# Optimizing Private Information Retrieval for Compromised Credential Checking

Daniel Günther
TU Darmstadt
guenther@encrypto.cs.tu-darmstadt.de

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

# Data Breaches

## And their Consequences

**Collection #1 - 5 (2019)**

Over 2.2 billion credentials are published in plaintext! [Greenberg, 2019]

**The Problem....**

6,9% of the credentials are still valid on non-exposed websites! [Thomas et al., USENIX Security'19]

**Credential Stuffing Attacks (CSA)**

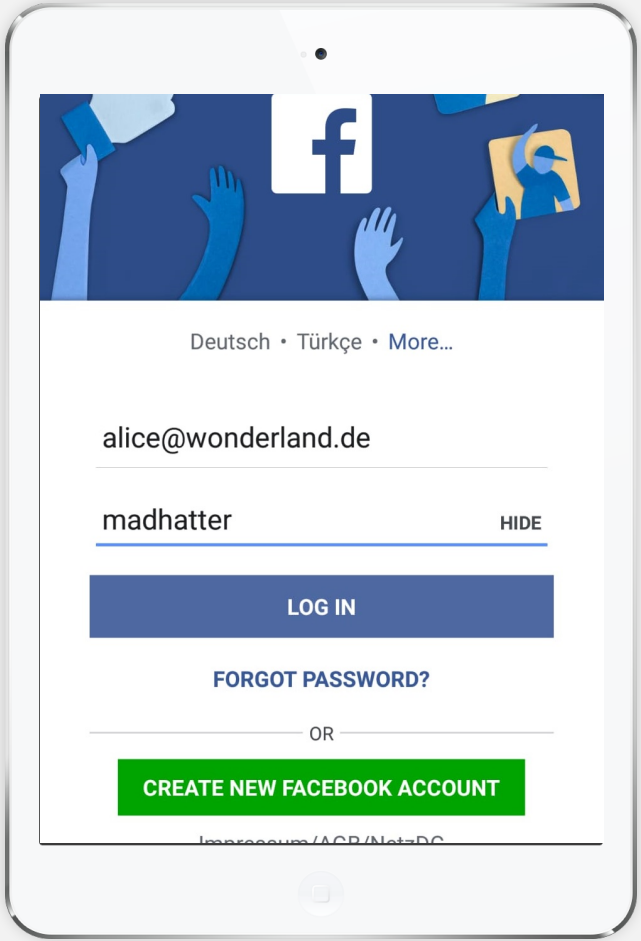
Adversaries can use this data to easily exploit CSA attacks! [Thomas et al., USENIX Security'19; Li et al., ACM CCS'19]

# Credential Stuffing Attack (CSA)

alice@wonderland.de,
madhatter

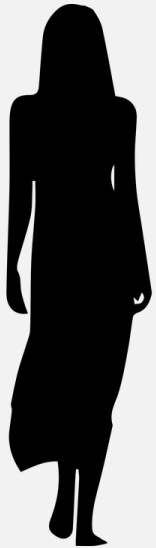
true / false

| Username | Password |
|---|---|
| ..... | ..... |
| alice@wonderland.de | madhatter |
| bob@builder.com | excavator |
| .... | .... |



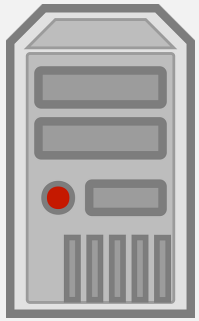
# Compromised Credential Checking (C3)

## Idea



**Malicious Server:**
This information should be hidden from the server!

alice@wonderland.de,
madhatter



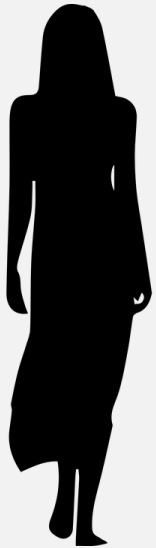
true / false

**Malicious Client:**
The database should be hidden from Alice!

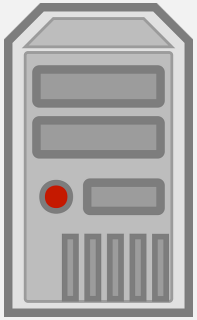
| Username | Password |
|---|---|
| ..... | ..... |
| alice@wonderland.de | madhatter |
| bob@builder.com | excavator |
| .... | .... |

# Compromised Credential Checking (C3)

## Current Approach: Bucketization



$= f(u, p)$



$= DB_{f(u,p)}$

contains all elements with property

$u$ = alice@wonderland.de

$p$ = madhatter

# Compromised Credential Checking (C3)

## How to define the Bucketization Function $f$?

Usually, $f$ is a prefix of a hash value.
This information is revealed to the server!

**HaveIBeenPwned (HIBP)**

**[Hunt, 2019]**

$f$ is a 20 bit hash prefix of the username or password.

Protection against malicious servers.

**Google Password Checkup**

**[Thomas et al., USENIX Security'19]**

$f$ is a 16 bit hash prefix of the username and password.

Protection against malicious clients and servers.

# Compromised Credential Checking (C3)

## Security of GPC [Thomas et al., USENIX Security'19]

If the adversary knows $f(u, p)$ and has access to $DB_{f(u,p)}$, he has non-negligible advantage to guess the correct password of Alice. [Li et al., ACM CCS'19]

Assume the adversary knows:

- the transcript of the GPC protocol
- the username of Alice
- the data breach

The adversary is able to:

- guess the correct password in 1000 attempts with probability of 60%
- guess the correct password in 10 attempts with probability of 33%

# Compromised Credential Checking (C3)

## Fix C3

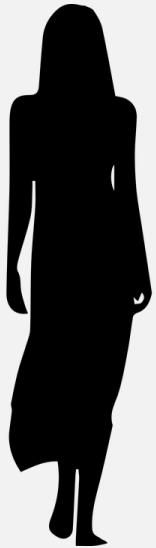
**C3 with Private Information Retrieval (PIR) [this thesis]**

- proposed by Thomas et al. [USENIX Security'19] and Lie et al. [ACM CCS'19]
  - both say that PIR is too inefficient

- We show that PIR is efficient enough for C3 by introducing a new PIR model

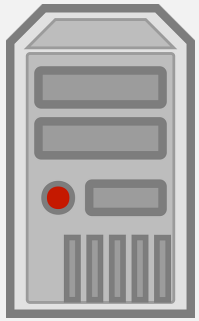
# Private Information Retrieval (PIR)

## Idea



**Malicious Server:**
Server shall not learn the index of the desired entry!

$idx$



$DB[idx]$

The database is public!

**Database *DB***

.....

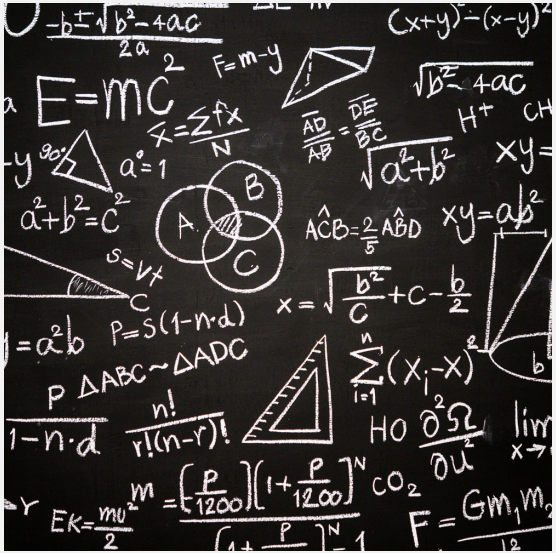
3a7e2652

3b9837ae

....

# Private Information Retrieval (PIR)

## Many Applications

### C3

Securely check if the user's credentials are leaked.

### Private Messaging App

Protect messages and metadata of the users as in OnionPIR [Demmler et al., ACNS'17].

### Private Set Membership

Securely check if an entry exists in a public database.

# Summary of our Contributions

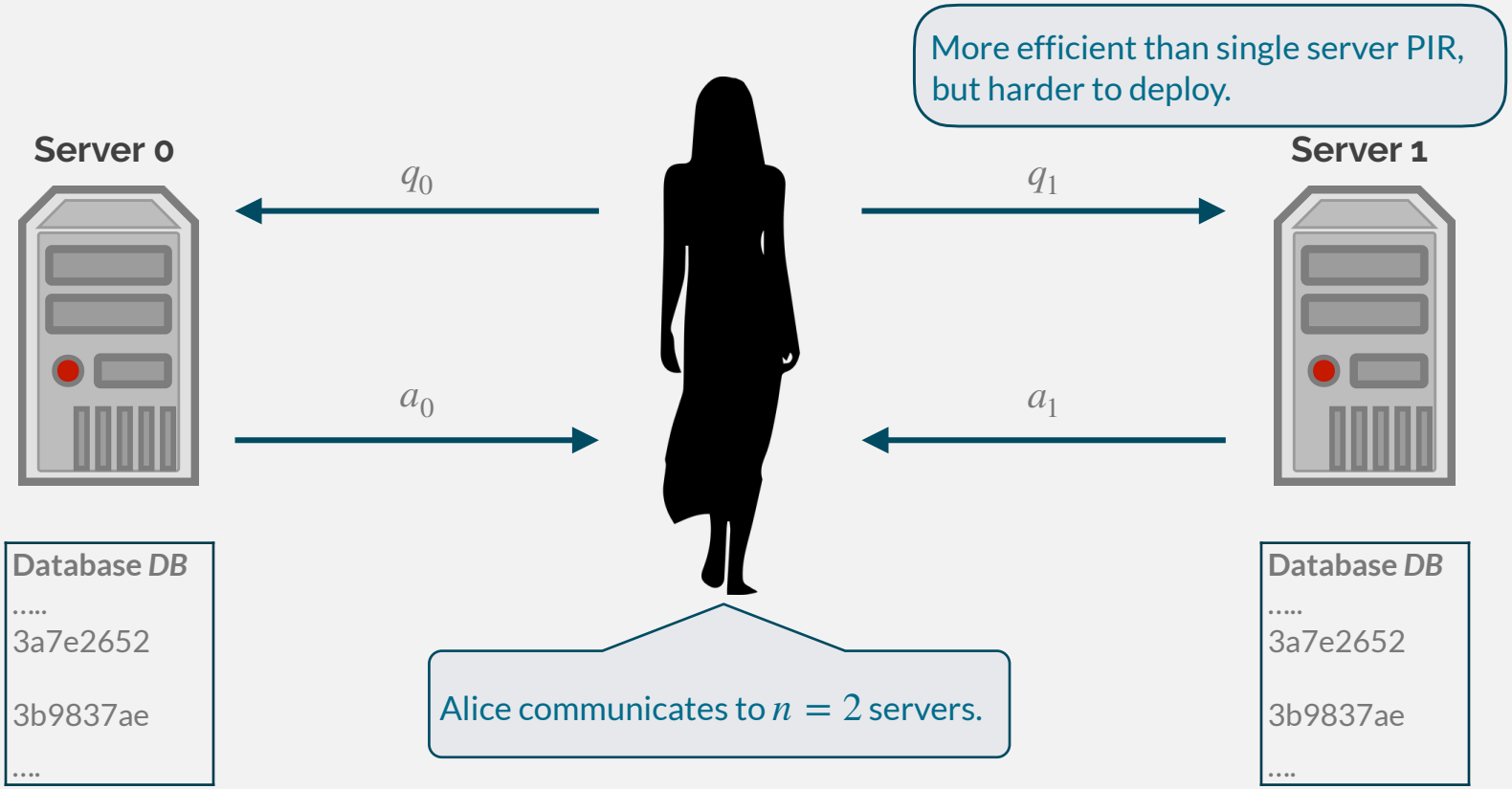
**First C3 scheme with perfect anonymity**

**Query-Dependent Preprocessing PIR Model**

**Implementation of RAID-PIR in our new Model**

# Private Information Retrieval (PIR)

## Multi Server PIR [Chor et al., IEEE FOCS'95]



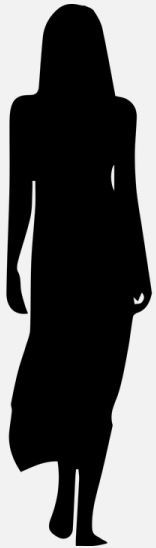
# Private Information Retrieval (PIR)

## Classical Model

**input:** index $idx$

**input:** data $D$

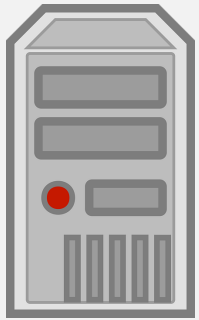
$(DB_i, M_i) \leftarrow \text{Create}(D)$



$(q_0, \ldots, q_{n-1}) \leftarrow \text{Request}(idx)$

$q_i$

**Server $i$**



$a_i \leftarrow \text{Response}(q_i, DB_i, M_i)$

$a_i$

$d \leftarrow \text{Combine}(a_0, \ldots, a_{n-1})$

**Database *DB***

.....

3a7e2652

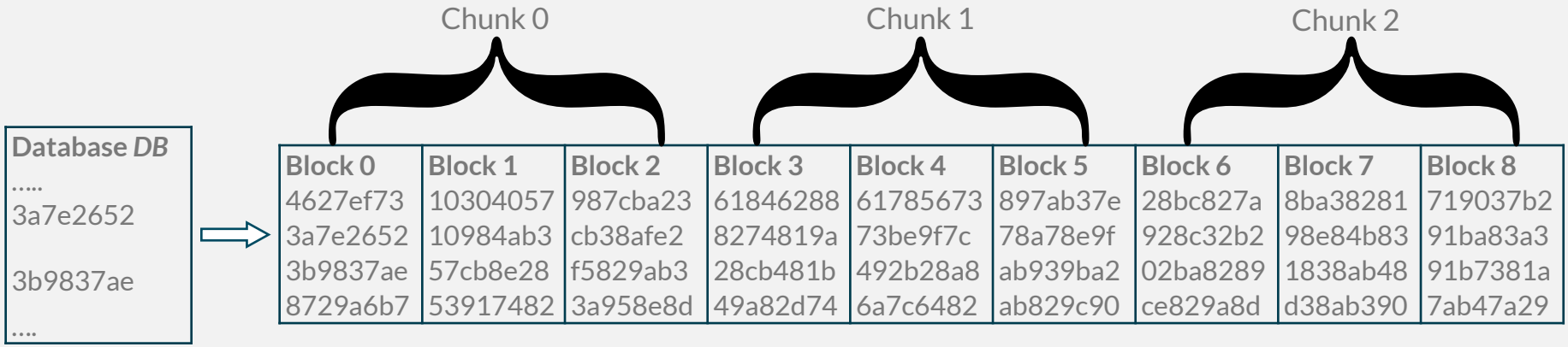
3b9837ae

....

**output:** $d = DB[idx]$

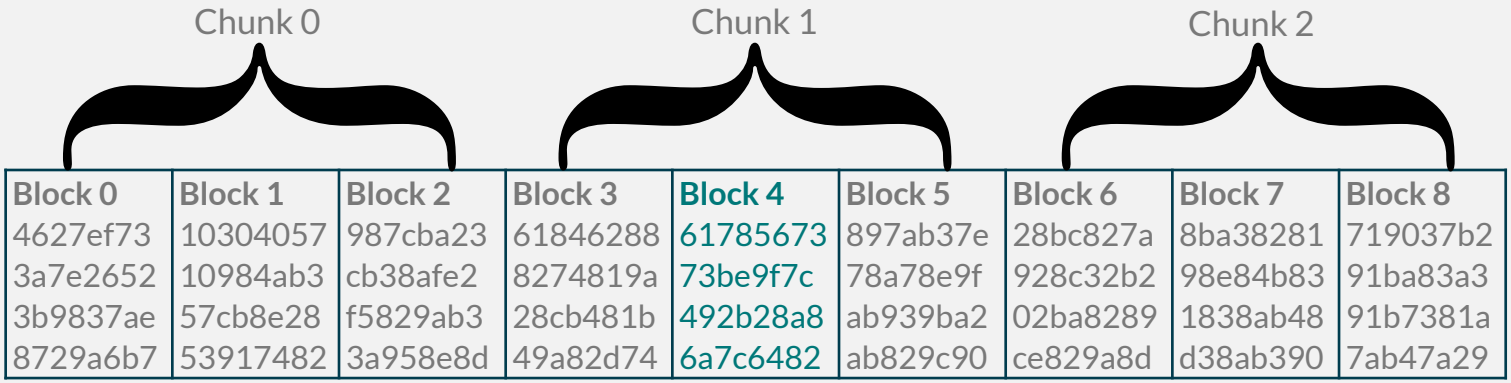
# RAID-PIR [Demmler et al., ACM CCSW'14]

## Create



# RAID-PIR [Demmler et al., ACM CCSW'14]

## Request



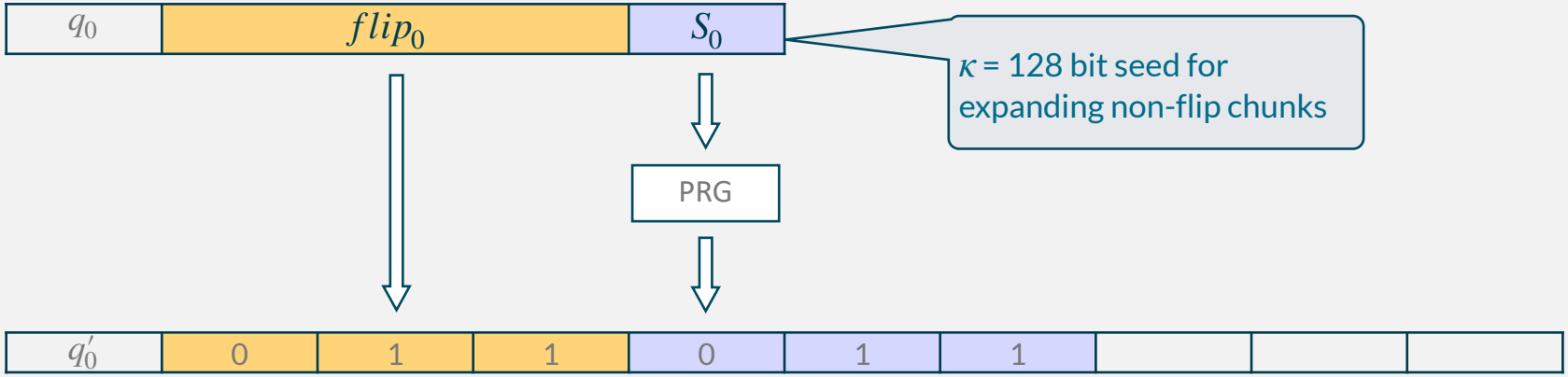

| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 |
|---|---|---|---|---|---|---|---|---|
| 4627ef73 | 10304057 | 987cba23 | 61846288 | 61785673 | 897ab37e | 28bc827a | 8ba38281 | 719037b2 |
| 3a7e2652 | 10984ab3 | cb38afe2 | 8274819a | 73be9f7c | 78a78e9f | 928c32b2 | 98e84b83 | 91ba83a3 |
| 3b9837ae | 57cb8e28 | f5829ab3 | 28cb481b | 492b28a8 | ab939ba2 | 02ba8289 | 1838ab48 | 91b7381a |
| 8729a6b7 | 53917482 | 3a958e8d | 49a82d74 | 6a7c6482 | ab829c90 | ce829a8d | d38ab390 | 7ab47a29 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $q_0$ | 0 | 1 | 1 | 0 | 1 | 1 | | |
| $q_1$ | | | | 0 | 0 | 1 | 0 | 1 | 1 |
| $q_2$ | 0 | 1 | 1 | | | | 0 | 1 | 1 |
| $\oplus\, q_i$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Each server has to process 2 chunks.
Chunk i = flip chunk of server i

# RAID-PIR [Demmler et al., ACM CCSW'14]

## Optimized Request Query [Demmler et al., ACNS'17]




Client chooses seed $S_i$ for each of the $n$ servers.

Why don't let the servers choose the seeds on their own?

# QDP-RAID-PIR

## Query-Dependent Preprocessing Model

Server 0

$S_0$

PRG

0 | 1 | 1

| Block 3 | Block 4 | Block 5 |
|---|---|---|
| 61846288 | 61785673 | 897ab37e |
| 8274819a | 73be9f7c | 78a78e9f |

Block 4 XOR 5
e802e50d
0b1911e3

Client

$S_0$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $q_0$ | 0 | 1 | 1 | 0 | 1 | 1 | | | |
| $q_1$ | | | | 0 | 0 | 1 | 0 | 1 | 1 |
| $q_2$ | 0 | 1 | 1 | | | | 0 | 1 | 1 |
| $\oplus\, q_i$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

$flip_0$

0 | 1 | 1

| Block 0 | Block 1 | Block 2 |
|---|---|---|
| 4627ef73 | 10304057 | 987cba23 |
| 3a7e2652 | 10984ab3 | cb38afe2 |

| Block $a_0$ | Block $a_1$ | Block $a_2$ |
|---|---|---|
| 604e1f79 | c3f7ab21 | 897ab37e |
| d0b9f4b2 | 5092b731 | 78a78e9f |

Block $a_0$
604e1f79
d0b9f4b2

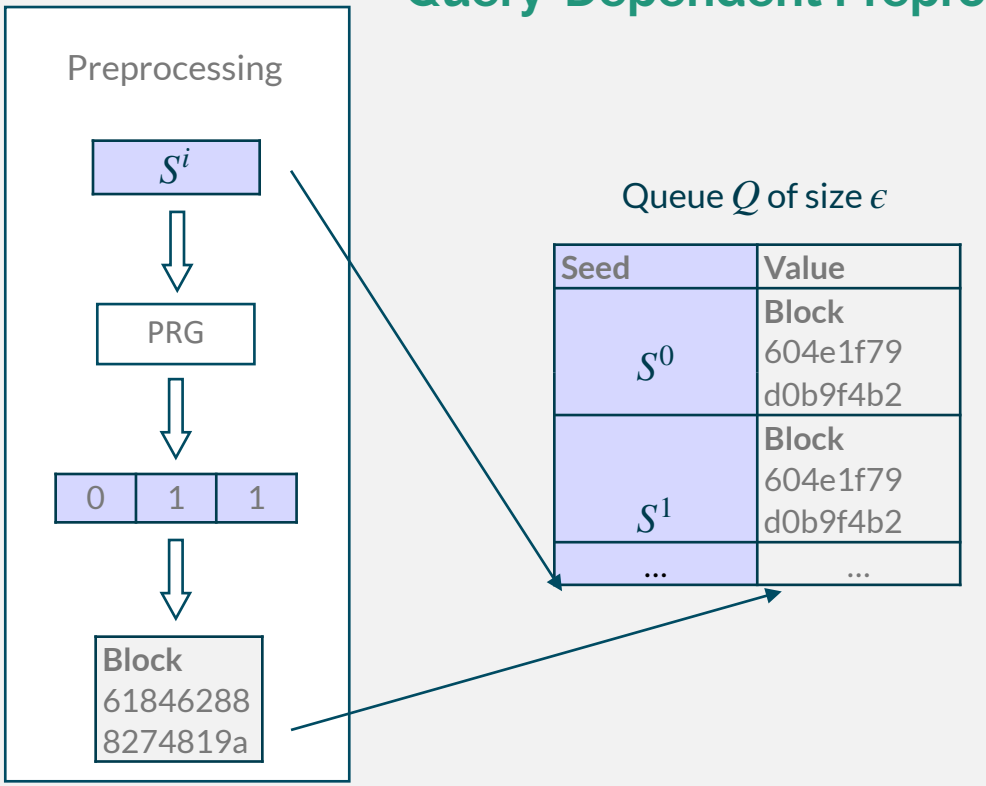
Block 4
61785673
73be9f7c

Block $a_0$
604e1f79
d0b9f4b2

Offline Computation

# QDP-RAID-PIR

## Query-Dependent Preprocessing Model




**Three Operation Modes:**

1. **ALWAYS**: Pause the Preprocessing process when a new query arrives.
2. **NEVER**: Never pause the Preprocessing process.
3. **HALF**: Pause the Preprocessing process when a new query arrives and the queue is less than half full.

# Private Information Retrieval (PIR)

## Query-Dependent Preprocessing Model

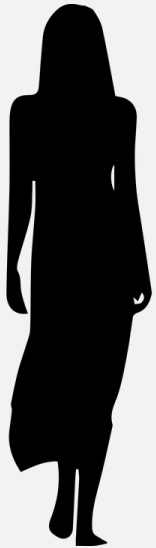
**input:** index $idx$

**input:** data $D$

$(DB_i, M_i) \leftarrow \text{Create}(D)$

$Q_i \leftarrow []$

Start Preprocess



$(S_i, A_i) \leftarrow Q_i.pop()$

$S_i$

$(q_0, \ldots, q_{n-1}) \leftarrow \text{Request}(idx, S_i)$

$q_i$

Pause Preprocess

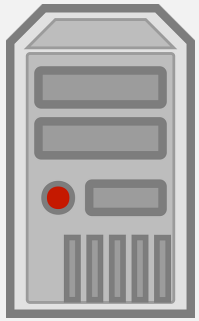
$a_i \leftarrow \text{Response}(q_i, DB_i, M_i)$

Start Preprocess

$a_i$

$d \leftarrow \text{Combine}(a_0, \ldots, a_{n-1})$

**Server** $i$



**Database** *DB*

.....

3a7e2652

3b9837ae

....

**output:** $d = DB[idx]$

# Implementation

## Three Components

### Database Generation

- Random 32 byte entries
- Quicksort
- Preprocessing

### Server

- Our preprocessing routine
- Pipeline
- Fast XOR operations with OpenMP

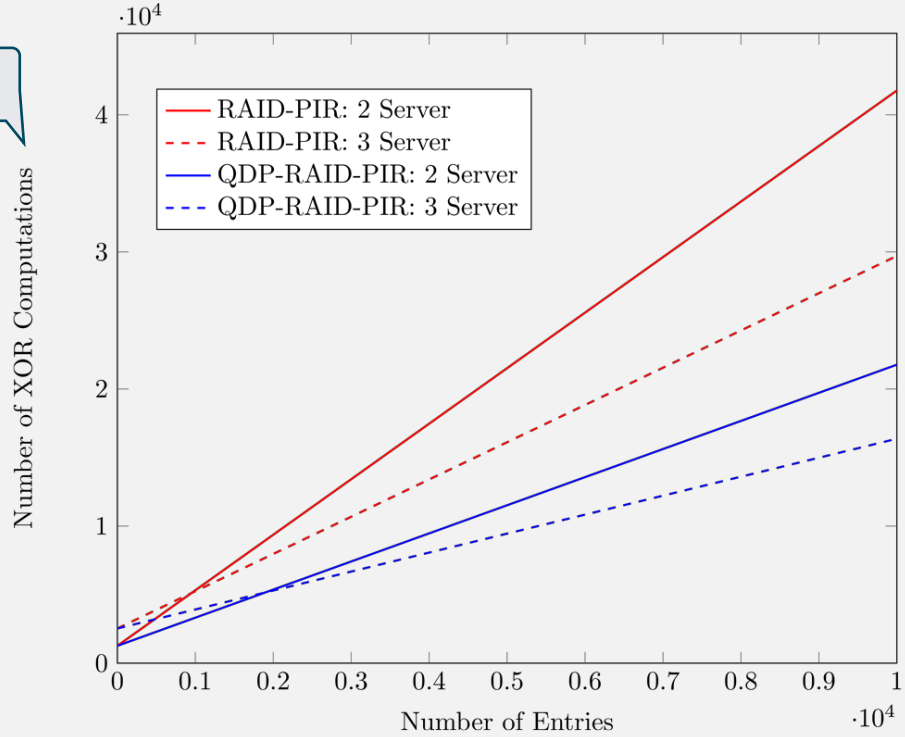
### Client

- Request block index

# Theoretical Complexity

## Computation

**Online Phase!**




Redundancy parameter $r = 2$

- ≈ 50% improvement over RAID-PIR for 2 server
- ≈ 33% improvement over RAID-PIR for 3 server

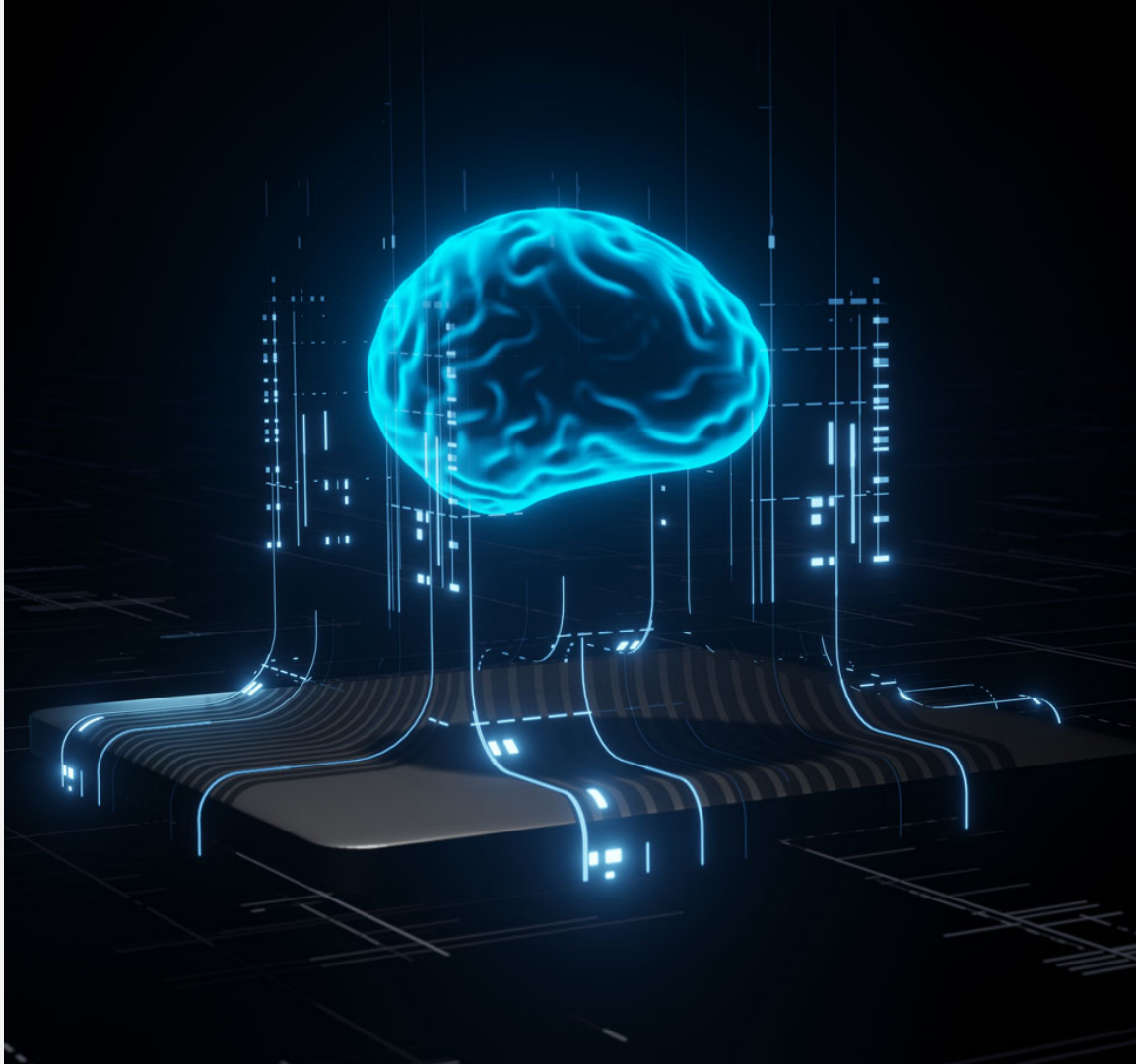
- Theoretical maximum improvement: $(n-1)/n$
- Theoretical improvement: $(r-1)/n$

# PIR Benchmarks

## Setup

**3 Servers + Client:**

- Arch Linux
- 16 Core Intel Core i9-7960X
- 128 Gb DDR4 RAM
- AVX-512 Instructions



Optimal Blocksize: $\sqrt{\frac{32(\#\text{Entries})}{\#\text{Server}}}$

# PIR Benchmarks

## Runtime and Communication (LAN)

Small Communication Overhead

40% improvement!

| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 265 | 2 | 13 | 9 | 2.53 | 2.59 |
| 1 000 000 | 4 000 | 3 | 40 | 23 | 8.00 | 8.06 |
| 10 000 000 | 12 650 | 6 | 273 | 174 | 25.30 | 25.36 |
| 50 000 000 | 28 285 | 14 | 846 | 517 | 56.57 | 56.63 |

**2 Server, redundancy parameter $r = 2$**

66% improvement!

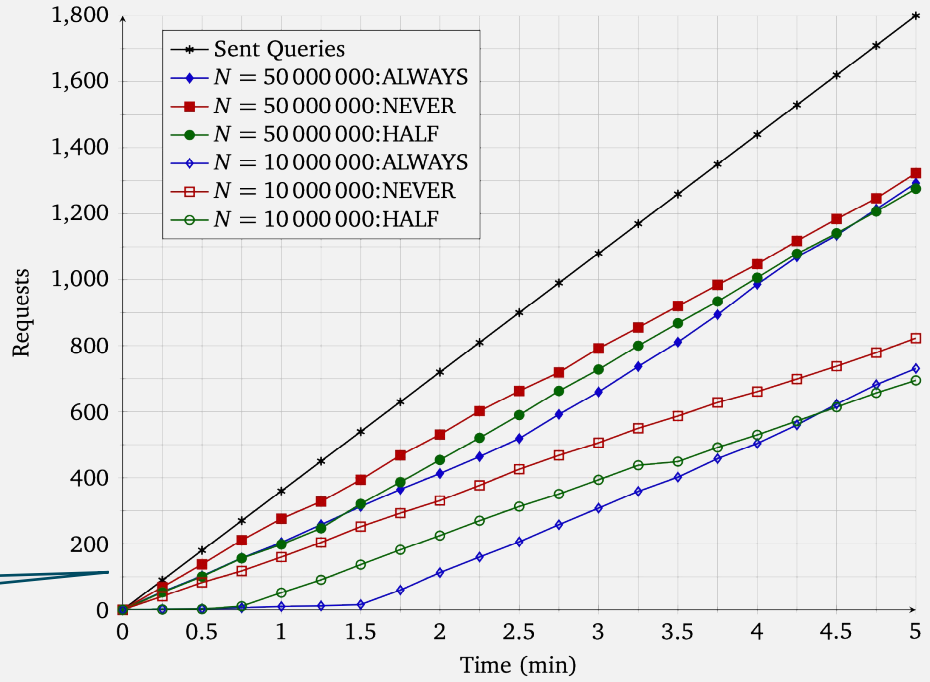
| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 033 | 2 | 6 | 3 | 3.10 | 3.16 |
| 1 000 000 | 3 265 | 5 | 72 | 26 | 9.80 | 9.89 |
| 10 000 000 | 10 372 | 8 | 381 | 132 | 30.85 | 31.21 |
| 50 000 000 | 23 095 | 14 | 1 287 | 425 | 69.28 | 69.38 |

**3 Server, redundancy parameter $r = 3$**

# PIR Robustness Benchmarks

**Experiment:**

- Three clients send a request each every 500 ms
- Each run starts with a full queue $Q_i$ of $\epsilon$ = 500 entries




Unprocessed Requests!

# Private Information Retrieval (PIR)

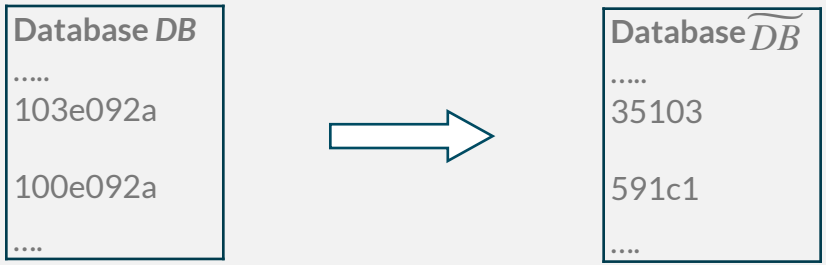
## Compressible PIR

Having a compression Function $\rho$, we can compress each block of the database.

→ better storage!

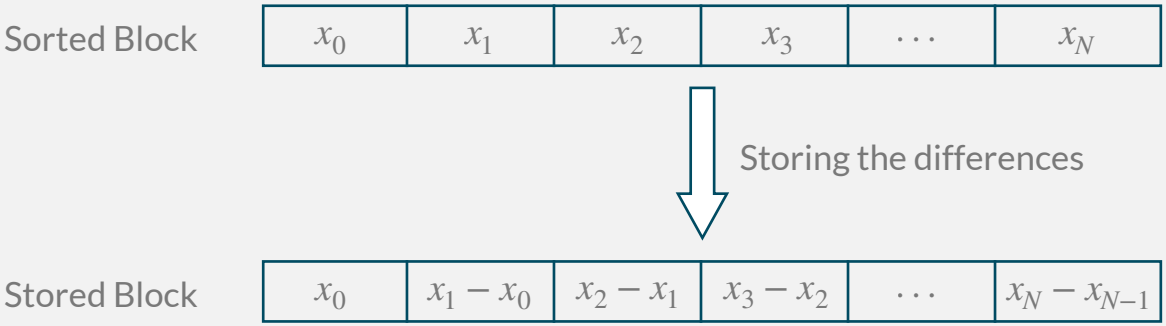
→ better online computation!

→ better communication!



# Private Information Retrieval (PIR)

## Sequence of Differences [Tamrakar et al., ACM ASIACCS'17]



# Private Information Retrieval (PIR)

## Compressible PIR

Sorted Database

Database *DB*
.....
100e092a

10139aeb

....

| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 |
|---|---|---|---|---|---|---|---|---|
| 100ab827 | 110a38bf | 1206cb3a | 13027fb3 | 14028ac3 | 15018427 | 1601836a | 17037ab3 | 180128ab |
| 100e092a | 110e5d3a | 12087b37 | 1307a2ad | 140ac3a2 | 1508b271 | 16047283 | 170847eb | 1802337a |
| 10139aeb | 110f3dd3 | 120fbe7a | 1307b4fc | 141429c3 | 150c28ab | 160b17e7 | 1710572b | 1803d216 |
| 1017b3e7 | 11128f37 | 1210f743 | 130c38a2 | 1418ab37 | 1510817a | 161018a2 | 171282b3 | 18057827 |

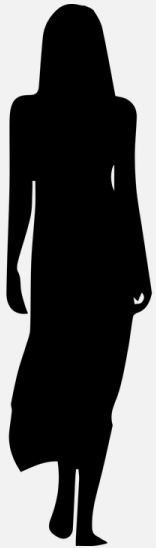
30% storage improvement!

| Block 0 | Block 1 | Block 2 | Block 3 | Block 4 | Block 5 | Block 6 | Block 7 | Block 8 |
|---|---|---|---|---|---|---|---|---|
| 100ab827 | 110a38bf | 1206cb3a | 13027fb3 | 14028ac3 | 15018427 | 1601836a | 17037ab3 | 180128ab |
| 35103 | 4247b | 1affd | 522fa | 838df | 72e4a | 2ef19 | 4cd38 | 10acf |
| 591c1 | e099 | 74343 | 124f | 96621 | 3763a | 6a564 | 80f40 | 19e9c |
| 418fc | 35164 | 138c9 | 483a6 | 48174 | 458cf | 500bb | 22b88 | 1a611 |

# Compromised Credential Checking (C3)

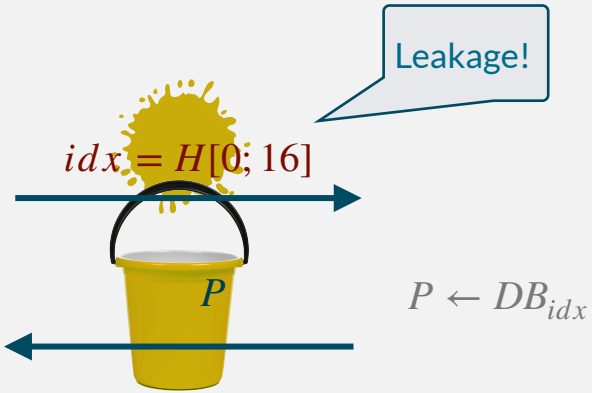
## Google Password Checkup (GPC)

**input:** username $u$,
password $p$



$H \leftarrow \mathscr{H}(u, p)$

**return** $H \in P$

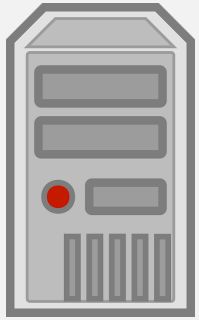



$P \leftarrow DB_{idx}$

**input:** data $D$

$DB \leftarrow \text{CreateDatabase}(D, b)$

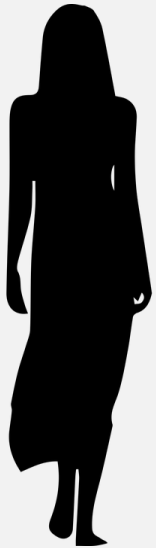
**Server**



# Compromised Credential Checking (C3)

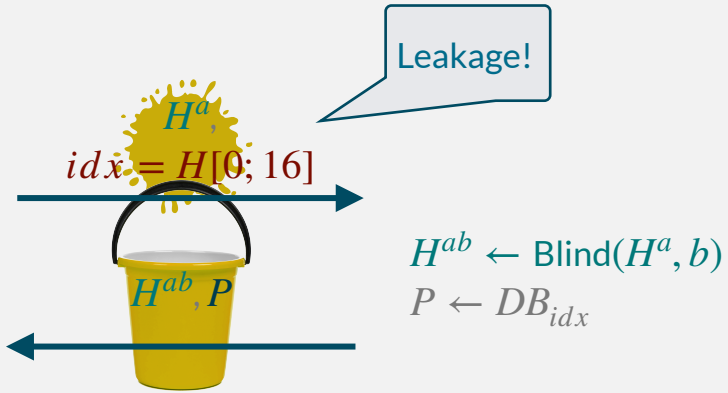
## Google Password Checkup (GPC)

**input:** username $u$,
password $p$



$a \leftarrow_{\$} \mathbf{G}$
$H \leftarrow \mathscr{H}(u, p)$
$H^a \leftarrow \text{Blind}(H, a)$

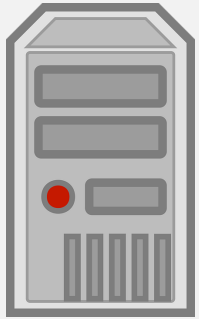
$H^b \leftarrow \text{Unblind}(H^{ab}, a)$
**return** $H^b \in P$




**input:** data $D$
$b \leftarrow_{\$} \mathbf{G}$
$DB \leftarrow \text{CreateDatabase}(D, b)$

$H^{ab} \leftarrow \text{Blind}(H^a, b)$
$P \leftarrow DB_{idx}$

**Server**



Private Set Intersection to protect against malicious clients.

# Compromised Credential Checking (C3)

## Our Scheme

**input:** username $u$,
password $p$

$a \leftarrow_{\$} \mathbf{G}$
$H \leftarrow \mathcal{H}(u, p)$
$H^a \leftarrow \text{Blind}(H, a)$
$idx = H[0; z]$
$(q_0, q_1) \leftarrow \text{PIR.Request}(idx, s_0, S_1)$

$H^a, q_i$ →

← $H^{ab}, P_i$

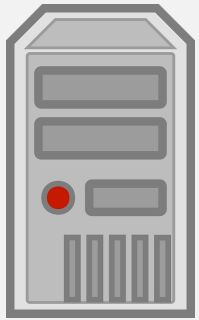
$H^b \leftarrow \text{Unblind}(H^{ab}, a)$
$DB_{idx} \leftarrow \text{PIR.Combine}(P_0, P_1)$

return $H^b \in DB_{idx}$

**input:** data $D$
$b \leftarrow_{\$} \mathbf{G}$
$(DB_i, M_i, Q_i) \leftarrow \text{CreateDatabase}(D, b)$

$H^{ab} \leftarrow \text{Blind}(H^a, b)$
$P_i \leftarrow \text{PIR.Response}(q_i, DB_i, M_i, A_i)$

**Server $i$**



# Compromised Credential Checking (C3)

## Our Scheme

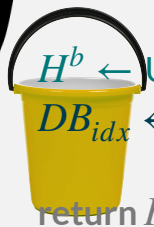
**input:** username $u$,
password $p$

$a \leftarrow_{\$} \mathbf{G}$
$H \leftarrow \mathscr{H}(u, p)$
$H^a \leftarrow \text{Blind}(H, a)$
$idx = H[0; z]$
$(q_0, q_1) \leftarrow \text{PIR.Request}(idx, s_0, S_1)$

$S_i$

$H^a, q_i$

$H^{ab}, P_i$



$H^b \leftarrow \text{Unblind}(H^{ab}, a)$
$DB_{idx} \leftarrow \text{PIR.Combine}(P_0, P_1)$

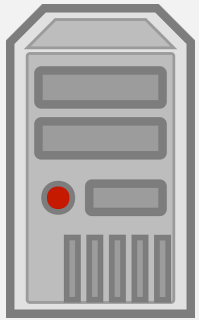
return $H^b \in DB_{idx}$

**input:** data $D$
$b \leftarrow_{\$} \mathbf{G}$
$(DB_i, M_i, Q_i) \leftarrow \text{CreateDatabase}(D, b)$

$(S_i, A_i) \leftarrow Q_i.pop()$

$H^{ab} \leftarrow \text{Blind}(H^a, b)$
$P_i \leftarrow \text{PIR.Response}(q_i, DB_i, M_i, A_i)$

**Server $i$**



# Compromised Credential Checking (C3)

## Our Scheme

**input:** username $u$,
password $p$

$a \leftarrow_{\$} \mathbf{G}$
$H \leftarrow \mathscr{H}(u, p)$
$H^a \leftarrow \text{Blind}(H, a)$
$idx = H[0; z]$
$(q_0, q_1) \leftarrow \text{PIR.Request}(idx, s_0, S_1)$

$S_i$

$H^a, q_i$

$H^{ab}, P_i$

$H^b \leftarrow \text{Unblind}(H^{ab}, a)$
$\widetilde{DB}_{idx} \leftarrow \text{PIR.Combine}(P_0, P_1)$
$DB_{idx} \leftarrow \rho^{-1}(\widetilde{DB}_{idx})$
**return** $H^b \in DB_{idx}$

**input:** data $D$
$b \leftarrow_{\$} \mathbf{G}$
$(DB_i, M_i, Q_i) \leftarrow \text{CreateDatabase}(D, b)$

$(S_i, A_i) \leftarrow Q_i.pop()$

$H^{ab} \leftarrow \text{Blind}(H^a, b)$
$P_i \leftarrow \text{PIR.Response}(q_i, DB_i, M_i, A_i)$

**Server $i$**

# C3 Benchmarks

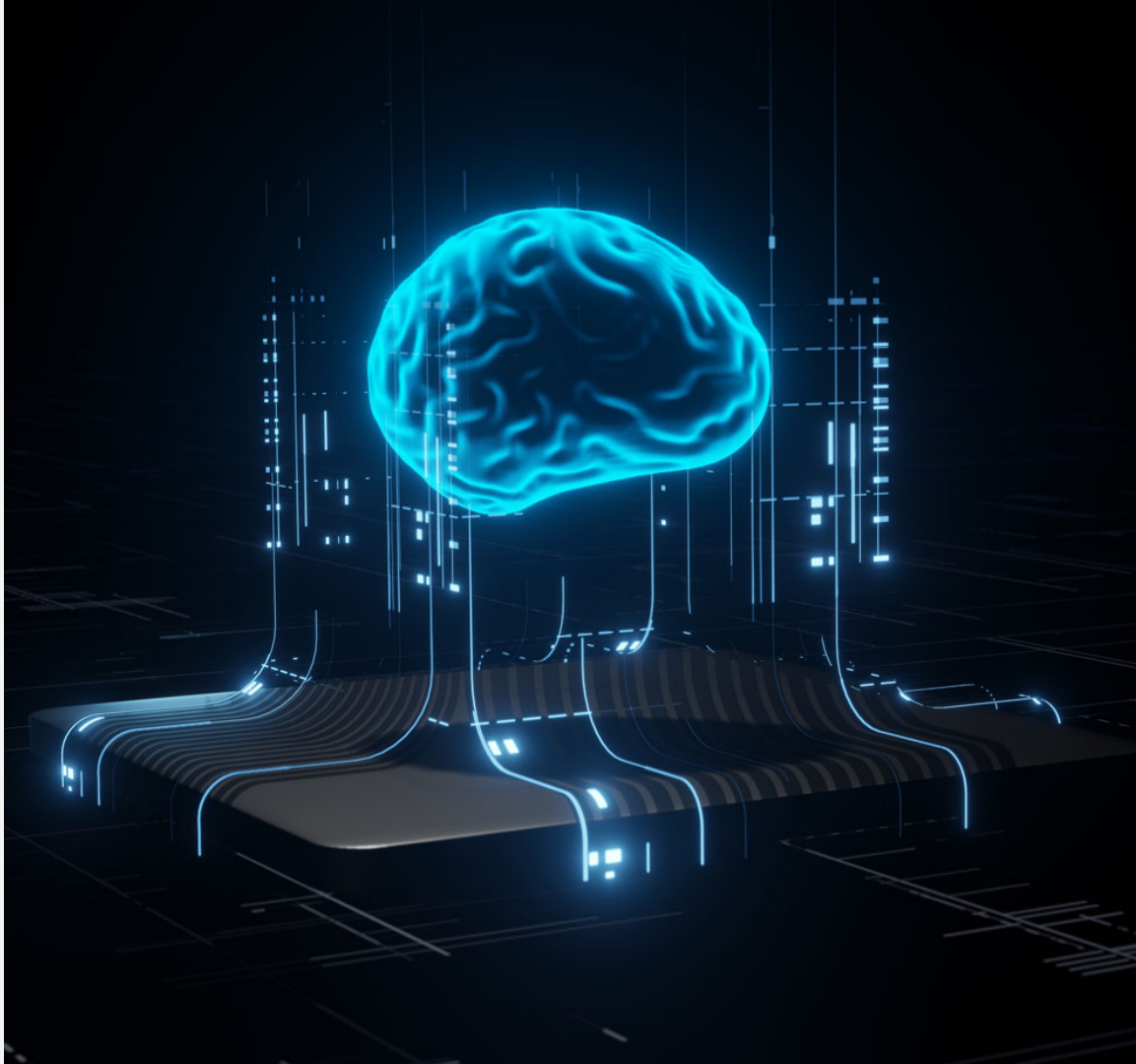
## Setup

### 2 Amazon EC2 Instances

- x1e.32xlarge
- Placed in Frankfurt
- 4x Intel Xeon E7 8880 v3
- 3904 GB DDR4 RAM
- AVX-2 Instructions

### Client: MacBook Pro

- Located in Darmstadt
- 10 ms latency



# C3 Benchmarks

## PIR Part

### Database

- 2 billion 32 byte entries
- 65,3 GB data
- After compression: 57,7 GB data
- 12% Compression

- Around 2 hours generation time
- 14,6 minutes to compute 100 (seed, value)-pairs

### Runtime

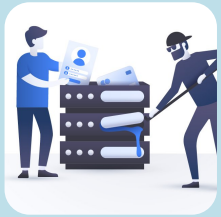
- Total: 13 seconds
- 9 seconds for XOR operations
- GPC has a total runtime of 8 seconds on average [Thomas et al, USENIX Security'19]

### Communication

- Total: 1,8 MB
- 1,7 MB is the size of the downloaded block
- PSI part takes additionally 64 Bytes of communication
- Less communication compared to GPC since our blocks are compressed

# Summary

## Our Contributions



C3 scheme

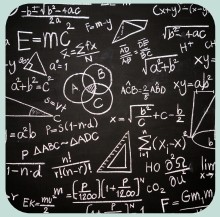
QDP PIR Model

QDP-RAID-PIR Implementation

## PIR Applications

C3


Private Messaging App




Private Set Inclusion

## Results

**Private Information Retrieval (PIR)**

- Up to (n-1)/n online computation improvement over RAID-PIR
- Not robust
- Communication and Storage only slightly affected

**Compromised Credential Checking (C3)**

- GPC: 8 seconds
- Ours: 13 seconds + overhead for PSI
- Perfect anonymity
- Secure against malicious server and client

# Thank you

Daniel Günther

TU Darmstadt

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

# Compromised Credential Checking (C3)

## Four Query Types

### Username

+ server does not learn any information about the password

- is the password really attacked?

### Password

+ user gets notified about all his attacked passwords

- too over-cautious [Thomas et al, USENIX Security'19]

    - knowledge of passwords often requires knowledge of username

### Domain

- what if the user is not active anymore?

- is the password really attacked?

### Username + Password

+ only attacked accounts are alerted

- user does not get notified for similar usernames

# Compromised Credential Checking (C3)

## Fix C3

**Frequency-Smoothing Bucketization (FSB) [Li et al., ACM CCS'19]**

- query only passwords
- passwords are assigned to blocks depending on their frequencies
  - more frequent passwords are assigned to many blocks
  - conditional probability of frequent passwords is now similar to those of less frequent passwords

- Better protection against CSA, but still leaks information

**C3 with Private Information Retrieval (PIR) [this thesis]**

- proposed by Thomas et al. [USENIX Security'19] and Li et al. [ACM CCS'19]
  - both say that PIR is too inefficient

- We show that PIR is efficient enough for C3 by introducing a new PIR model

**Optimal Blocksize:**

$$\sqrt{\frac{32(\#\text{Entries})}{\#\text{Server}}}$$

# PIR Benchmarks

## Runtime and Communication (WAN)

| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 265 | 13 | 12 | 8 | 2.53 | 2.59 |
| 1 000 000 | 4 000 | 26 | 64 | 34 | 8.00 | 8.06 |
| 10 000 000 | 12 650 | 89 | 182 | 89 | 25.30 | 25.36 |
| 50 000 000 | 28 285 | 189 | 638 | 389 | 56.57 | 56.63 |

**2 Server, redundancy parameter $r = 2$**

| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 033 | 7 | 16 | 6 | 3.10 | 3.16 |
| 1 000 000 | 3 265 | 19 | 56 | 19 | 9.80 | 9.89 |
| 10 000 000 | 10 372 | 61 | 313 | 82 | 30.85 | 31.21 |
| 50 000 000 | 23 095 | 182 | 1 146 | 283 | 69.28 | 69.38 |

**3 Server, redundancy parameter $r = 3$**

# PIR Benchmarks

## Runtime and Communication (LAN and WAN)

| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 033 | 2 | 6 | 3 | 3.10 | 3.16 |
| 1 000 000 | 3 265 | 5 | 72 | 26 | 9.80 | 9.89 |
| 10 000 000 | 10 372 | 8 | 381 | 132 | 30.85 | 31.21 |
| 50 000 000 | 23 095 | 14 | 1 287 | 425 | 69.28 | 69.38 |

**3 Server, redundancy parameter $r = 3$, LAN**

| Entries (32 bytes) | Blocksize (bytes) | Communication (ms) | Computation (ms) | | Upload (kB) | Download (kB) |
|---|---|---|---|---|---|---|
| | | | RAID-PIR | QDP RAID-PIR | | |
| 100 000 | 1 033 | 7 | 16 | 6 | 3.10 | 3.16 |
| 1 000 000 | 3 265 | 19 | 56 | 19 | 9.80 | 9.89 |
| 10 000 000 | 10 372 | 61 | 313 | 82 | 30.85 | 31.21 |
| 50 000 000 | 23 095 | 182 | 1 146 | 283 | 69.28 | 69.38 |

**3 Server, redundancy parameter $r = 3$, WAN**

# PIR Complexity

## Online Computation

$r$ - redundancy parameter
$B$ - number of blocks
$b$ - blockige (bytes in one block)
$n$ - number of servers
$c$ - number of chunks
$t$ - groupsize

| | Server | Client |
|---|---|---|
| **RAID-PIR** | $\frac{rB}{tc}$ | $Br + (n-1)b$ |
| **QDP-RAID-PIR** | $\frac{B}{tc}$ | $Br + (n-1)b$ |