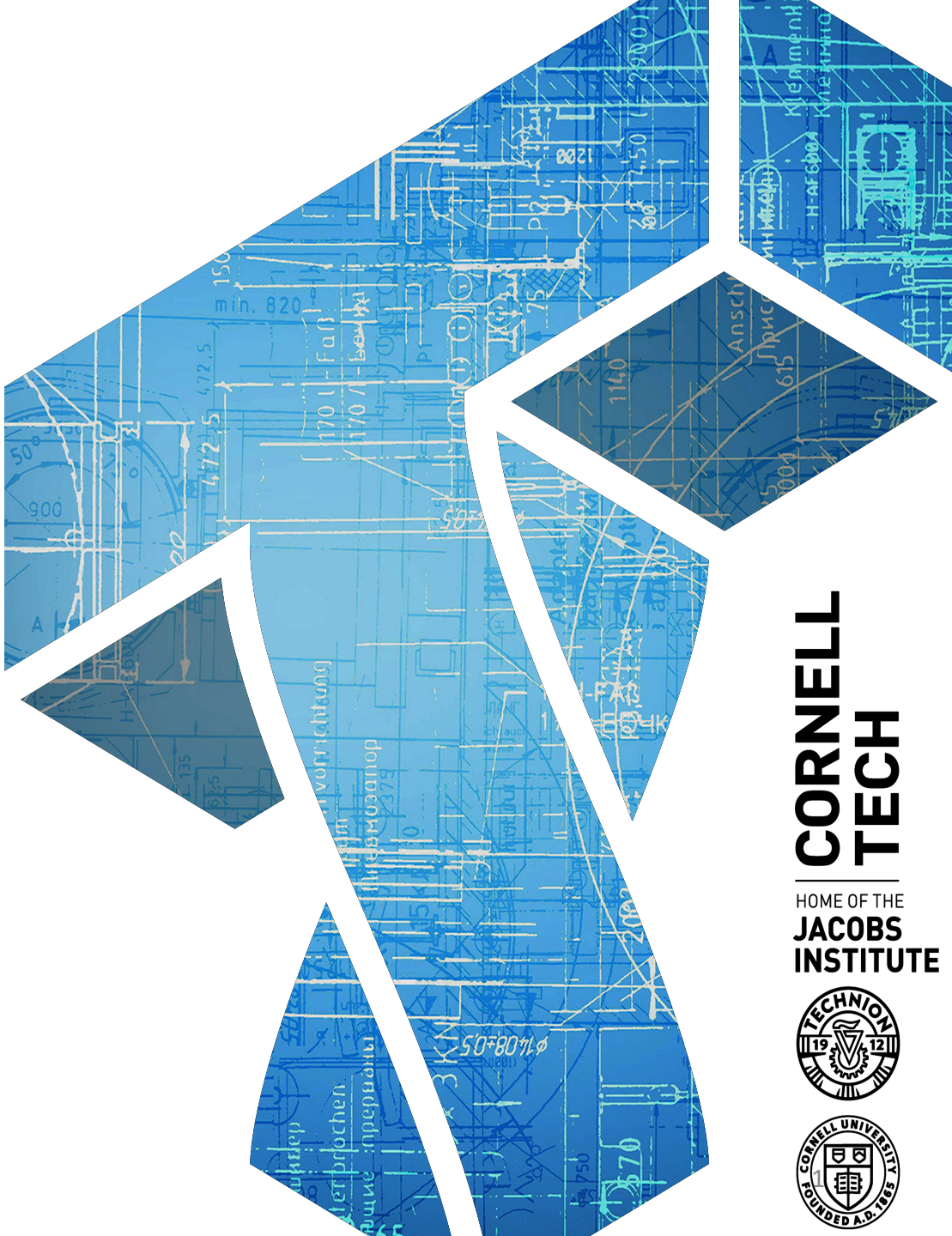
CS 6431: Encrypted Databases

Instructor: Tom Ristenpart

https://github.com/tomrist/cs6431-fall2021



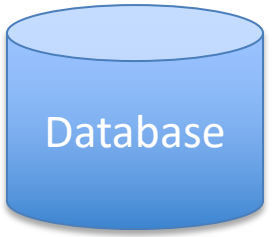
CORNELL TECH
HOME OF THE JACOBS INSTITUTE
TECHNION
CORNELL UNIVERSITY FOUNDED A.D. 1865

Minimizing data exposure risk

- Access controls
  - Least privilege principle applied to database
  - Monitoring & logging accesses
- Encrypted databases
  - Encryption at rest
  - Property-revealing encryption
- Data privacy protections
  - De-identification
  - K-anonymity
  - Differential privacy

Threat models for databases

Client
(e.g., laptop,
workstation,
web server)



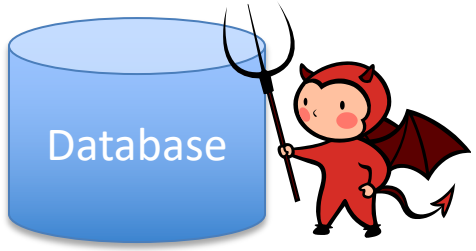
Person	Age	Income
Alice	22	100,000
Bob	29	25,000
Charlie	31	40,000

Malicious client accesses:

- Missing / misconfigured access control
- Break-in to authenticated client
- SQL injection attacks

Threat models for databases

Client
(e.g., laptop,
workstation,
web server)

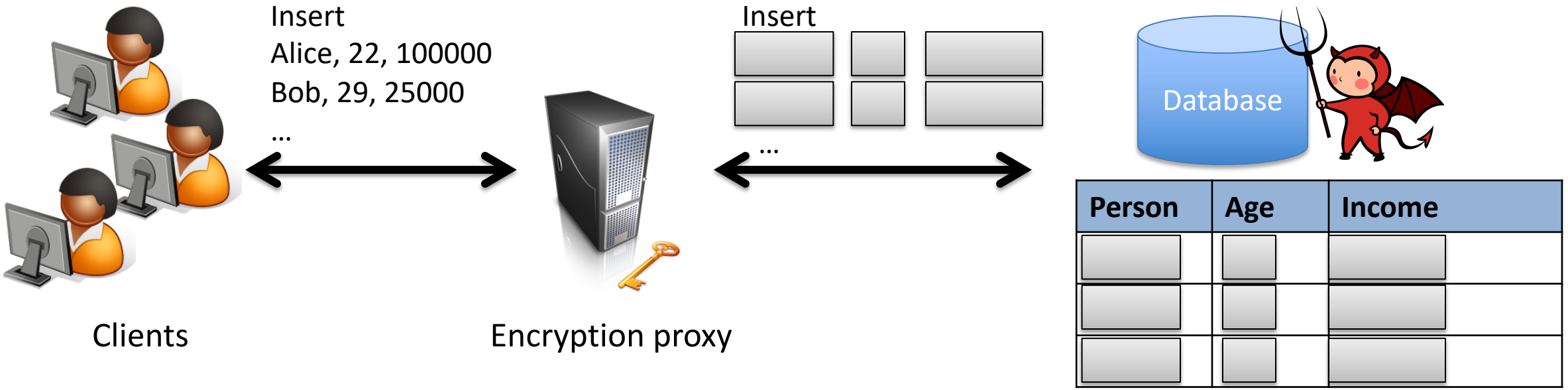


Person	Age	Income
Alice	22	100,000
Bob	29	25,000
Charlie	31	40,000

Compromise of database system:

- Loss or theft of persistent storage (hard disks)
- Malicious insider attack
- Subpoena / government access
- Complete compromise of system hosting database

One approach: database encryption



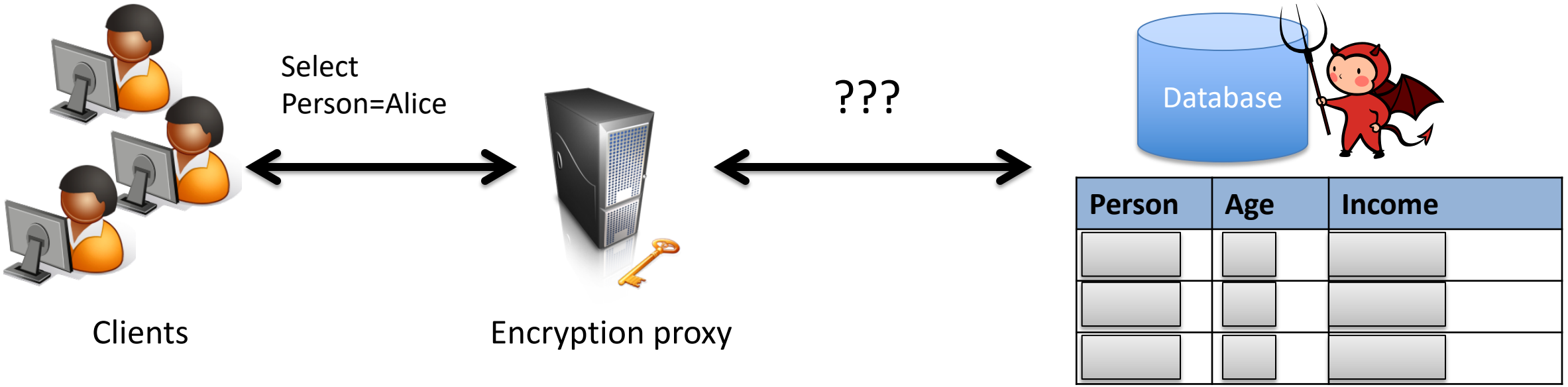
Encrypt data using secret key before insertion into database

Decrypt records fetched via queries

Encryption vs. DB functionality tension!
How can we fetch encrypted records?

Proxy can handle many clients (e.g., entire organization)

One approach: database encryption



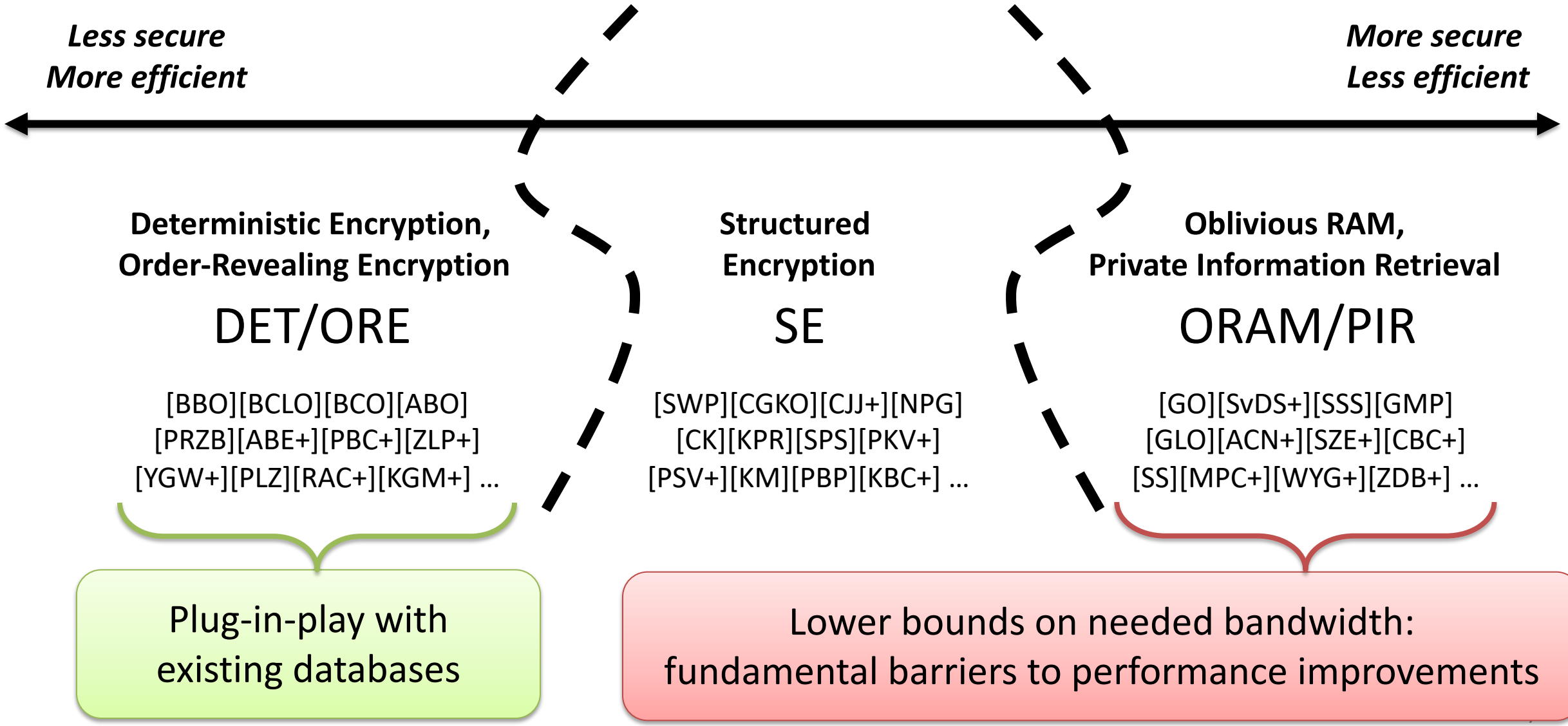
Encrypt data using secret key before insertion into database

Decrypt records fetched via queries

Encryption vs. DB functionality tension!
How can we fetch encrypted records?

Proxy can handle many clients (e.g., entire organization)

Decades of academic work on this tension

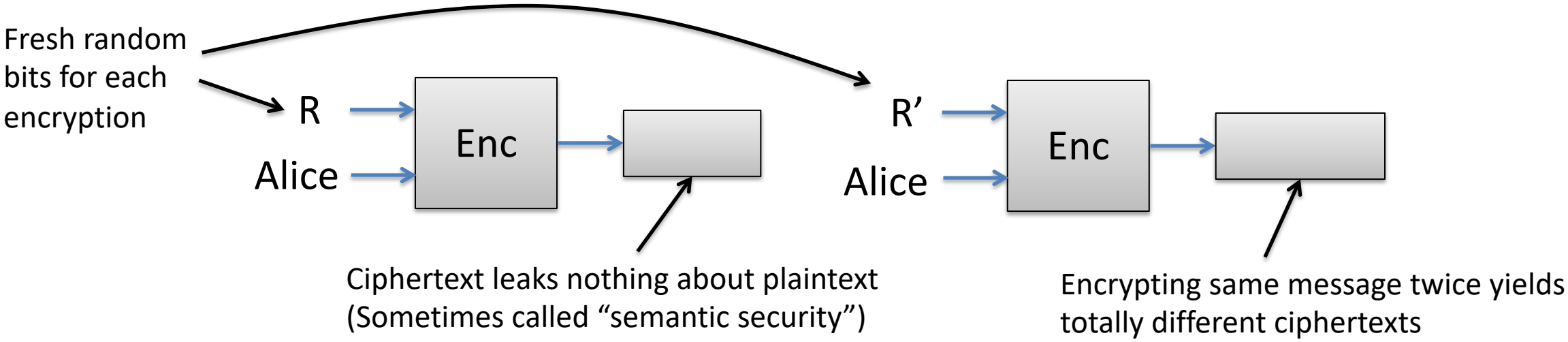


Deterministic encryption (DET)

The goal:

Encryption ***reveals plaintext equality*** to allow efficient record search

Conventional encryption *does not* reveal plaintext equality, by design:



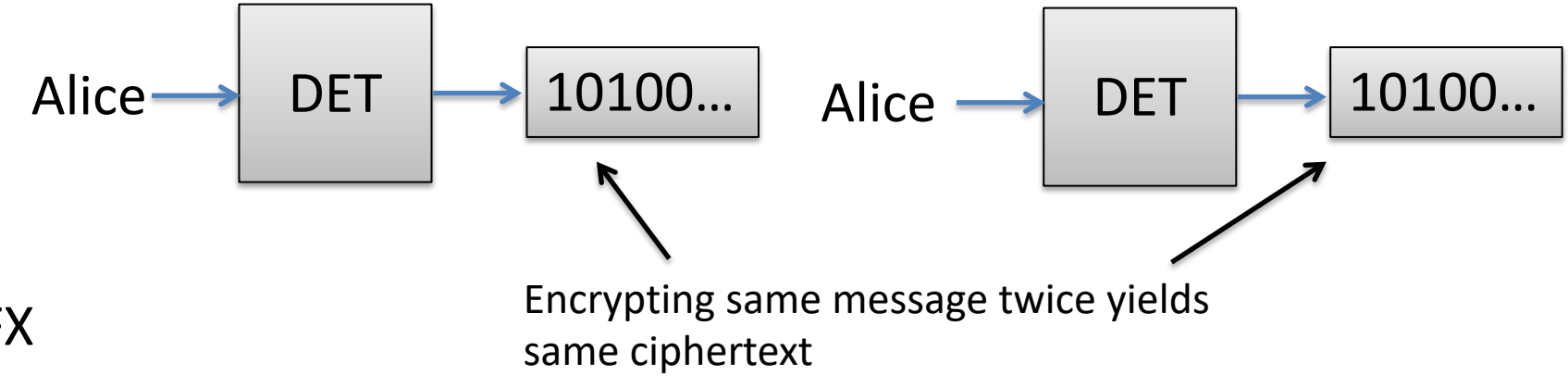
Deterministic encryption (DET)

The goal:

Encryption ***reveals plaintext equality*** to allow efficient record search

Deterministic encryption: same plaintext maps to same ciphertext

DET(M) = DET(M') iff M = M'



Examples:
SIV mode, FFX

Leaks plaintext equality Allows equality search (find all rows with name "Alice")

Using DET in encrypted DBs

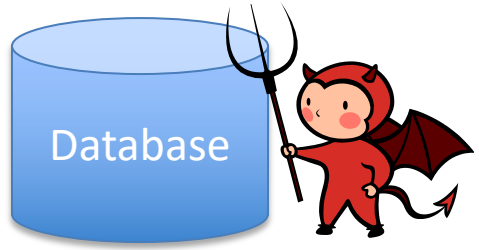
Clients

Insert
Alice, 22, 100000
Bob, 29, 25000
…

Proxy using DET for Person column

Insert

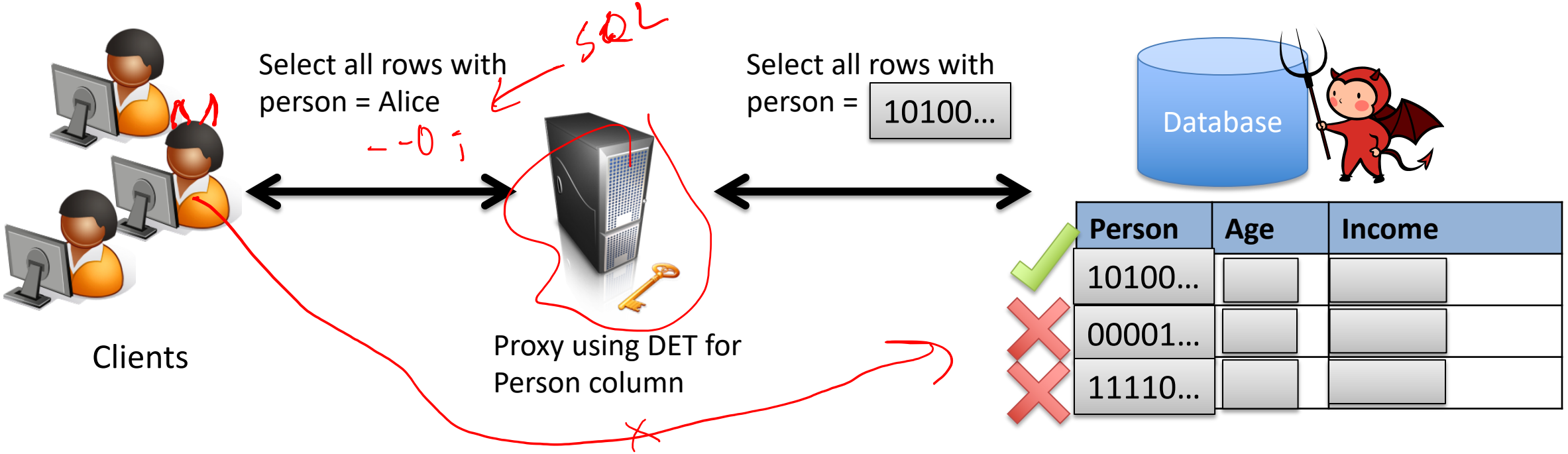
10100…		
00001…		
…		



Person	Age	Income
10100…		
00001…		
11110…		

10100…	= DET(Alice)
00001…	= DET(Bob)
11110…	= DET(Charlie)

Using DET in encrypted DBs



Proxy recomputes C1 by encrypting search term Alice

10100… = DET(Alice)

Deterministic encryption enables fast retrieval of matching rows, but…

ciphertexts leak plaintext equality, allowing ***leakage abuse attacks***

CryptDB paper [Popa et al. 2011]

- One of early academic works to suggest using property-revealing encryption for SQL-style DBs
  - "CryptDB's approach is to execute queries over encrypted data"
- Popularized idea, particularly in academic circles & tech media
  - Described as efficient alternative to fully homomorphic encryption
- Combine existing DET, OPE schemes (plus one new one for database joins) with onioning (wrapping with randomized encryption)

Companies have deployed DE and/or ORE

~~Sometimes referred to as enabling "*computing on encrypted data*"~~

Encryption ***reveals just enough*** about plaintexts to perform specific DB operations

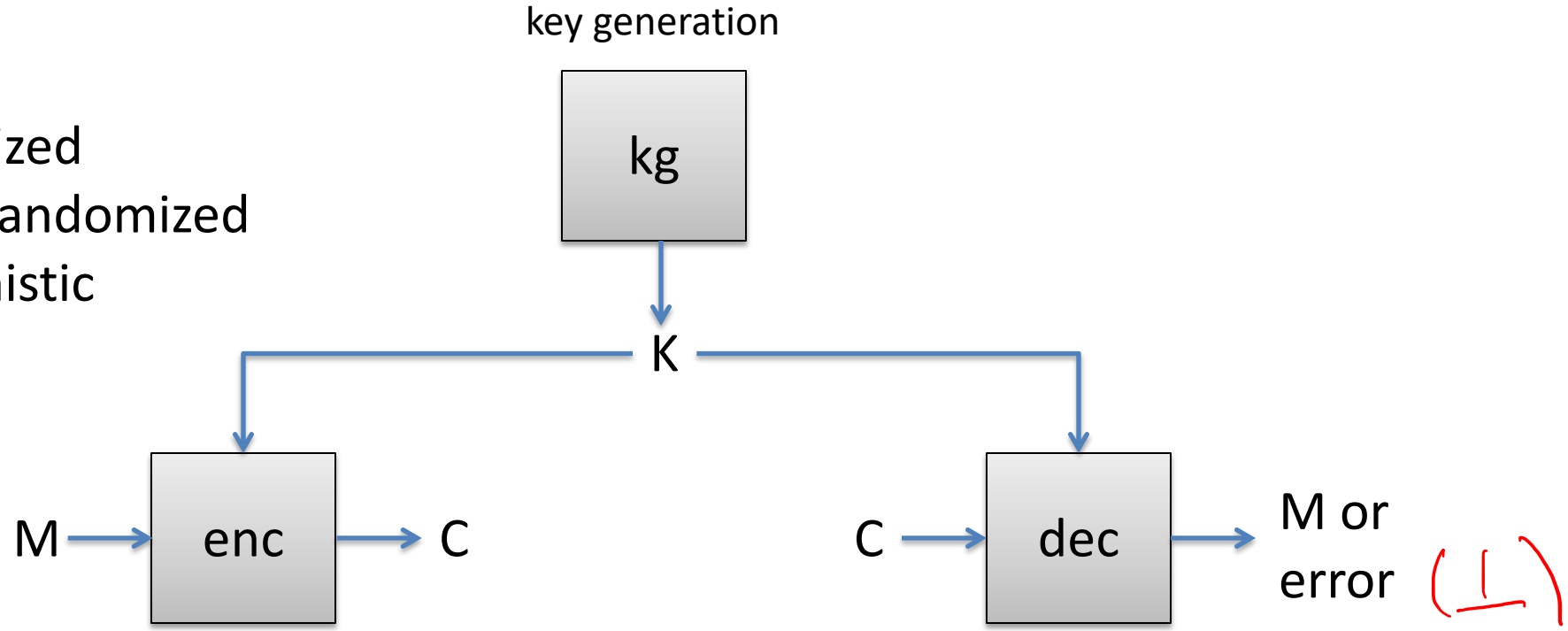
"***Property-revealing encryption***"

Formal security analyses ***proving security****

* for definitions of security that allow leaking information about plaintexts

Symmetric encryption (SE) definition

kg is randomized
enc ***may*** be randomized
dec deterministic



C is a ciphertext

Correctness: Pr[dec(K , enc(K,M)) = M] = 1 , probability over enc's randomness

Write $C \leftarrow\!\!\$\ \mathrm{enc}_K(M)$ to signify running enc with fresh randomness

Assume |C| = clen(|M|) for any M

Real-or-random indistinguishability (IND$)

Standard security goal for symmetric encryption

Can't distinguish encryption of chosen messages from random string of bits

$\underline{\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}}}$
$K \xleftarrow{\$} \mathsf{kg}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathrm{Enc}}$
Ret b'

$\underline{\mathrm{Enc}(M)}$
$C \xleftarrow{\$} \mathsf{enc}_K(M)$
Ret C

$\underline{\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}}}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathrm{Enc}}$
Ret b'

$\underline{\mathrm{Enc}(M)}$
$C \xleftarrow{\$} \{0,1\}^{\mathsf{clen}(|M|)}$
Ret C

Adversary can repeatedly query Enc on same message

OTP encryption

$\mathsf{enc}_K(m) = K \oplus m$

$\mathsf{dec}_K(C) = C \oplus K$

Adversary is a randomized algorithm with access to Enc oracle

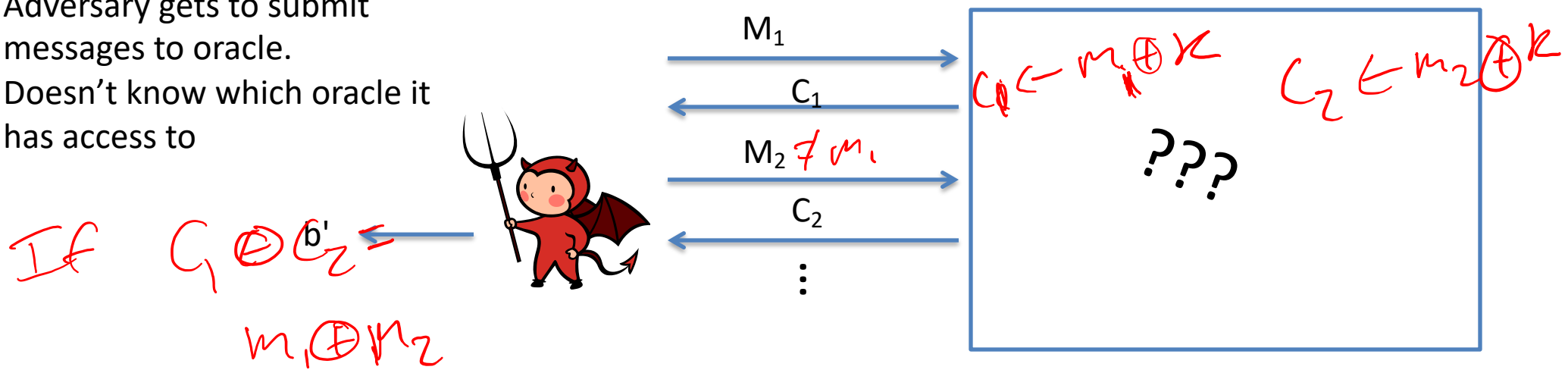
Advantage function measures efficacy of an adversary:

$$\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}} \Rightarrow 1\,\right]\right|$$

Probabilities are over random coins used in games

Real-or-random indistinguishability (IND$)

Adversary gets to submit messages to oracle.
Doesn't know which oracle it has access to



Intuition:

Ciphertexts are indistinguishable from random bits. Contain no information about plaintext

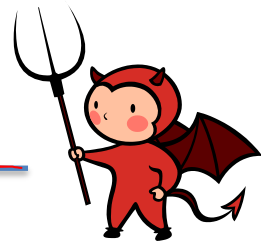
- Such information-theoretic security only possible if |K| = |M| (e.g., one-time pad)

Instead, restrict to ***computationally efficient*** adversaries

- Breakthrough paradigm shift from 1980s (e.g., Goldwasser, Micali paper)

Deterministic encryption is IND$-insecure

Adversary gets to submit messages to oracle.
Doesn't know which oracle it has access to



M_1
C_1
$M_2 = m_1$
C_2
⋮

$C_i \leftarrow \text{enc}(M_i)$
???

$C_i \leftarrow_{\$} \{0,1\}^{\text{clen}(|M_i|)}$

How can adversary choose queries in order to distinguish between REAL and RAND?
REAL: getting back deterministic encryptions of plaintext
RAND: getting back randomly chosen bit string no matter what is input

Real-or-random indistinguishability for deterministic encryption

$\underline{\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}}}$
$K \leftarrow\!\!\$\ \mathsf{kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret b'

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \mathsf{enc}_K(M)$
Ret C

$\underline{\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}}}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret b'

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
Ret C

Adversary can repeatedly query Enc on same message

Real-or-random indistinguishability for deterministic encryption

$\underline{\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}}}$
$K \leftarrow\!\!\$\ \mathsf{kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret b'

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \mathsf{enc}_K(M)$
Ret C

$\underline{\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}}}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret b'

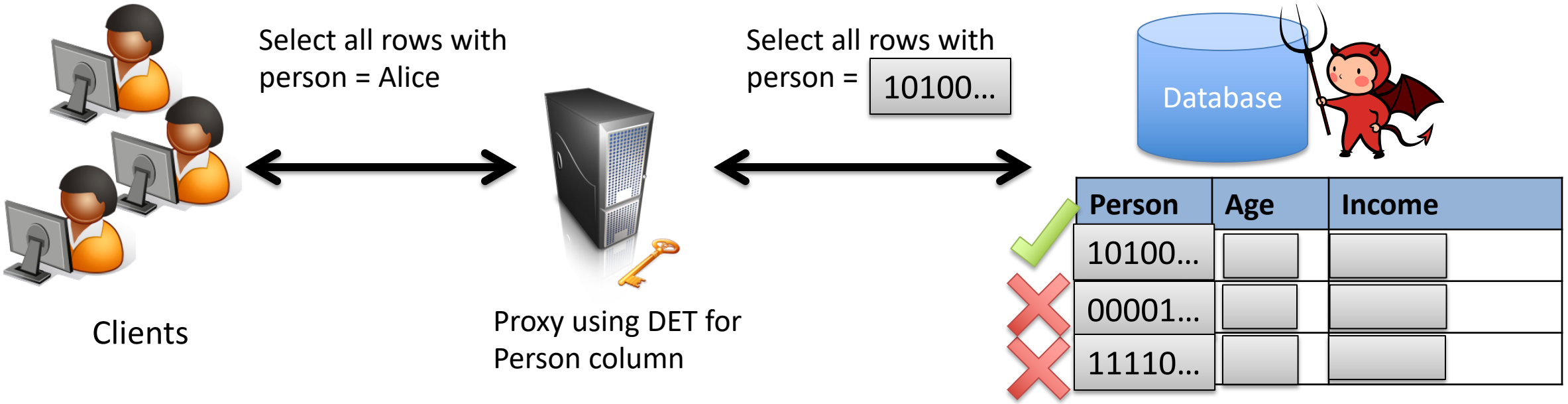
$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
Ret C

Adversary ***cannot*** repeatedly query Enc on same message

Can formalize via pseudocode changes to game or as restriction on class of adversaries considered

Can show schemes that achieve this: Synthetic IV (SIV) from [Rogaway, Shrimpton 2006]
What does this notion of security allow in terms of plaintext leakage?

Using DET in encrypted DBs



Proxy recomputes C1 by encrypting search term Alice

10100… = DET(Alice)

Deterministic encryption enables fast retrieval of matching rows, but…

ciphertexts leak plaintext equality, allowing ***leakage abuse attacks***

Leakage-Abuse Attacks against DET

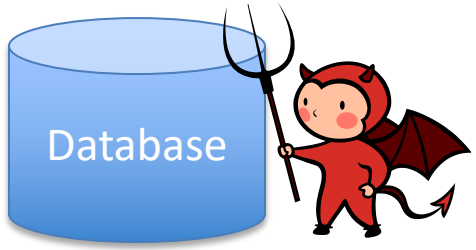
[Naveed, Kamara, Wright 2015]

Clients

Select all rows with person = Alice

Proxy using DET for Person column

Select all rows with person = 10100…



Person	Age	Income
10100…		
00001…		
11110…		
10100…		
00001…		
11110…		
11110…		
10100…		
00001…		
11110…		
11110…		

Adversary can generate histogram of ciphertexts

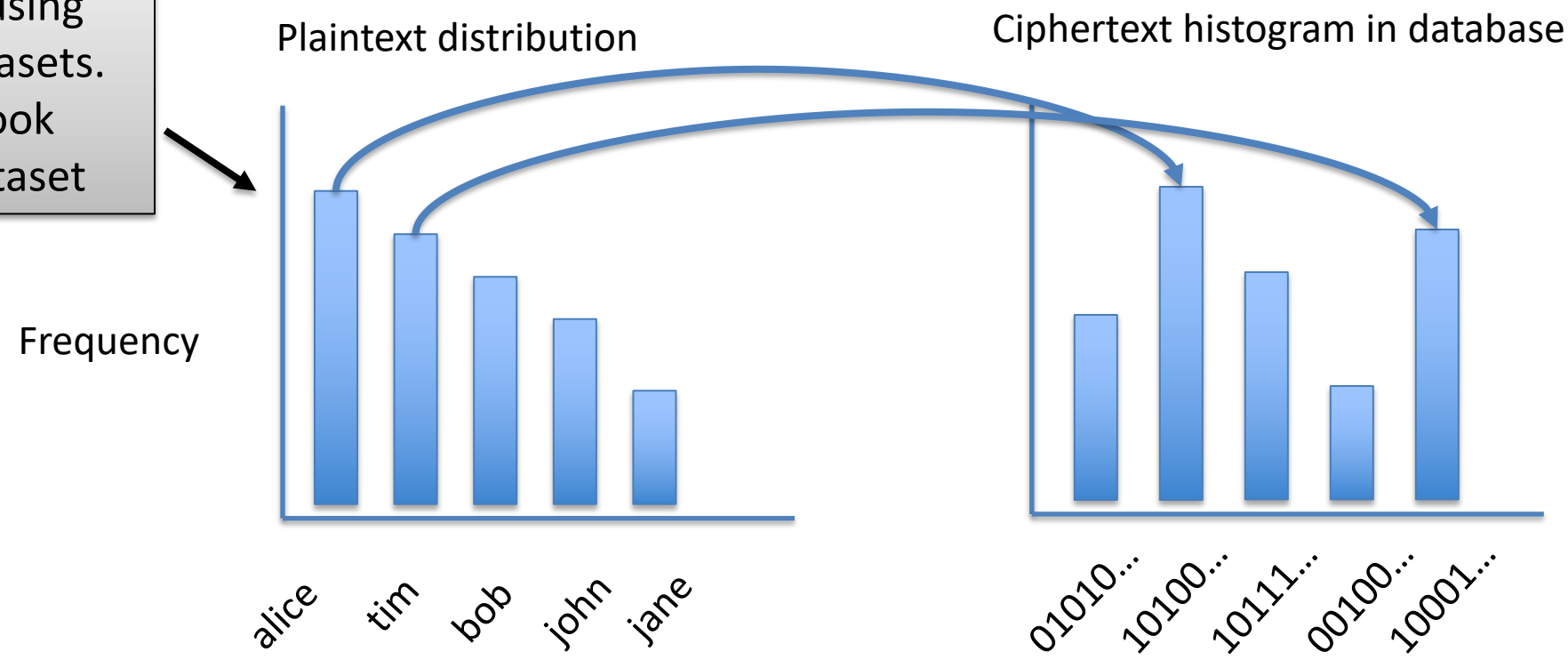
Given some info on plaintext distribution, use ***frequency analysis*** to recover plaintexts

Frequency attacks against DET

Idea: if attacker knows something about distribution of plaintexts, we can exploit it

[Al-Kindi 9th century BCE] [NKW 2016]

Estimate using public datasets. Ex: Facebook names dataset



For deterministic encryption this is all information adversary gets from database

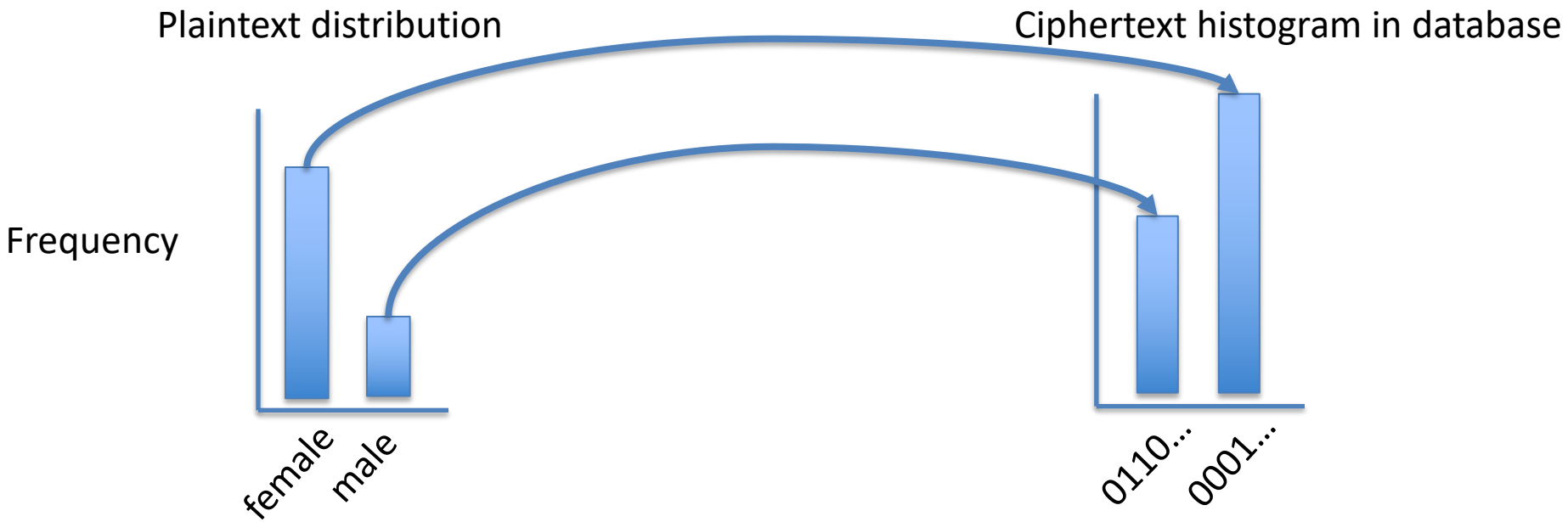
Frequency analysis optimal attack [Lacharite, Paterson 2015]

DET frequency analysis can be very effective

Non-uniform, small-domain values easy for adversary to recover

[Bindschaedler et al. 2018] simulated attacks:
Recovers 100% of DET-encrypted values in hospital datasets

Most columns binary or small number of possibilities



DET frequency analysis can be very effective

Non-uniform, small-domain values easy for adversary to recover

[Bindschaedler et al. 2018] simulated attacks:
Recovers 100% of DET-encrypted values in hospital datasets

Most columns binary or small number of possibilities

More uniform, larger-domain values can be harder for adversary to recover

[Naveed et al. 2016] simulated attacks:
Recovers 10% of DET-encrypted age values in hospital datasets

Data closer to uniform
Good plaintext distribution estimates harder to get

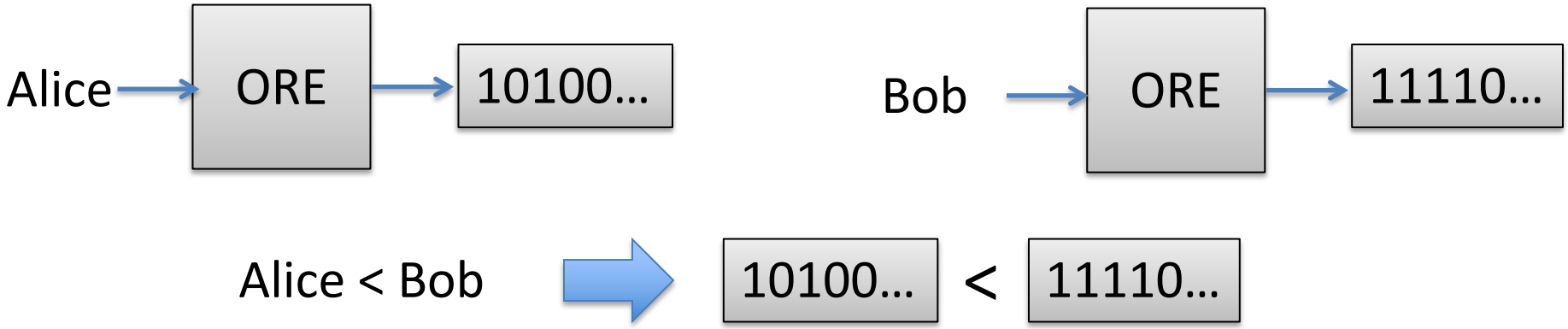
Order-revealing encryption (ORE)

The goal:

Encryption ***reveals plaintext equality and ordering*** to allow efficient range searches

Deterministic encryption such that ciphertexts reveal plaintext order

$$\text{ORE}(M) \leq \text{ORE}(M') \text{ iff } M \leq M'$$



Practical schemes by: [Boldyreva, Chennette, Lee, O'Neill 2009] (BCLO)
[Chenette, Lewi, Weis, Wu 2015] (CLWW)

CLWW construction

Uses a pseudorandom function (PRF), e.g.: $F_K(X)$ = HMAC(K,X) for secret key K

Let n-bit input message be $M = M_1 M_2 \ldots M_n$

$$u_i = F_K(i \| M_1 M_2 \cdots M_{i-1} \| 0^{n-i}) + M_i \bmod 3$$

Ciphertext is $u_1, u_2, \cdots, u_n$

Given two ciphertexts $u_1, u_2, \cdots, u_n$ and $u'_1, u'_2, \cdots, u'_n$ how to infer ordering of M, M' ?

Find index j s.t. $u_j \neq u_j'$

If $u_j > u_j'$ then output M > M'

Otherwise output M' < M

Leaks more than order! What does it leak?

Trivial attacks against ORE: BCLO leakage

Some ORE schemes leak a lot more than just order of plaintexts

[BCO 11] showed that BCLO leaks about half of plaintext bits trivially

Plaintext	Ciphertext	m_c
michael	cyrzjipnouushzh	michaekypfbkfr
david	aenpse cevvpkmr	david jwbvhec
robert	emlqrnycvblqqnd	robert lwyeorr
john	ccnnczzzpruvjhd	johmzzzysfbunn
james	bzkxrq gzortby	james zyovtq
daniel	aelfspocabjdvjc	daniel jgaginu
richard	ekrzjmjhjxykbba	richardkmfnwwx
jose	ccqrlzzziozokby	josdzzzxvfruqg
mark	cwmlfzzzjxhlklh	marjzzzxzqyduv
christopher	zokwwbrbibyouo	christotnqfolw

Apply a few arithmetic operations to ciphertext

[Grubbs et al. 2016]

[Grubbs et al. 2017] attacks

New attack framework that combines:

frequency analysis + ordering leakage + scheme-specific leakage

Scheme(s)	First names	Last names
Kerschbaum [27]	26%	6%
Popa et al. [36], Kerschbaum [28]	**84%**	38%
BCLO [12, 13]	**99%**	**97%**
CLWW [18]	**98%**	**75%**
BCLO + CLWW [18]	**85%**	44%
Baseline Guessing	4%	1%

Percentage of row values recovered from encrypted database

Academic literature on leakage abuse attacks

Improvements on attacks

Varying leakage types

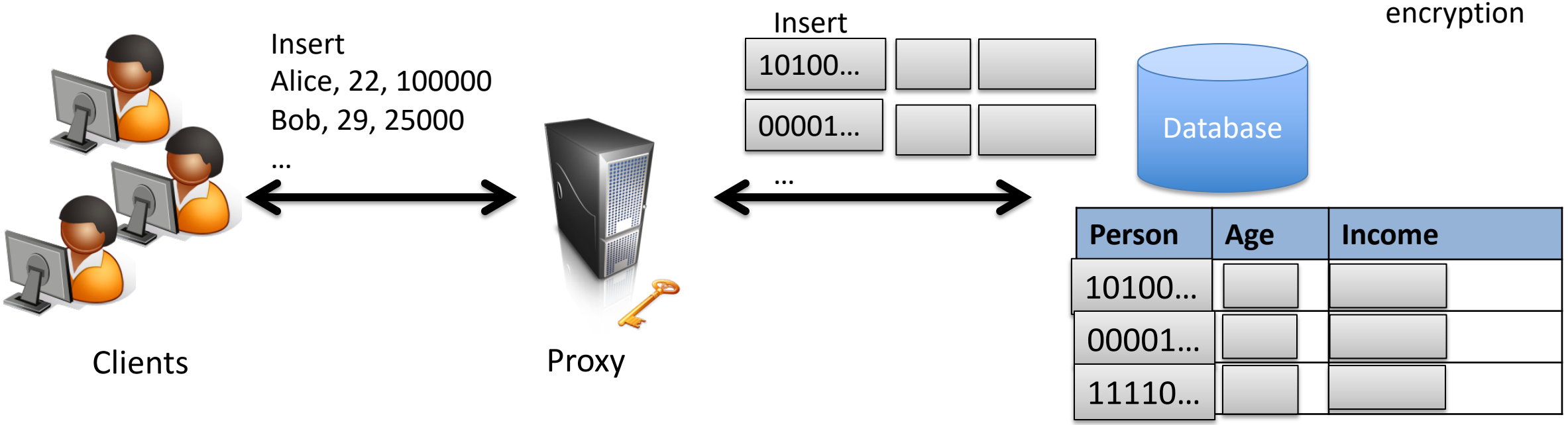
- Multiple-column leakage
- Nearest neighbor access patterns
- “Volume” leakage (# of values returned for range query)

Important for guiding practice:

[Islam, Kuzu, Kantarcioglu 2013]
[Cash, Grubbs, Perry, Ristenpart 2015]
[Naveed, Kamara, Wright 2015]
[Durak, DuBuisson, Cash 2016]
[Poillot, Wright 2016]
[Grubbs, McPherson, Naveed, Ristenpart, Shmatikov 2016]
[Zhang, Katz, Papamanthou 2016]
[Kellaris, Kollios, Nissim, O’Neill 2016]
[Grubbs, Sekniqi, Bindschaedler, Naveed, Ristenpart 2017]
[Grubbs, Ristenpart, Shmatikov 2017]
[Bindschaedler, Grubbs, Cash, Ristenpart, Shmatikov 2018]
[Grubbs, Lacharité, Minaud, Paterson 2018]
[Grubbs, Lacharité, Minaud, Paterson 2019]
[Kornaropoulos, Papamanthou, Tamassia 2019]
[Kornaropoulos, Papamanthou, Tamassia 2020]
…

Encryption wrapping (aka onioning)

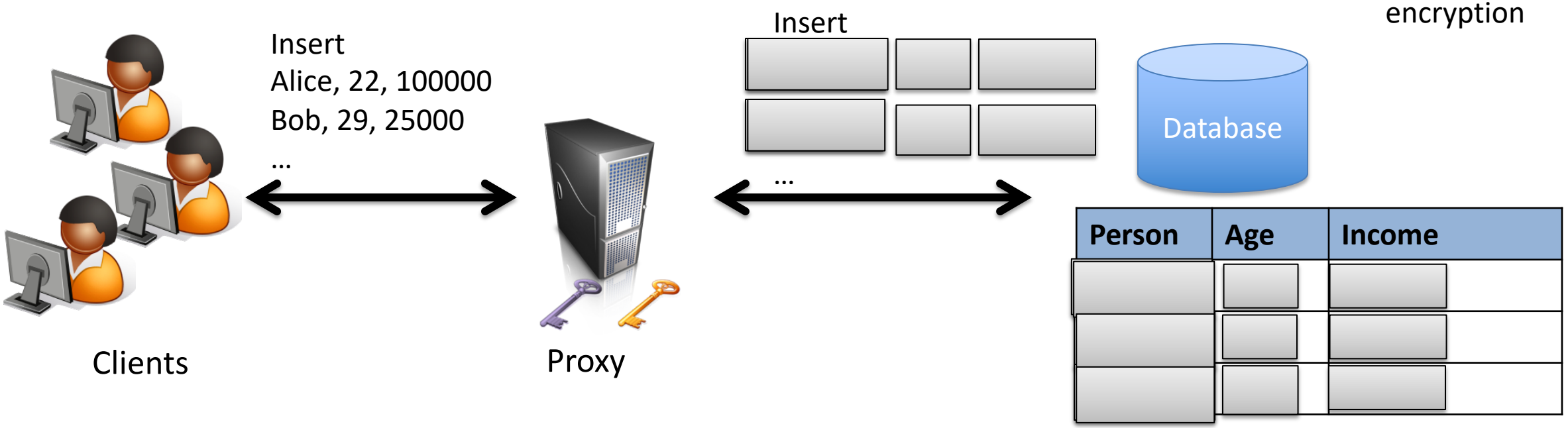
Example of structured encryption



10100…	=	DET(Alice)
00001…	=	DET(Bob)
11110…	=	DET(Charlie)

Encryption wrapping (aka onioning)

Example of structured encryption



= Enc(DET(Alice))

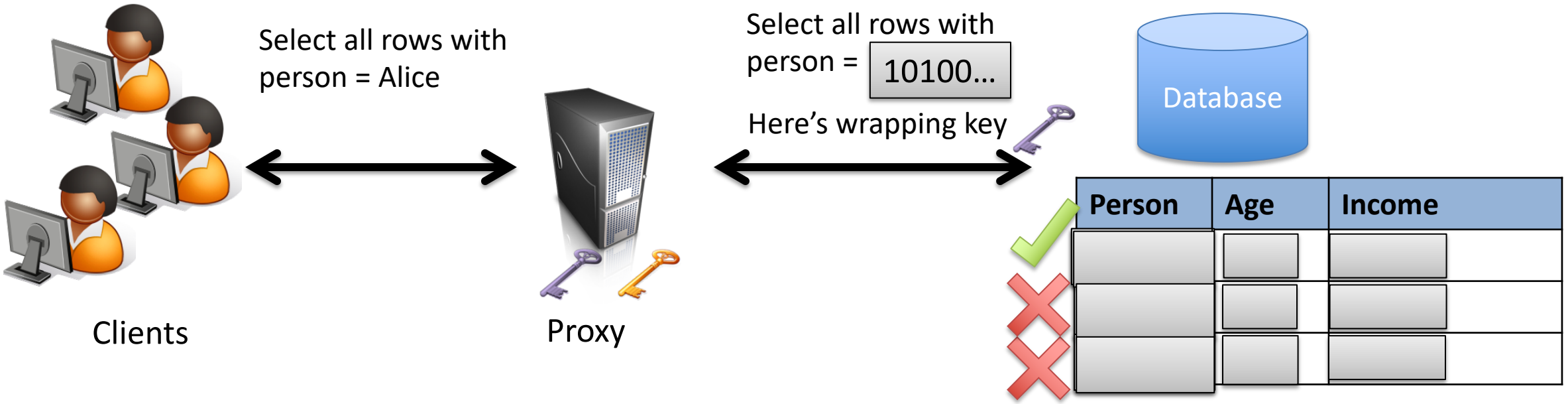
= Enc(DET(Bob))

= Enc(DET(Charlie))

Wrap inner weaker encryption with strong randomized encryption

Encryption wrapping (aka onioning)

Example of structured encryption



= Enc(DET(Alice))

= Enc(DET(Bob))

= Enc(DET(Charlie))

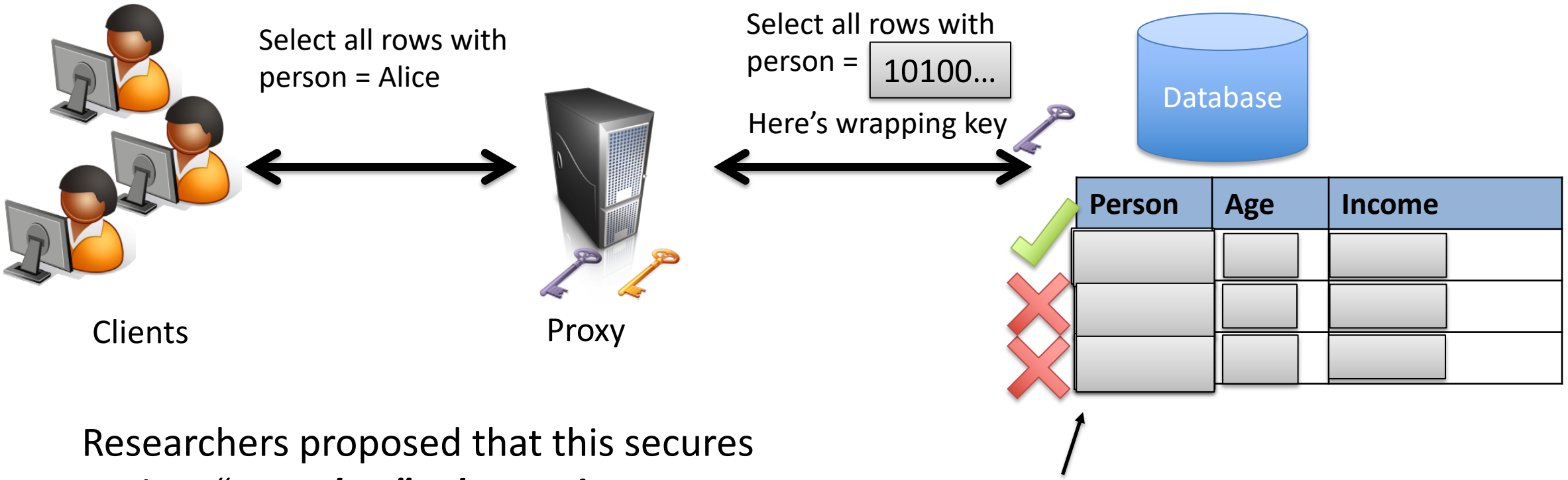
Wrap inner weaker encryption with strong randomized encryption

Server first unwraps (decrypts) column
Then looks for 10100…

Server can re-wrap (encrypt) column

Encryption wrapping (aka onioning)

Example of structured encryption



Researchers proposed that this secures against "***snapshot" adversaries***:

- Attackers that only obtain dump of database entries learn nothing

Server first unwraps (decrypts) column
Then looks for 10100…

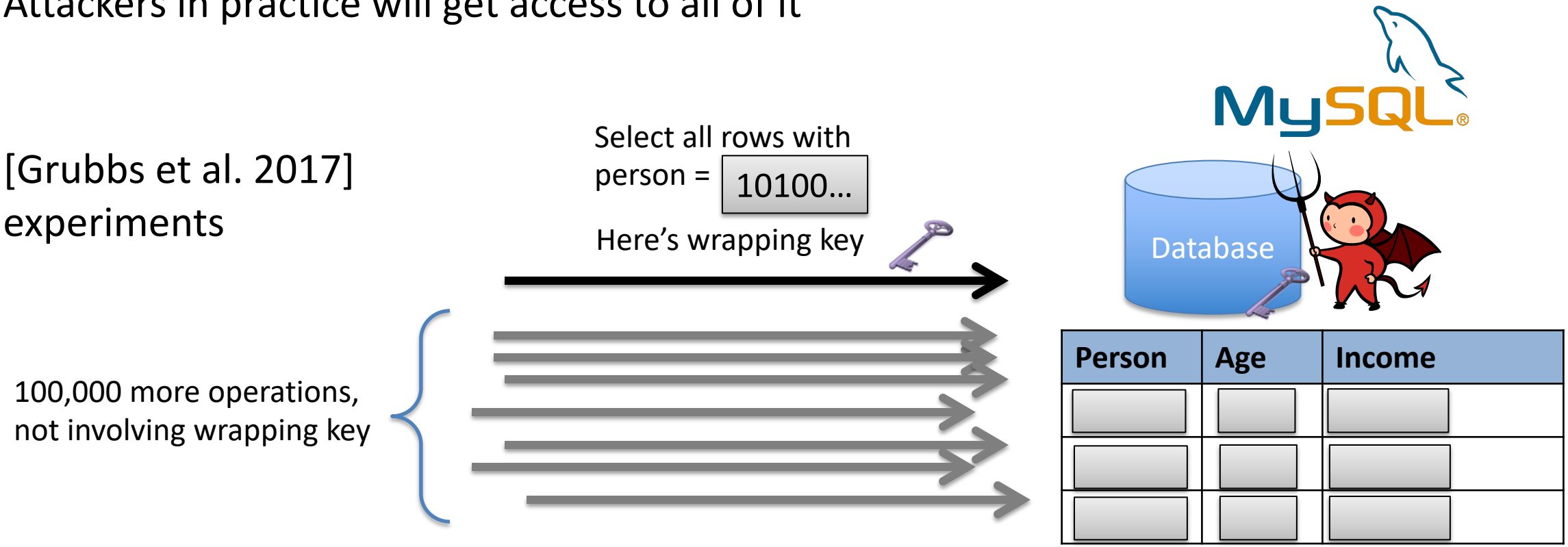
Server can re-wrap (encrypt) column

Currently: snapshot adversaries don't exist

Queries are left in memory, cached, and even written to logs in typical databases

Attackers in practice will get access to all of it

[Grubbs et al. 2017] experiments



Wrapping key still in DB's memory, available to attacker!

Encrypted DBs

- Big research area:
  - Property-revealing encryption
  - Systems designs
  - Systems security
  - Leakage-abuse attacks
- Used widely in practice
- Illustrative pitfalls in terms of “provable” security
  - “CryptDB is a system that provides practical and provable confidentiality …”