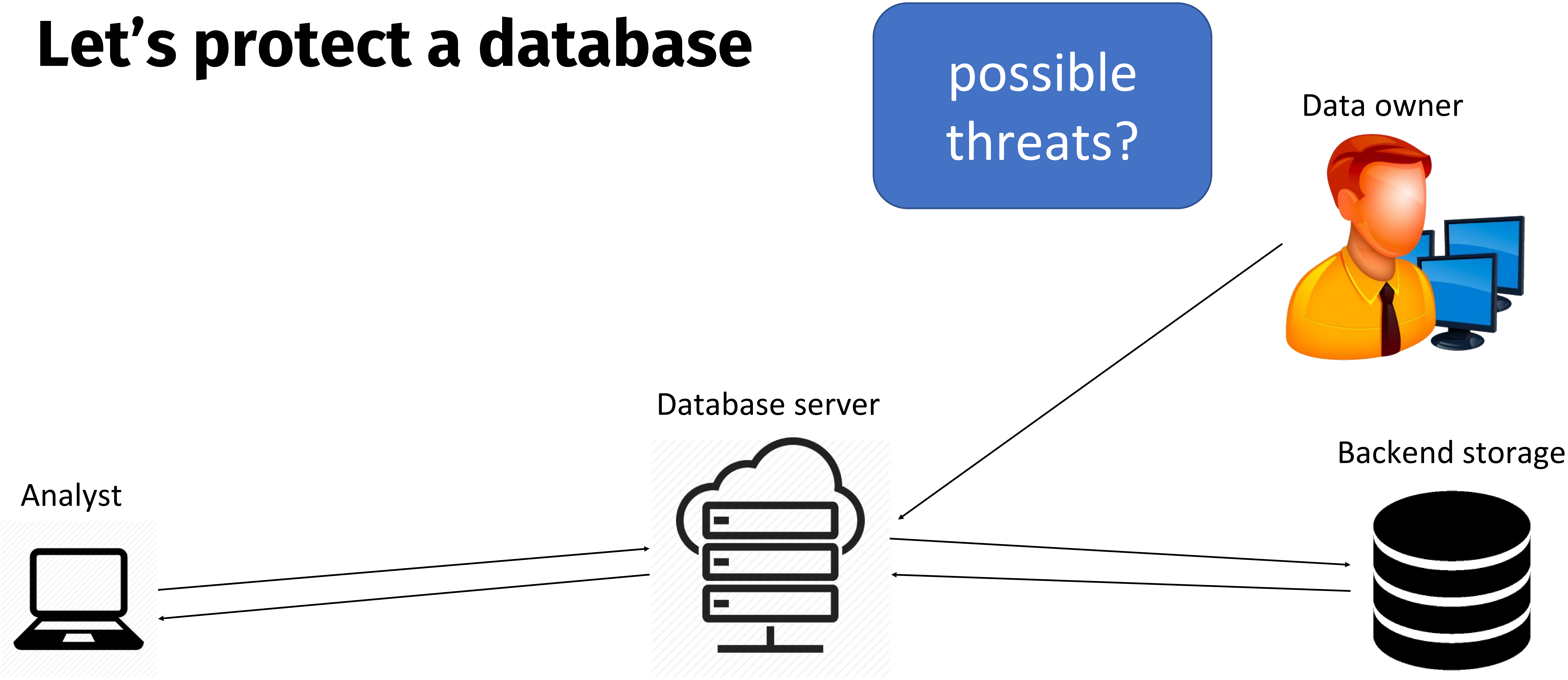
- Secure Remote Password (SRP) protocol: faster PAKE
- Cloudflare's Privacy Pass: reduces CAPTCHAs with Tor
- Callisto's MeToo system
  "Identifying information about a survivor and the accused can only be decrypted by a lawyer when at least 2 users name the same perpetrator"



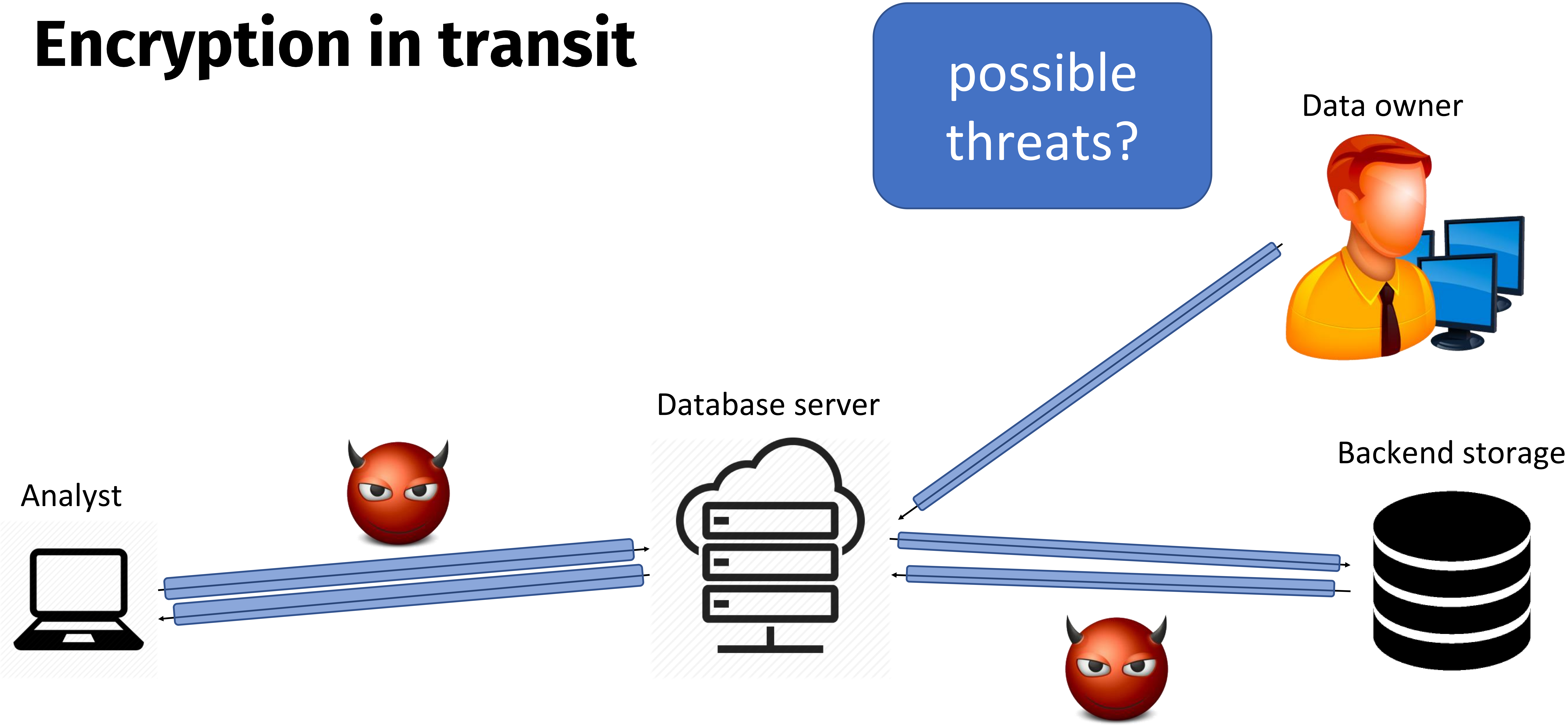


3. Protecting database search

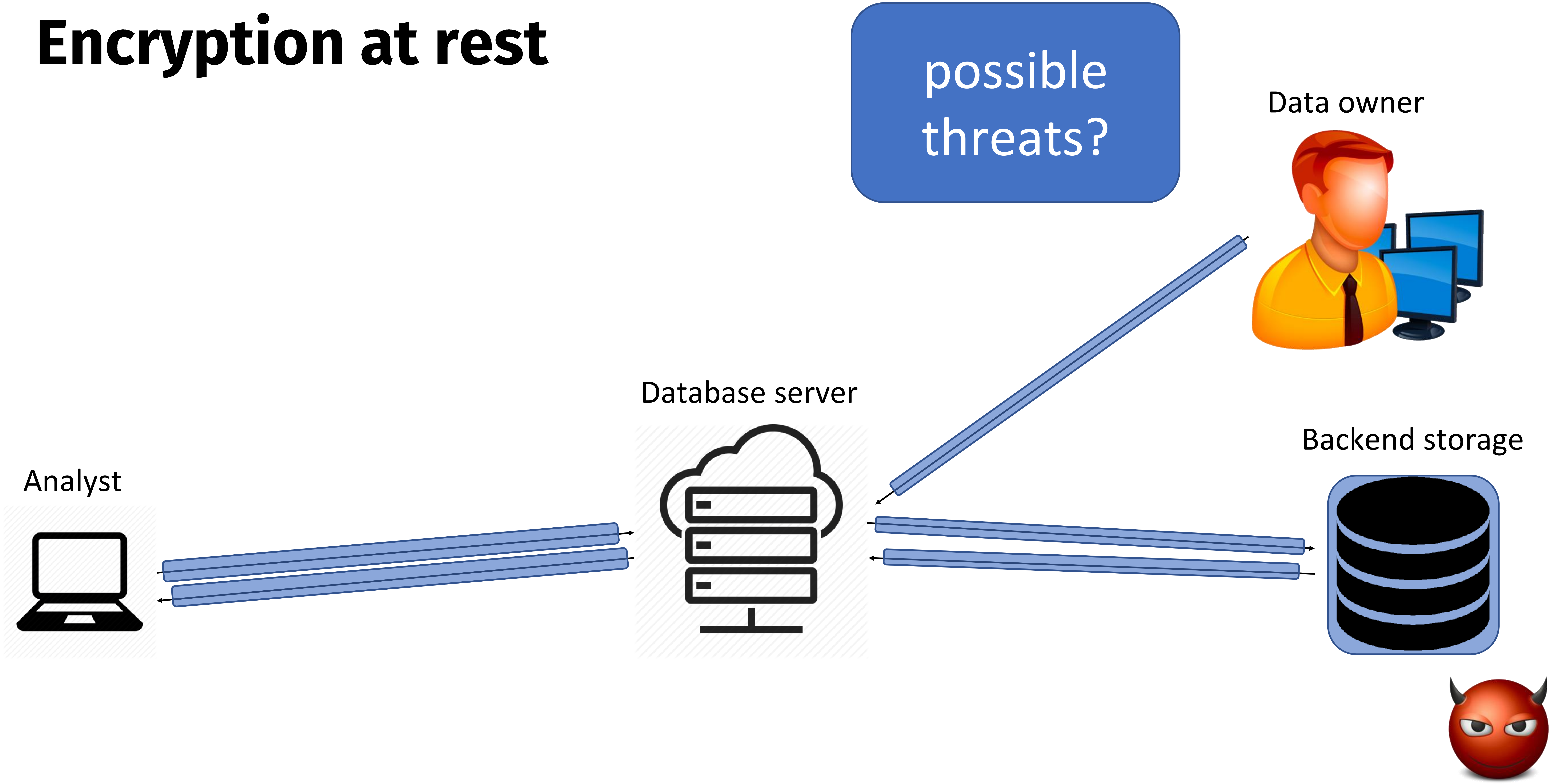
Let's protect a database



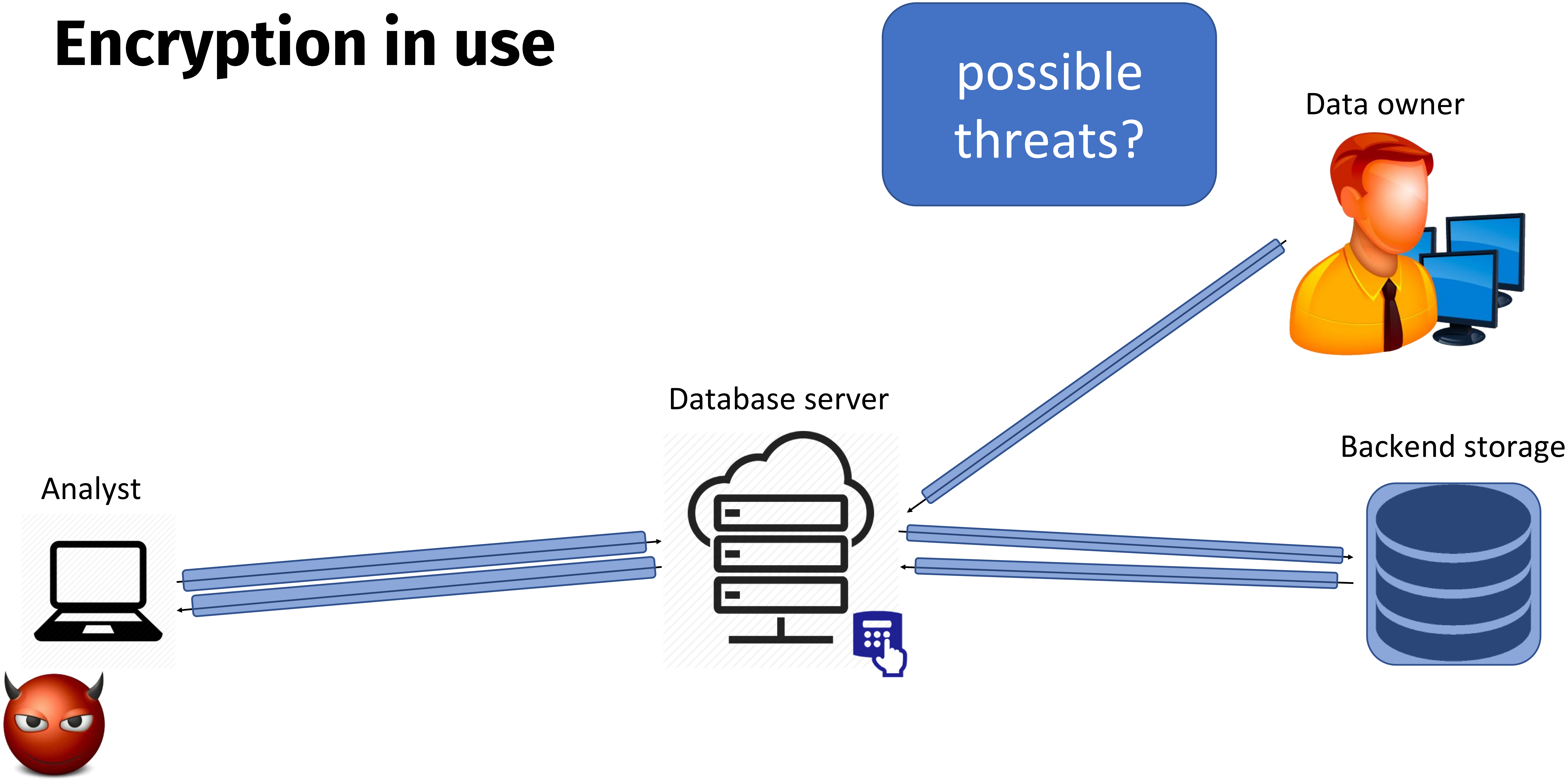
Encryption in transit



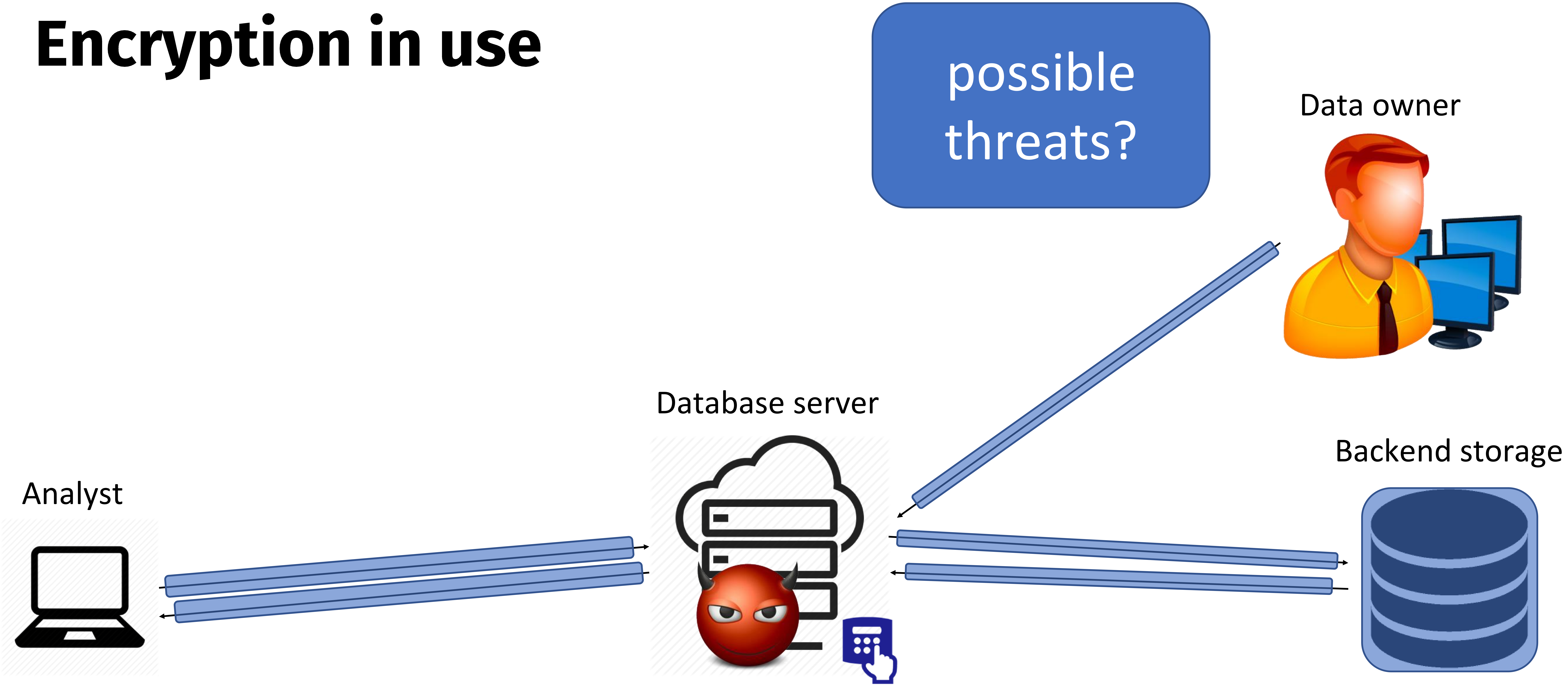
Encryption at rest



Encryption in use



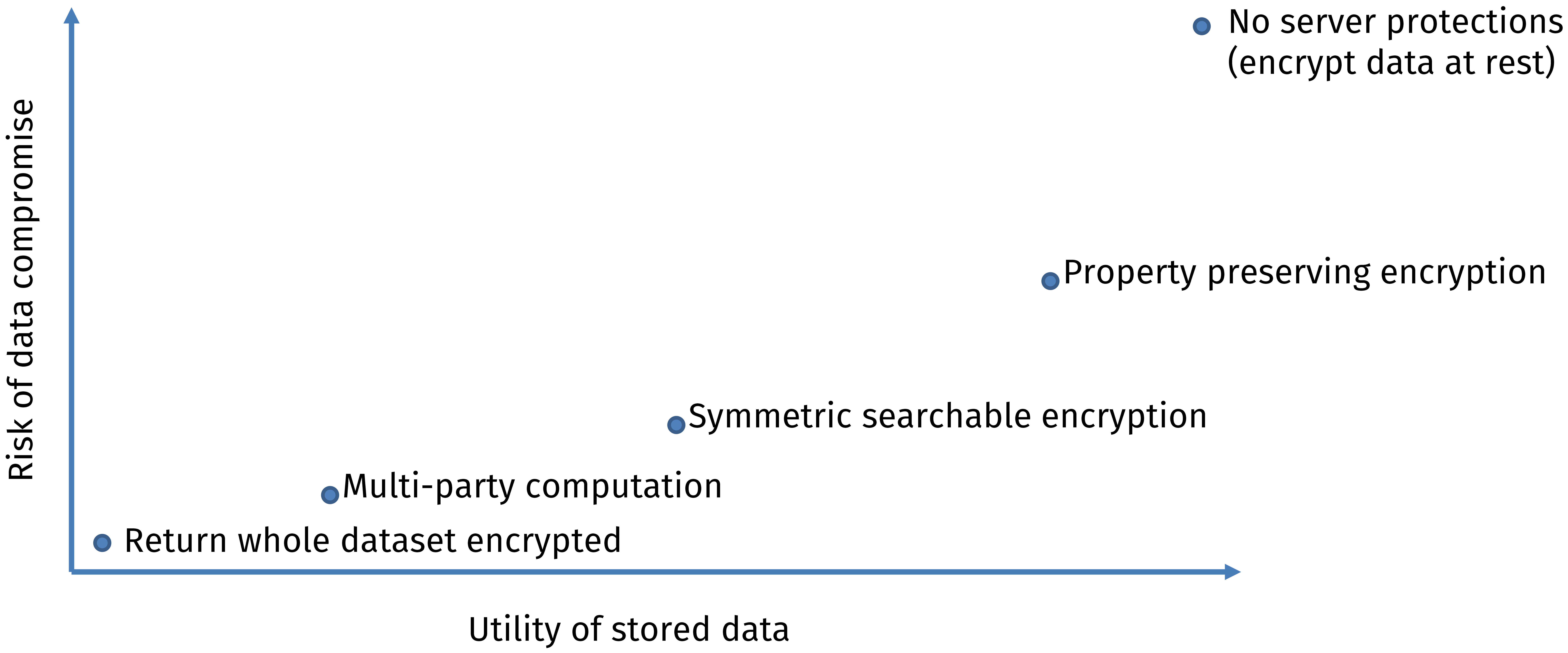
Encryption in use



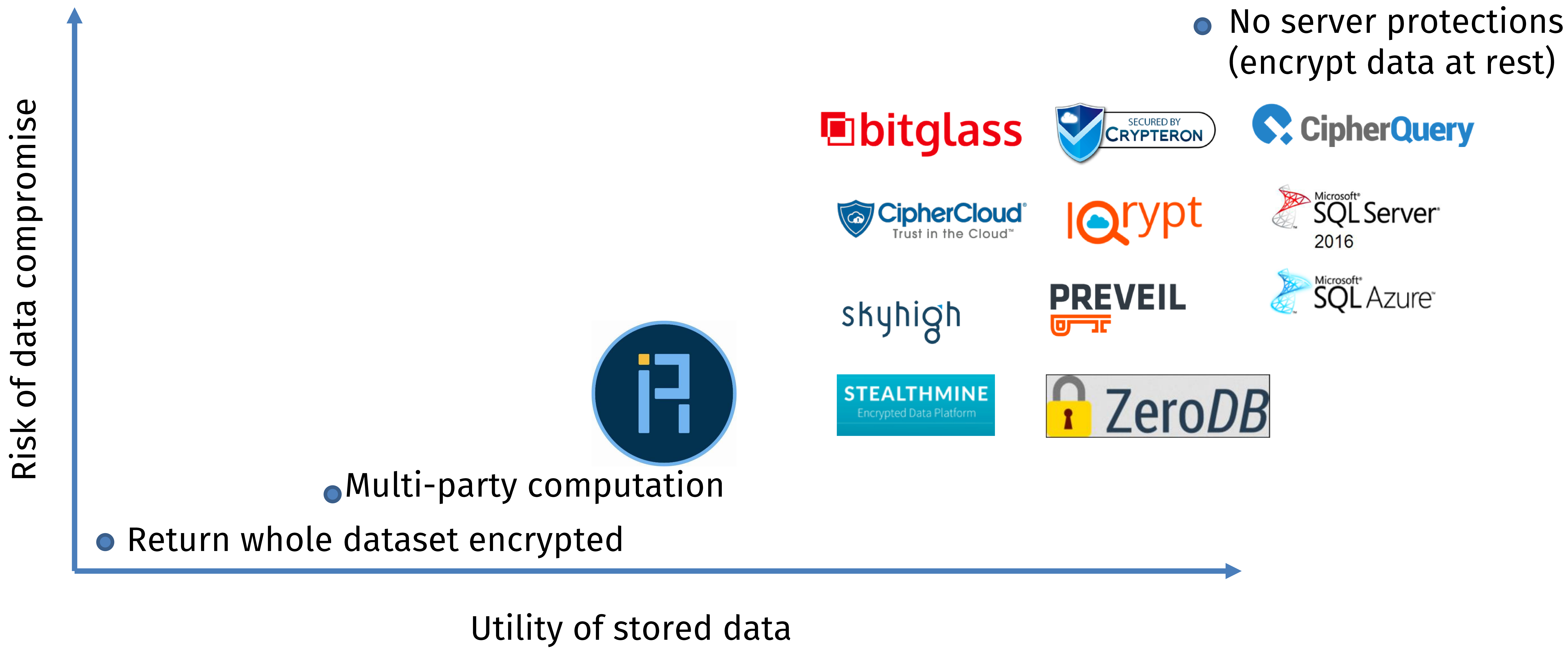
Desired goal: "encrypted indexes" that permit the server to search directly over encrypted records

- Server shouldn't see either data or queries
- Server might observe access patterns though

Cryptographically protected database search

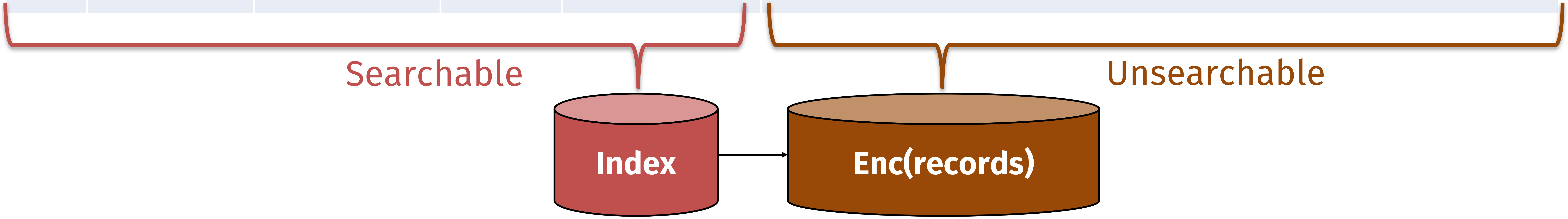


State of the art



Abstract view of a single-table database

id	fname	lname	Age	Income	Photo
1	Alice	Jones	20	71,000	<alice.jpg>
2	Bob	Jones	25	58,000	<bob.jpg>
3	Charlie	Smith	50	62,000	<charlie.jpg>
4	David	Williams	55	75,000	<david.jpg>



Small data structure: map searchable terms to associated record ids

Large file store: standard authenticated encryption applied to each record

1. Property Preserving Encryption (PPE)

- Apply transformation that preserves relevant features
- Insert into a legacy database for indexing & searching

id	fname	lname	Age	Income
1	Alice	Jones	20	71,000
2	Bob	Jones	25	58,000
3	Charlie	Smith	50	62,000
4	David	Williams	55	75,000

⇒

id	fname	lname	Age	Income
1	qlap1	Lf4Pz	cnr	$g^{71}\ r^{90}$
2	7fBwo	Lf4Pz	duo	$g^{58}\ r^{84}$
3	AKx0k	sw2AD	syv	$g^{62}\ r^{22}$
4	CK6ZD	6lVTH	tng	$g^{75}\ r^{38}$

Operation:	**DET (=)**	**OPE (<)**	**HOM (+, x)**
Method:	Choose Enc function at random	Choose random *monotonic* function	Public-key crypto
Drawback:	Cloud sees equality patterns	Cloud sees < and ~distances	Slow

1. Property Preserving Encryption (PPE)

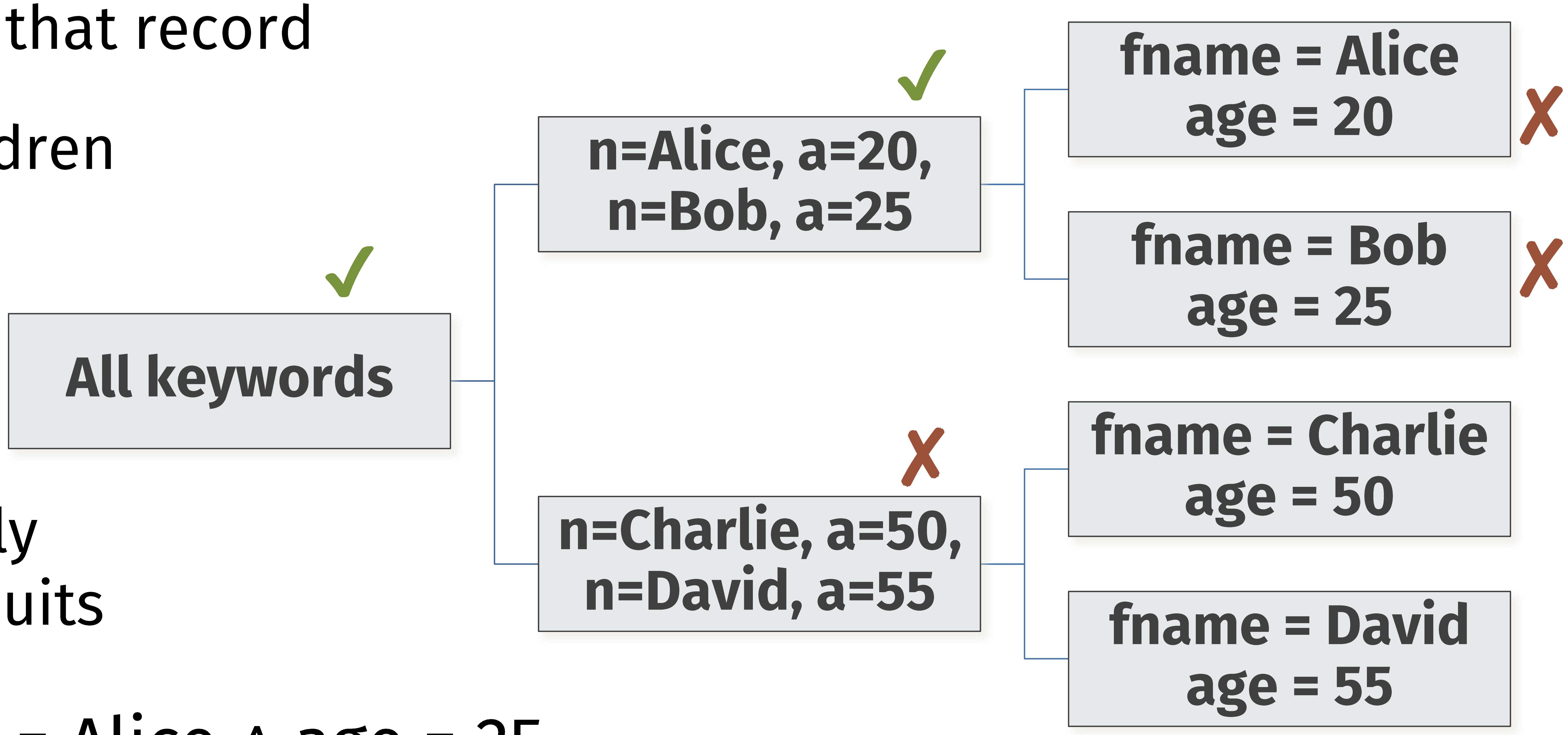
- Fast & legacy compliant
- Supported by a database near you!
  - Google: Encrypted BigQuery
  - Microsoft: SQL Server 2016, Azure SQL Database
  - Startups: Bitglass, Ciphercloud, CipherQuery, Crypteron, IQrypt, Kryptonostic, PreVeil, Skyhigh, ZeroDB, …
- Weakness: even though data isn't stored in the clear, the revealed information is strong enough to reconstruct data and queries

2. Searchable Symmetric Encryption (SSE)

- Privacy: reveals or “leaks” less information to the database server
- Query expressivity: large subset of SQL
- Scale: tested on databases with 100m records
- Performance: ~3-5x of MySQL

SSE example (Blind Seer)

- Consider a tree in which each node stores a set
  - Leaves: set of keywords in that record
  - Other nodes: union of children
- Roles
  - Data owner makes tree
  - Cloud server & client jointly traverse using garbled circuits



- Consider the query name = Alice ∧ age = 25
- Imperfect security: tree search pattern reveals info about data

SSE example (Blind Seer)

- Main cryptographic innovation: represent set as *encrypted* Bloom filter
- Evaluate each node of the tree using secure two-party computation

n=Alice, a=20,
n=Bob, a=25

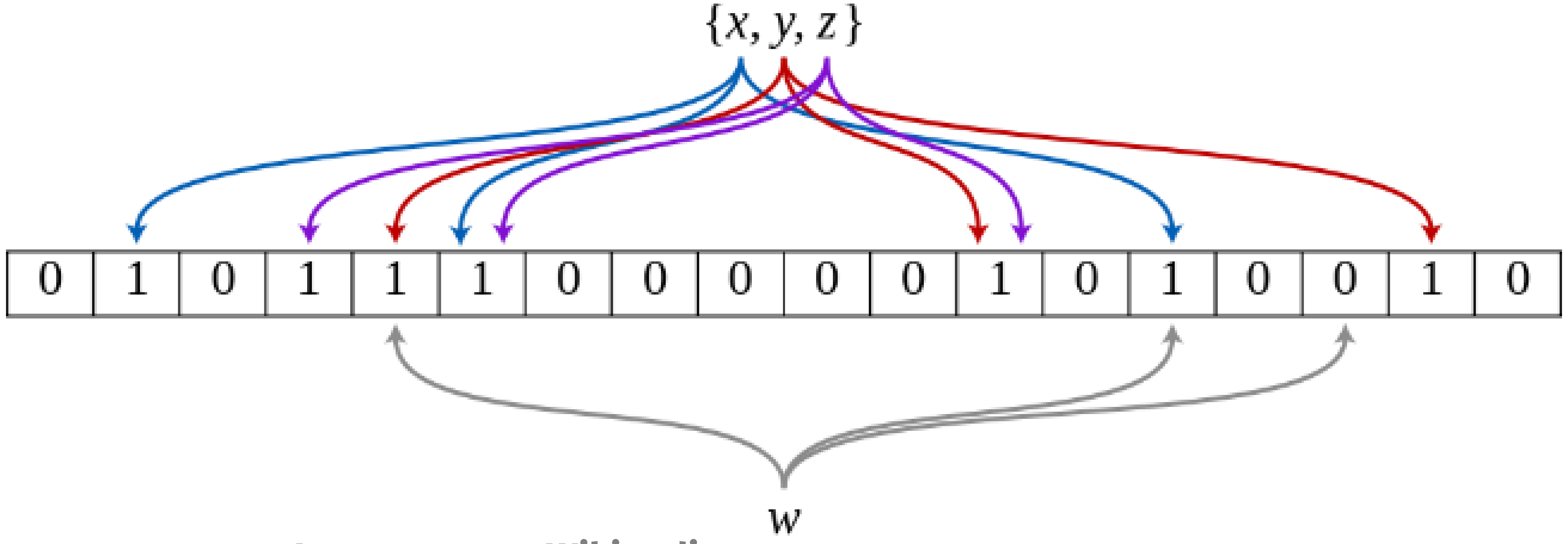


Image source: Wikipedia

Information revealed by SSE

- Protected search schemes reveal or leak some information about the query, data set, and result set to each party.
    1. Structure: size of an object, e.g. length of a string or cardinality of a set
    2. Identifiers: pointers to objects that persist across multiple accesses
    3. Equality or Order of values
- Some schemes leak:
    1. At Initialization on entire DB
    2. At Query on relevant records

Weekly reading: the Pareto Frontier of protected databases

Query type	Scheme (References)	Approach	Threats: # of parties	Threats: Adversarial Q	Threats: Adversarial S	S leakage: Init	S leakage: Query	Scale: Updatable?	Scale: Implemented?	Scale: Scale tested	Crypto: Crypto type	Crypto: Insert: # ops	Crypto: Query: # ops	Network: # round trips	Network: Data sent	Unique feature
Equality	Arx-EQ [14]	Legacy	2	—	◐	○	◔	●	✔	◐	●	●	●	●	●	legacy compliant
	Kamara-Papamanthou [106]	Custom	2	—	◐	○	◔	●	—	—	●	○	○	●	●	parallelizable
	Blind Storage [100]	Custom	2	—	◐	○	◔	●	✔	◐	●	◐	●	◐	●	low S work
	Sophos ($\Sigma o \phi o \varsigma$) [101]	Custom	2	—	◐	○	◔	●	✔	◐	○	◐	●	●	●	**Refresh** w/ **Insert**
	Stefanov et al [107]	Custom	2	—	◐	○	◔	●	✔	◐	●	○	○	●	●	**Refresh** w/ **Insert**
	vORAM+HIRB [120]	Obliv	2	—	◐	○	○	●	✔	●	●	○	○	○	◔	history independ.
	TWORAM [121]	Obliv	2	—	◐	○	○	●	—	—	◐	○	○	◐	◔	const round
	3PC-ORAM [124]	Obliv	3	◐	◐	○	○	●	✔	◔	●	○	○	○	◔	dual S
Boolean	DET [15], [92]	Legacy	2	—	◐	◕	◕	●	✔	●	●	◐	◐	●	●	supports JOINs
	BLIND SEER [16], [17]	Custom	3	●	●	○	◐	◐	✔	●	◐	◐	○	○	◔	hide field, r_i's
	OSPIR-OXT [18]–[21], [104]	Custom	3	●	◐	○	◐	●	✔	●	◐	◐	◐	◕	●	excels w/ small r_1
	Kamara-Moataz [102]	Custom	2	—	◐	○	◐	○	—	—	◐	◐	○	●	◔	relational SPC
Range	OPE [93]–[95]	Legacy	2	—	◐	●	●	●	✔	●	●	●	●	●	●	leak some content
	Mutable OPE [97]	Legacy	2	—	◐	●	●	●	✔	◐	●	○	○	○	◐	interactive
	Partial OPE [111]	Custom	2	—	◐	○	●	●	✔	◐	●	●	●	◐	●	fast insertions
	Arx-RANGE [110]	Custom	2	—	◐	○	◐	●	✔	◐	◐	○	○	●	○	non-interactive
	SisoSPIR [22]	Obliv	3	◐	◐	○	○	○	✔	●	●	●	●	○	◔	split, non-colluding S
Other	GraphEnc_1 [116]	Custom	2	—	◐	◐	◔	○	✔	◐	●	●	●	●	◔	approx. graph dist.
	GraphEnc_3 [116]	Custom	2	—	◐	◐	◐	○	✔	◐	○	●	●	●	●	approx. graph dist.
	Chase-Shen [109], [126]	Custom	2	—	●	○	◐	○	✔	◔	●	●	●	◐	●	substring search
	Moataz-Blass [123]	Obliv	2	—	◐	○	○	●	✔	●	●	○	○	○	◔	substring search

Weekly reading: inference attacks from leaked information

Attacker goal	Required S leakage: Init	Required S leakage: Query	Required attack conditions: Ability to inject data	Required attack conditions: Prior knowledge	Attack efficacy: Runtime	Attack efficacy: Sensitivity to prior knowledge	Attack efficacy: Keyword universe tested	Attack name
Query Recovery	○	○	—	◔	●	?	○	Communication Volume Attack [125]
	○	◔	✔	○	○	○	○	Binary Search Attack [127]
	○	◔	—	◔	●	?	○	Access Pattern Attack [125]
	○	◔	—	◕	◑	●	●	Partially Known Documents [128]
	○	◔	✔	◕	◑	○	●	Hierarchical-Search Attack [127]
	○	◔	—	●	◑	●	●	Count Attack [128]
Data Recovery	○	◔	—	◑	●	●	◑	Graph Matching Attack [129]
	◕	—	—	◑	○	?	○	Frequency Analysis [130]
	◕	—	✔	◑	○	?	●	Active Attacks [128]
	◕	—	—	◕	○	?	●	Known Document Attacks [128]
	●	—	—	◑	○	○	●	Non-Crossing Attack [131]

TABLE III

Summary of current leakage inference attacks against protected search base queries. S is the server and the assumed attacker for all attacks listed. S leakage symbols have the same meaning as in Table II. Each attack is relevant to schemes in Table II with at least the S leakage specified in this table. Some attacks require the attacker to be able to inject data by having the provider insert it into the database. Legends for the rest of the columns follow. In all columns except "Keyword universe tested," bubbles that are more filled in represent properties that are better for the scheme and worse for the attacker.

Prior knowledge
- ●– Contents of full dataset
- ◕– Contents of a subset of dataset
- ◑– Distributional knowledge of dataset
- ◔– Distributional knowledge of queries
- ○– Keyword universe

Runtime (in # of keywords)
- ●– More than quadratic
- ◑– Quadratic
- ○– Linear

Sensitivity to prior knowledge
- ●– High
- ○– Low
- ? – Untested

Keyword universe tested
- ●– > 1000
- ◑– 500 to 1000
- ○– < 500