Deploying MPC in Open Finance: Challenges and Opportunities

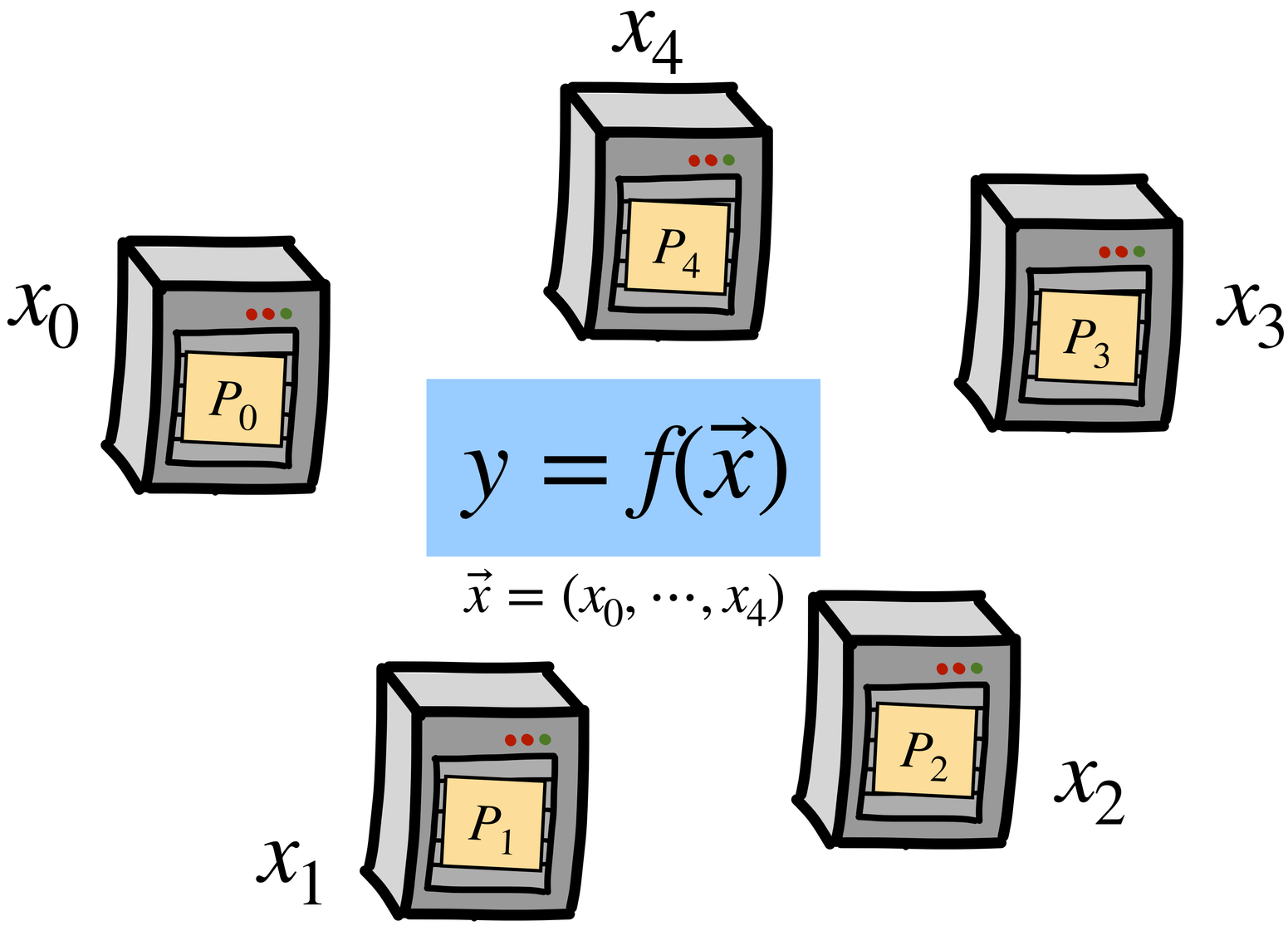
Nidhish Bhimrajka Yashvanth Kondi Daniel Noble Supreeth Varadarajan

S:L ENCE ABORATORIES

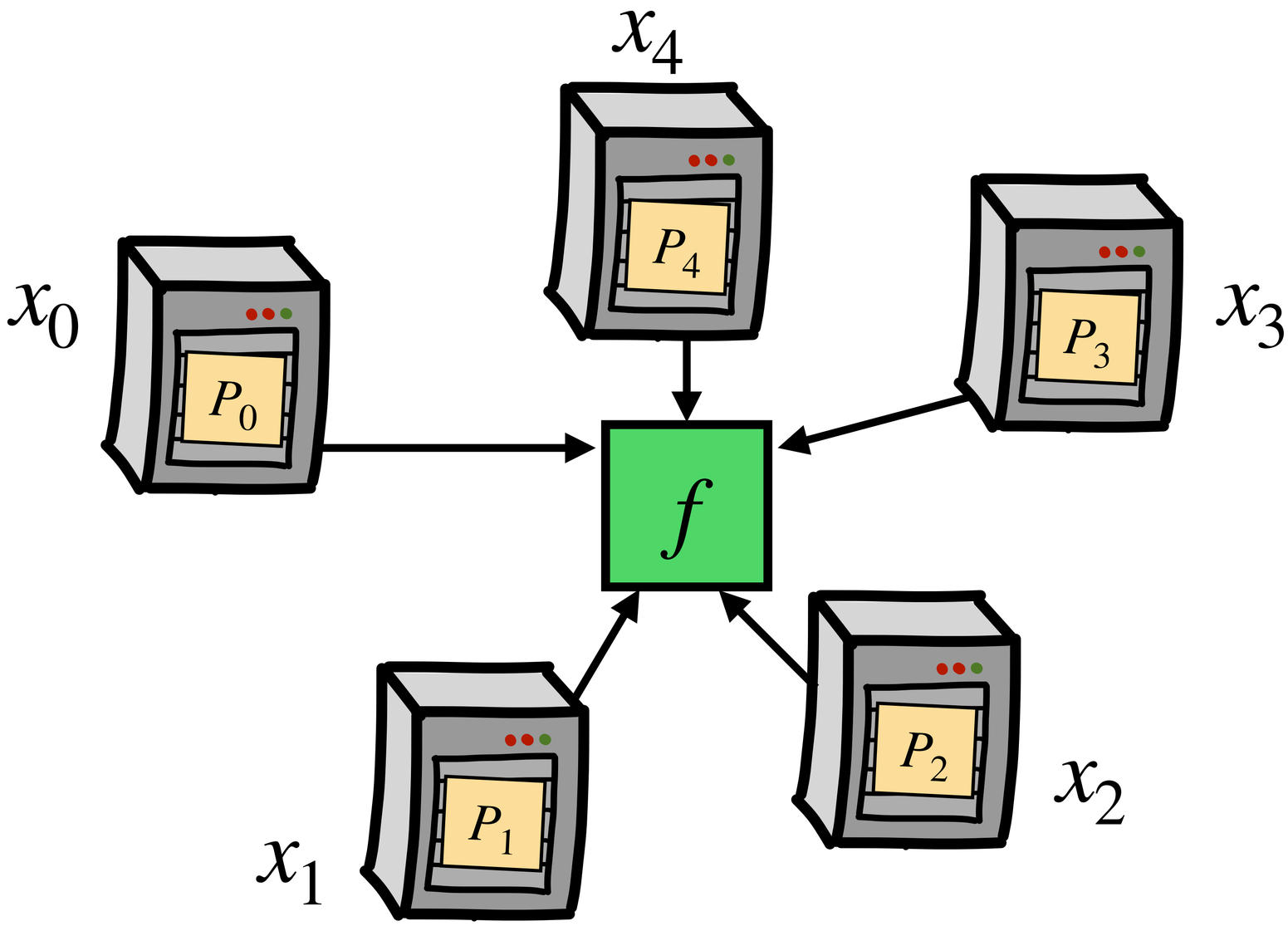
In this talk...

- Introduction to the Account Aggregator (AA) ecosystem, a regulator-driven Open Finance framework in India
- Identify gaps in trust and incentives in present ecosystem
- Propose an MPC-based solution which:
  - Mitigates data duplication concerns to reinstate trust
  - Facilitates fair compensation to balance incentives
- Discuss our design principle of drop-in replacement, and the technical challenges posed by integration with AA ecosystem

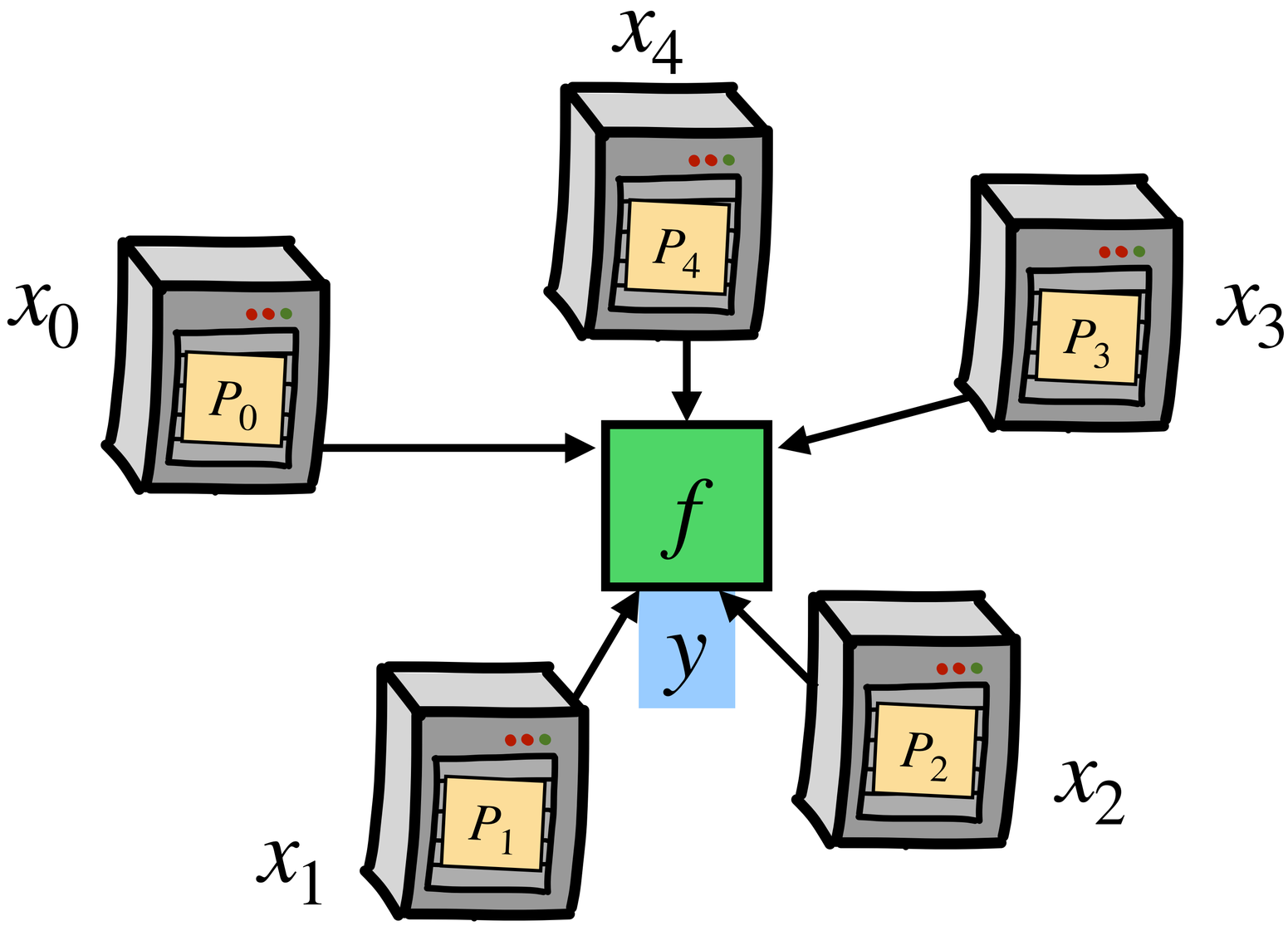
Secure Multiparty Computation (MPC)



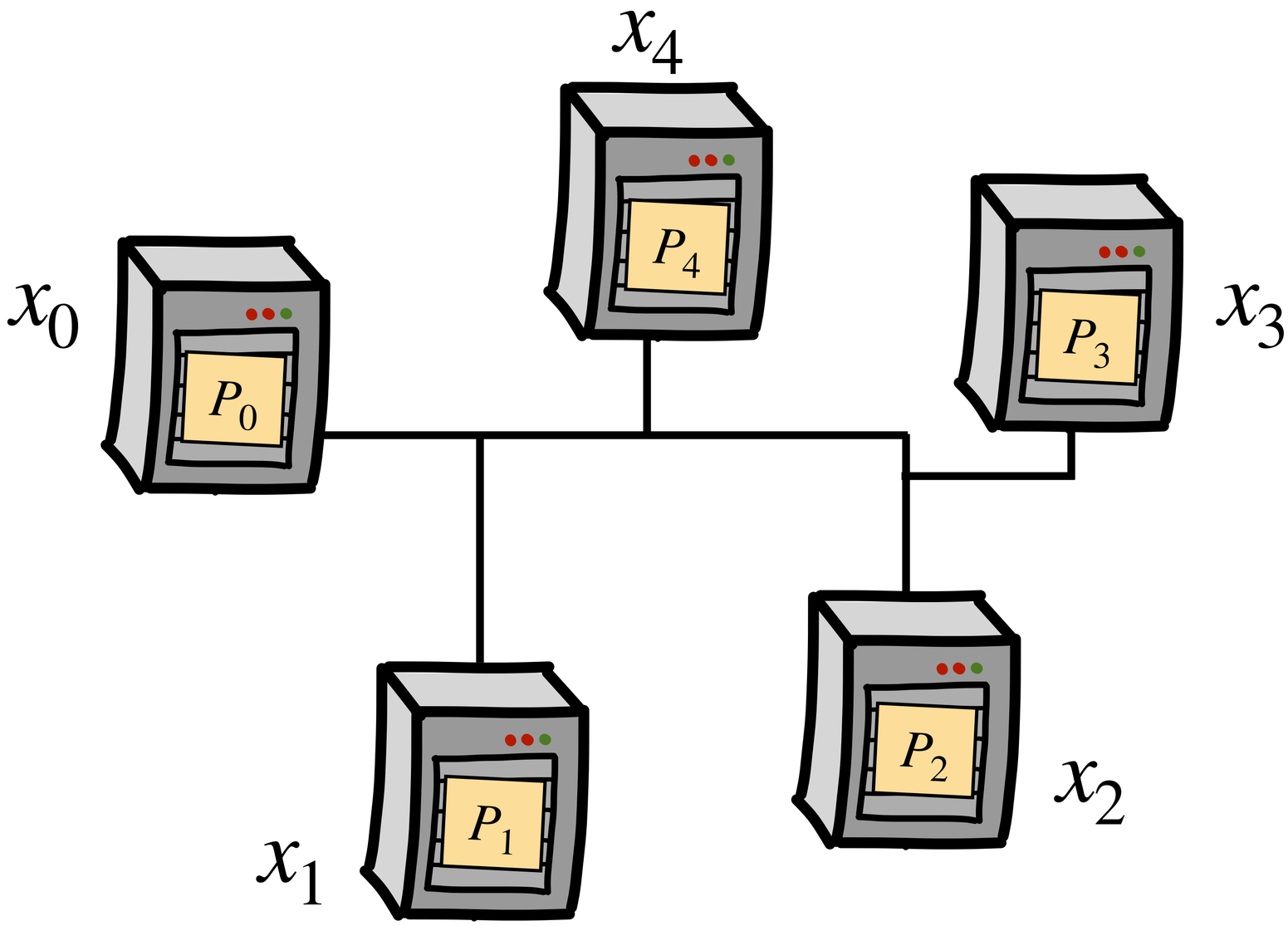
Secure Multiparty Computation (MPC)



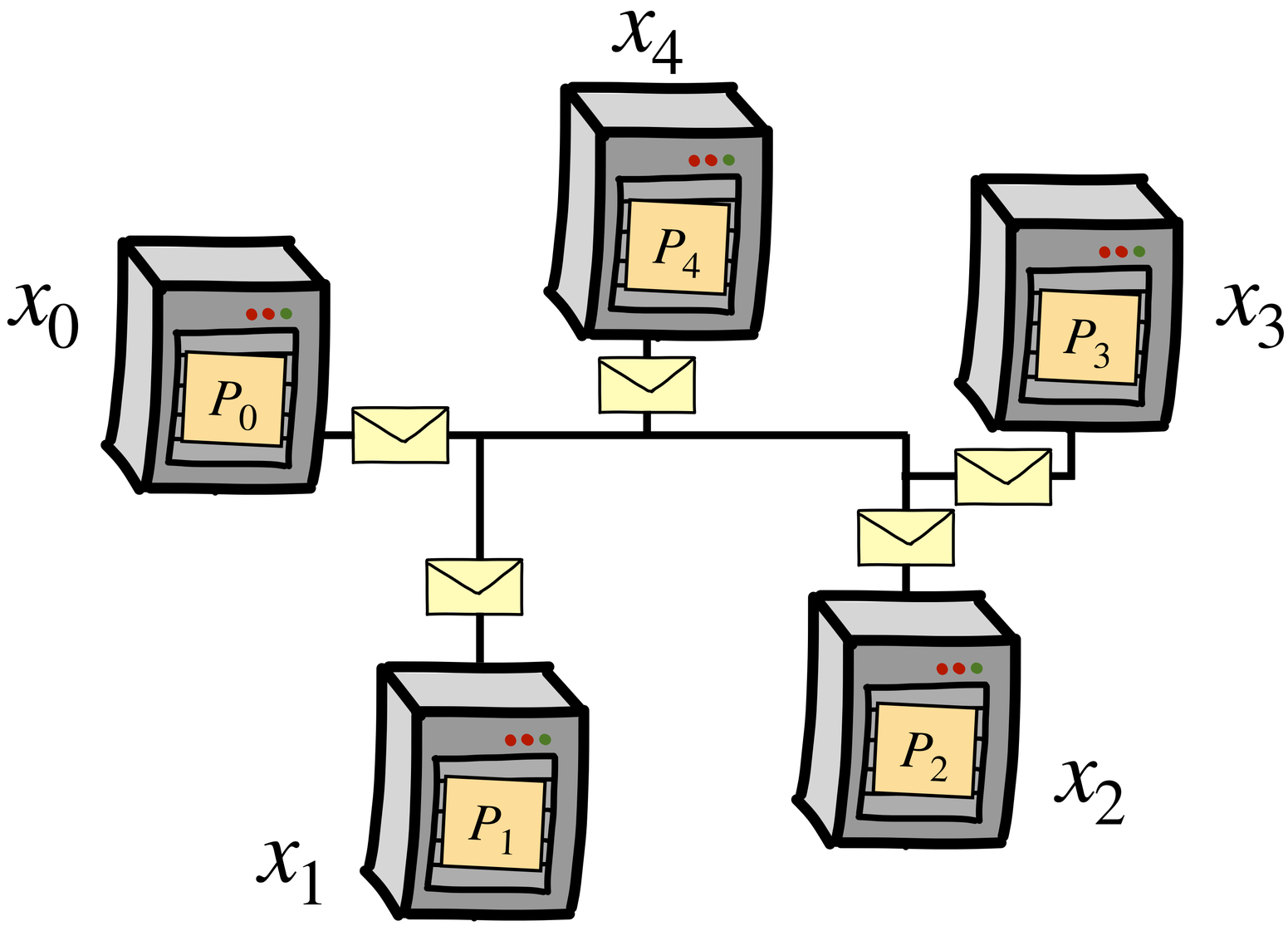
Secure Multiparty Computation (MPC)



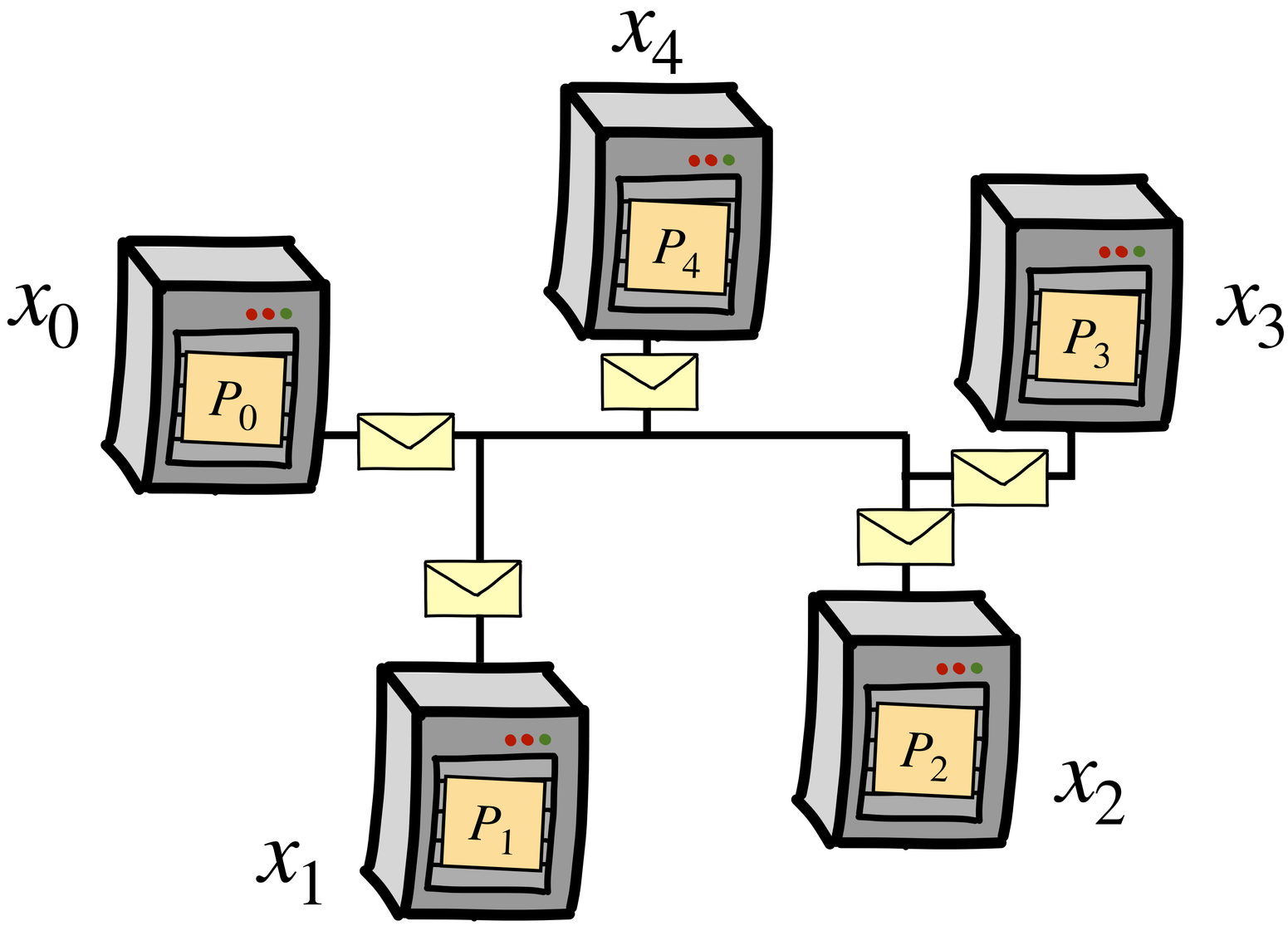
Secure Multiparty Computation (MPC)



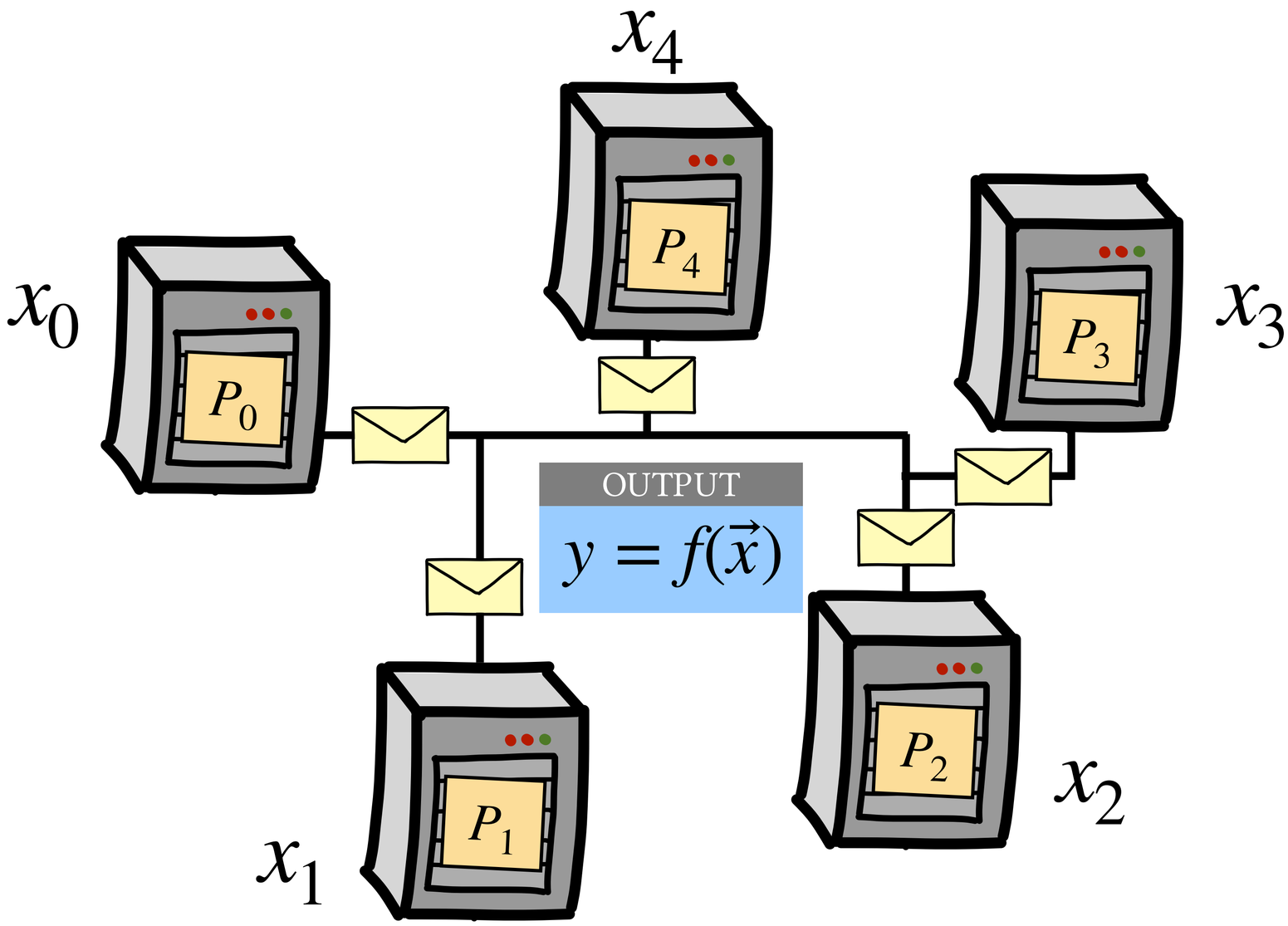
Secure Multiparty Computation (MPC)



Secure Multiparty Computation (MPC)



Secure Multiparty Computation (MPC)



What is Open Finance?

- Open Finance frameworks enable people to securely mobilize personal financial data in order to avail of financial services
  eg. Proving financial health when applying for a loan, apartment rental, etc.

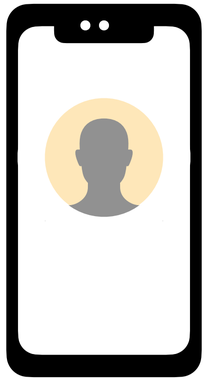
- Cryptographic protocols for provenance replace unverifiable physical documents, and heuristics such as screen scraping

- The Account Aggregator (AA) ecosystem in India is one such Open Finance framework, regulated by the central bank

Account Aggregator (AA)

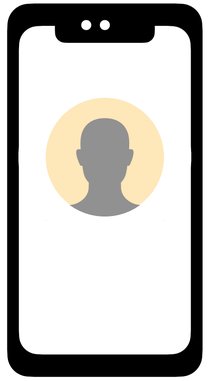
- Users consolidate their financial information across multiple regulated Financial Information Providers (FIPs)
- Upon user consent, the AA shares a curated view of finances to licensed Financial Information Users (FIUs)
  — who then provide financial services to the user
- According to Sahamati (regulated authority for AA), the AA ecosystem as of 2024 has over 80 million users, 155 FIPs, 475 FIUs
- However, issues persist: incentive gaps, and single points of failure

User / Data Principal



Financial Information Provider

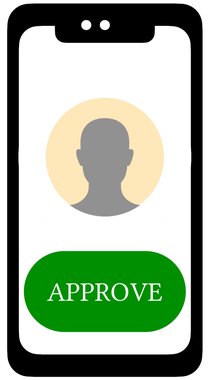
Financial Information User



FIP

Account Aggregator

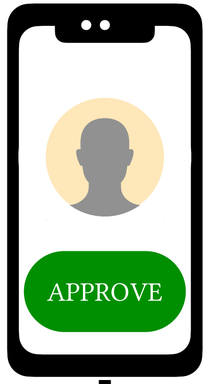
FIU



FIP

Account Aggregator

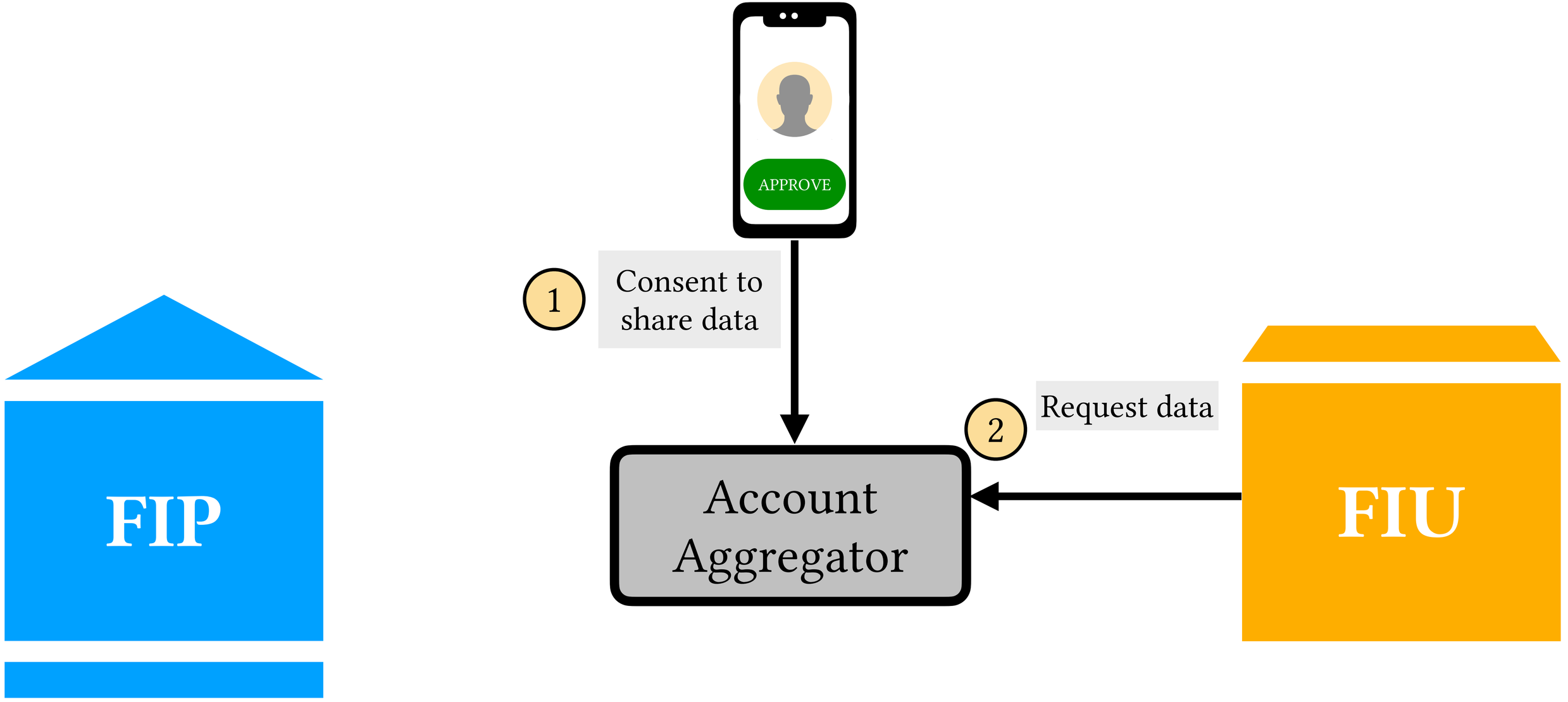
FIU

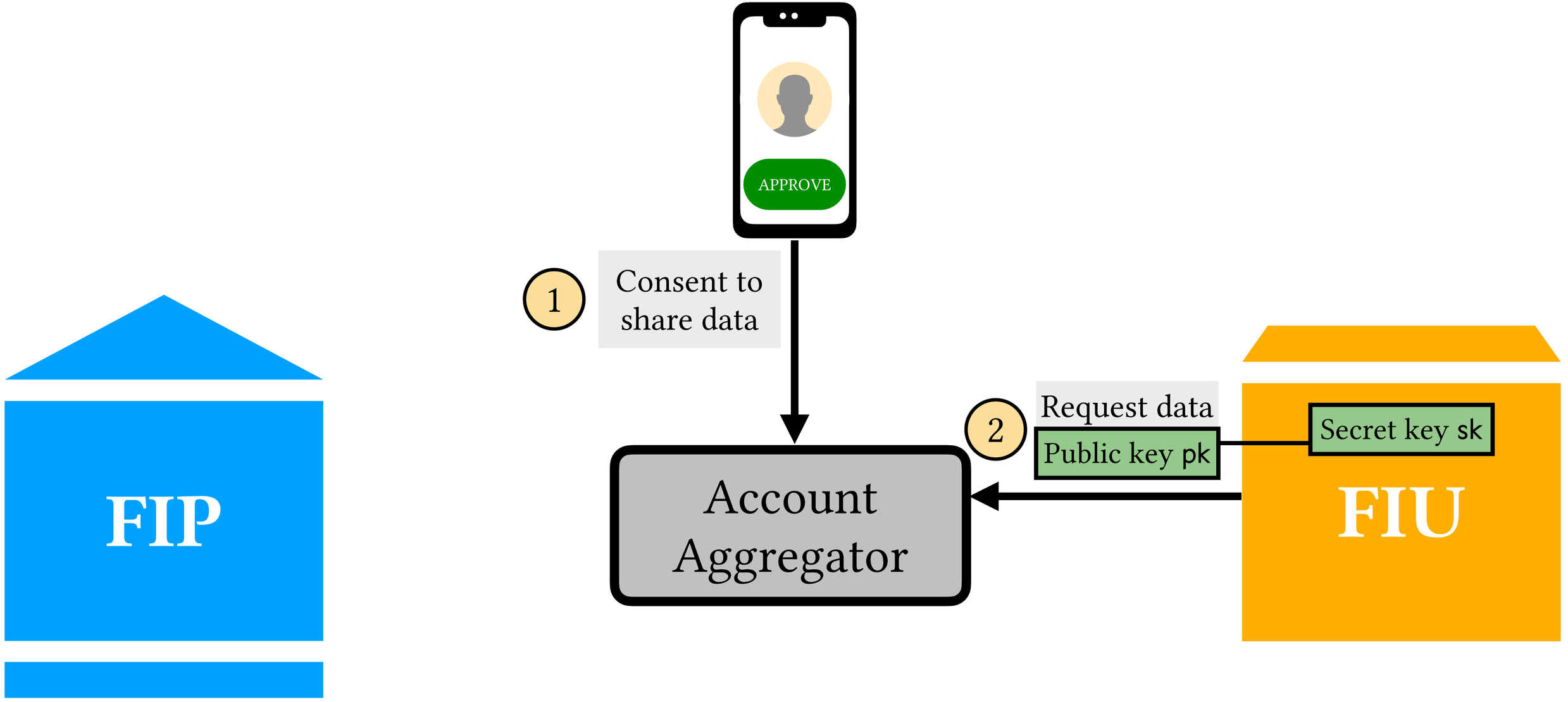


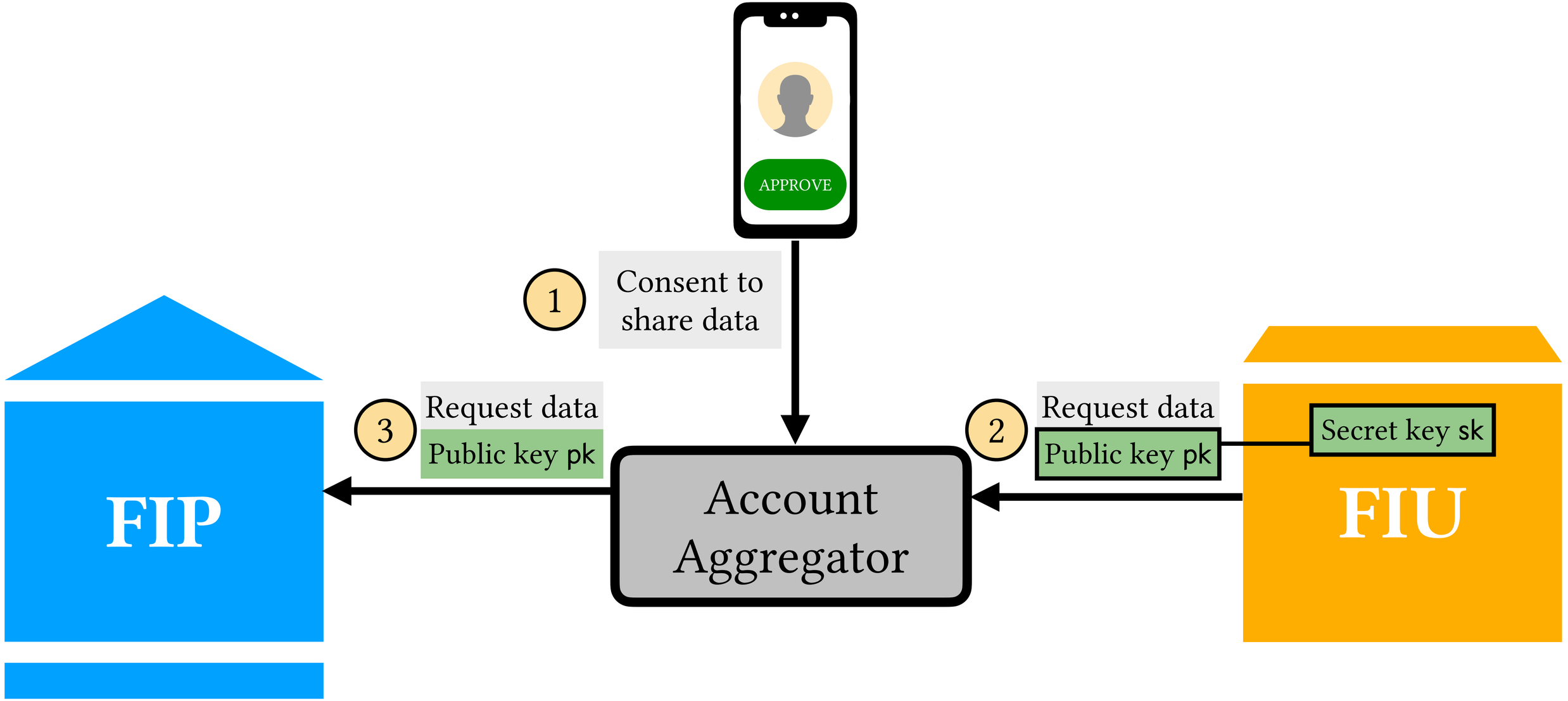
1

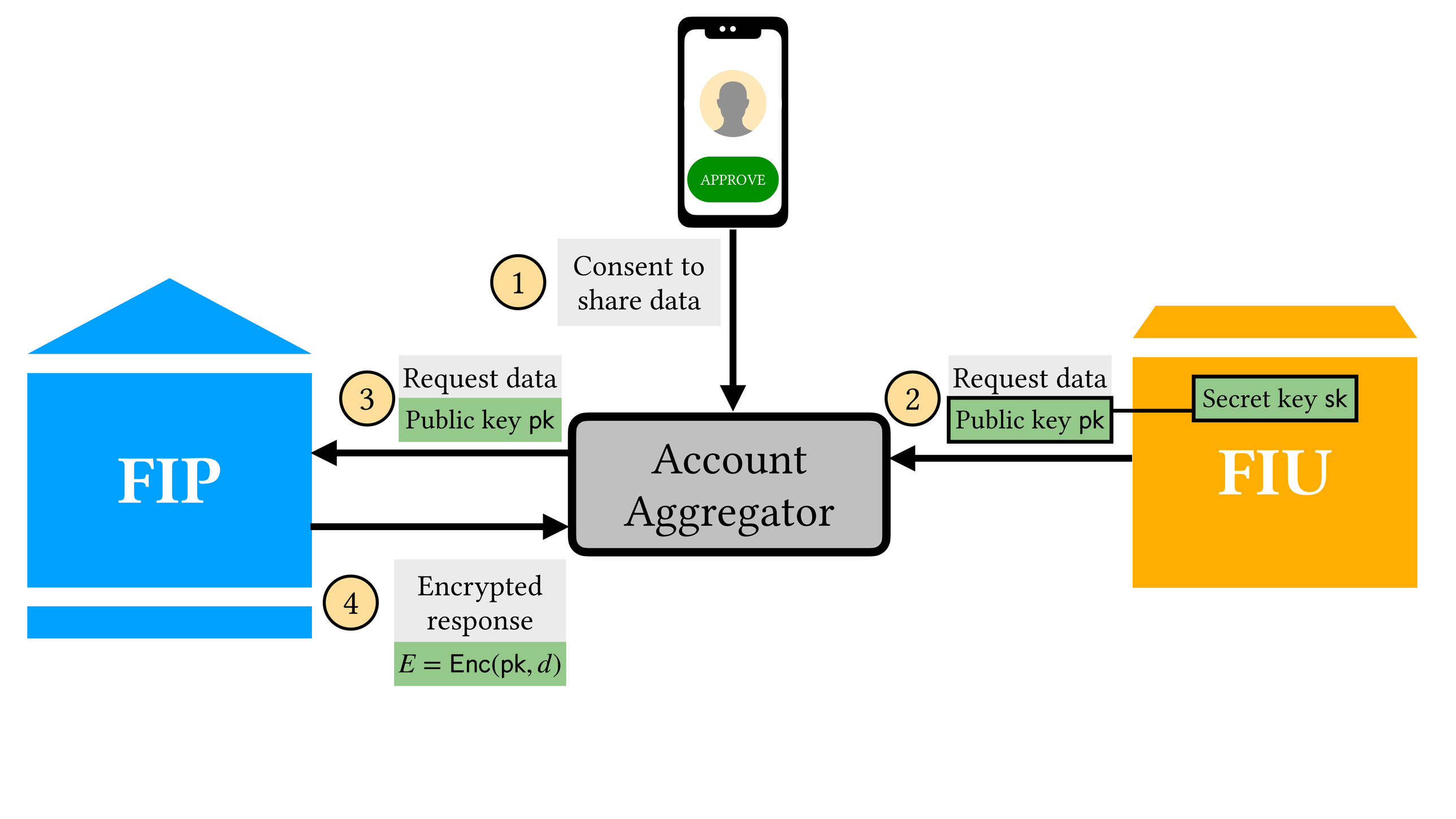
Consent to share data

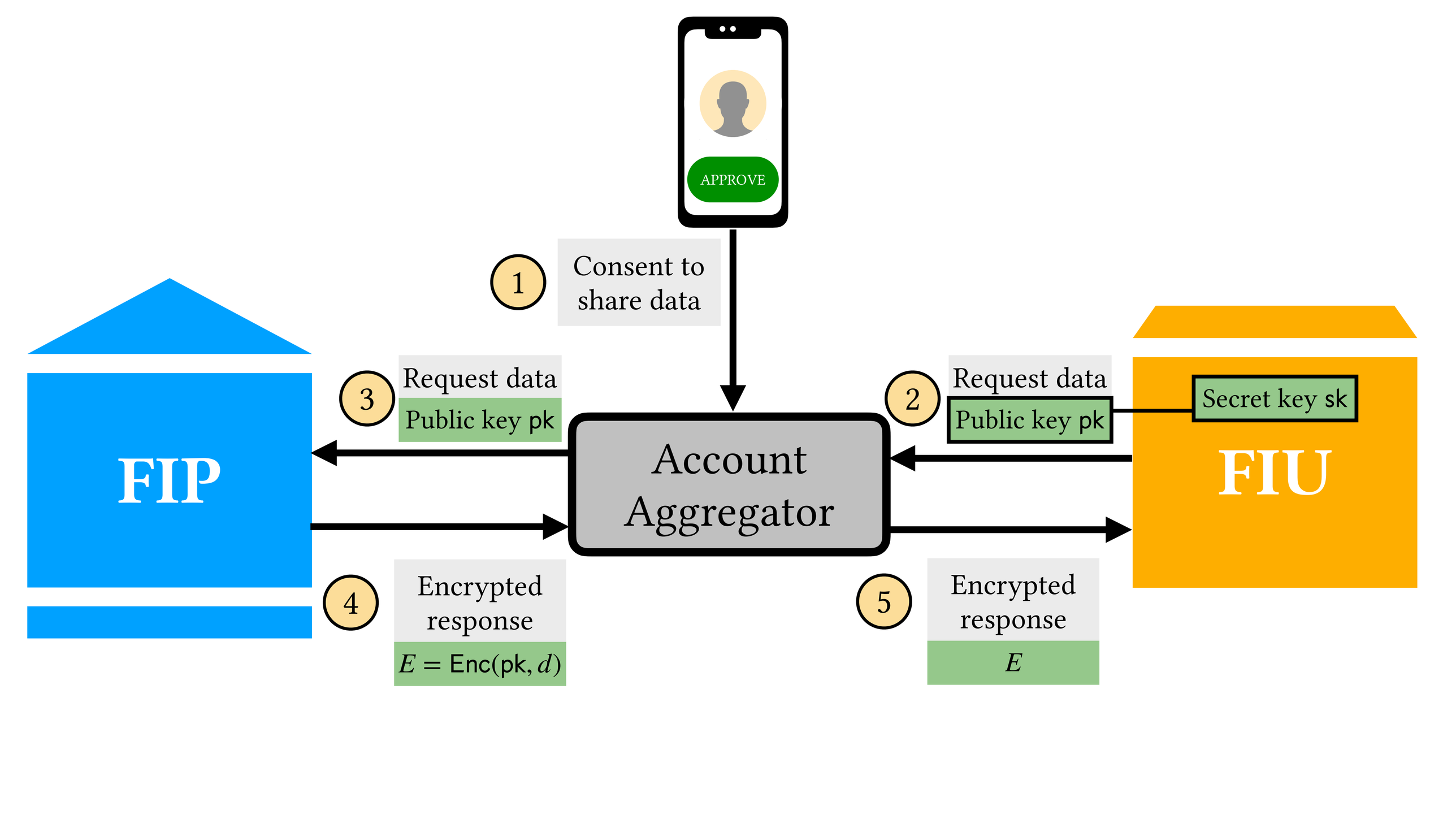
Account Aggregator

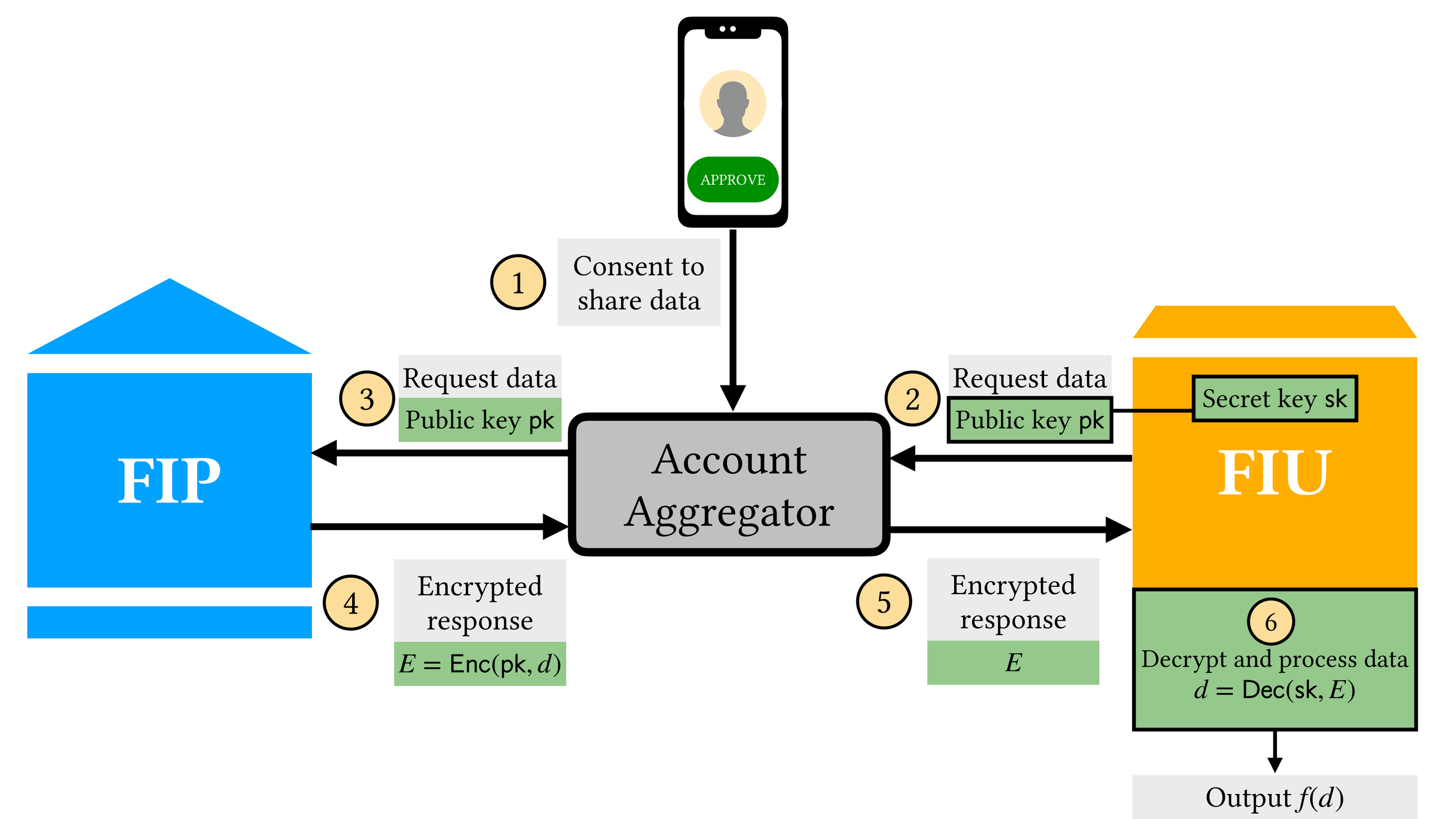

APPROVE
1
Consent to share data
2
Request data
FIP
Account Aggregator
FIU


APPROVE
1
Consent to share data
FIP
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU

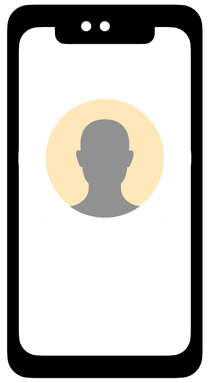

APPROVE
1
Consent to share data
FIP
3
Request data
Public key pk
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU


APPROVE
1
Consent to share data
FIP
3
Request data
Public key pk
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU
4
Encrypted response
$E = \mathsf{Enc}(\mathsf{pk}, d)$


APPROVE
1
Consent to share data
FIP
3
Request data
Public key pk
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU
4
Encrypted response
$E = \mathsf{Enc}(\mathsf{pk}, d)$
5
Encrypted response
E


APPROVE
1
Consent to share data
FIP
3
Request data
Public key pk
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU
4
Encrypted response
$E = \mathsf{Enc}(\mathsf{pk}, d)$
5
Encrypted response
E
6
Decrypt and process data
$d = \mathsf{Dec}(\mathsf{sk}, E)$
Output $f(d)$

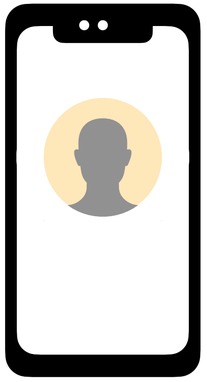
User device



Account Aggregator

Provides	Data d	Consent to mobilize d	Pipeline from consent→data	Services to user
Obtains	—	Services from FIU	Fees from FIU	Depends on business model

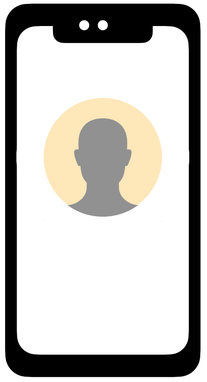
User device



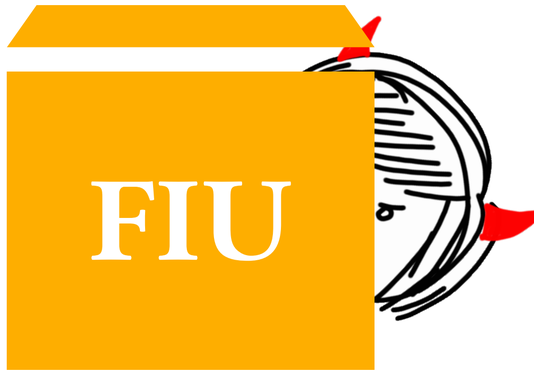
Account Aggregator

Provides	Data d	Consent to mobilize d	Pipeline from consent→data	Services to user
Obtains	—	Services from FIU	Fees from FIU	Depends on business model Data d

User device

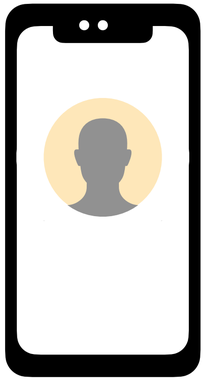


Account Aggregator

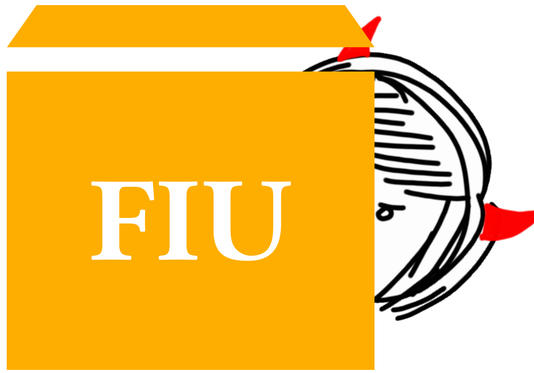


Provides	Data d	Consent to mobilize d	Pipeline from consent→data	Services to user
Obtains	—	Services from FIU	Fees from FIU	Depends on business model Data d

User device



Account Aggregator



	FIP	User device	Account Aggregator	FIU
Provides	Data d	Consent to mobilize d	Pipeline from consent→data	Services to user
Obtains	—	Services from FIU	Fees from FIU	Depends on business model

Data theft

No consent

Lost fees

Data d

Unrestricted reuse

Implications of Exposing d to FIU

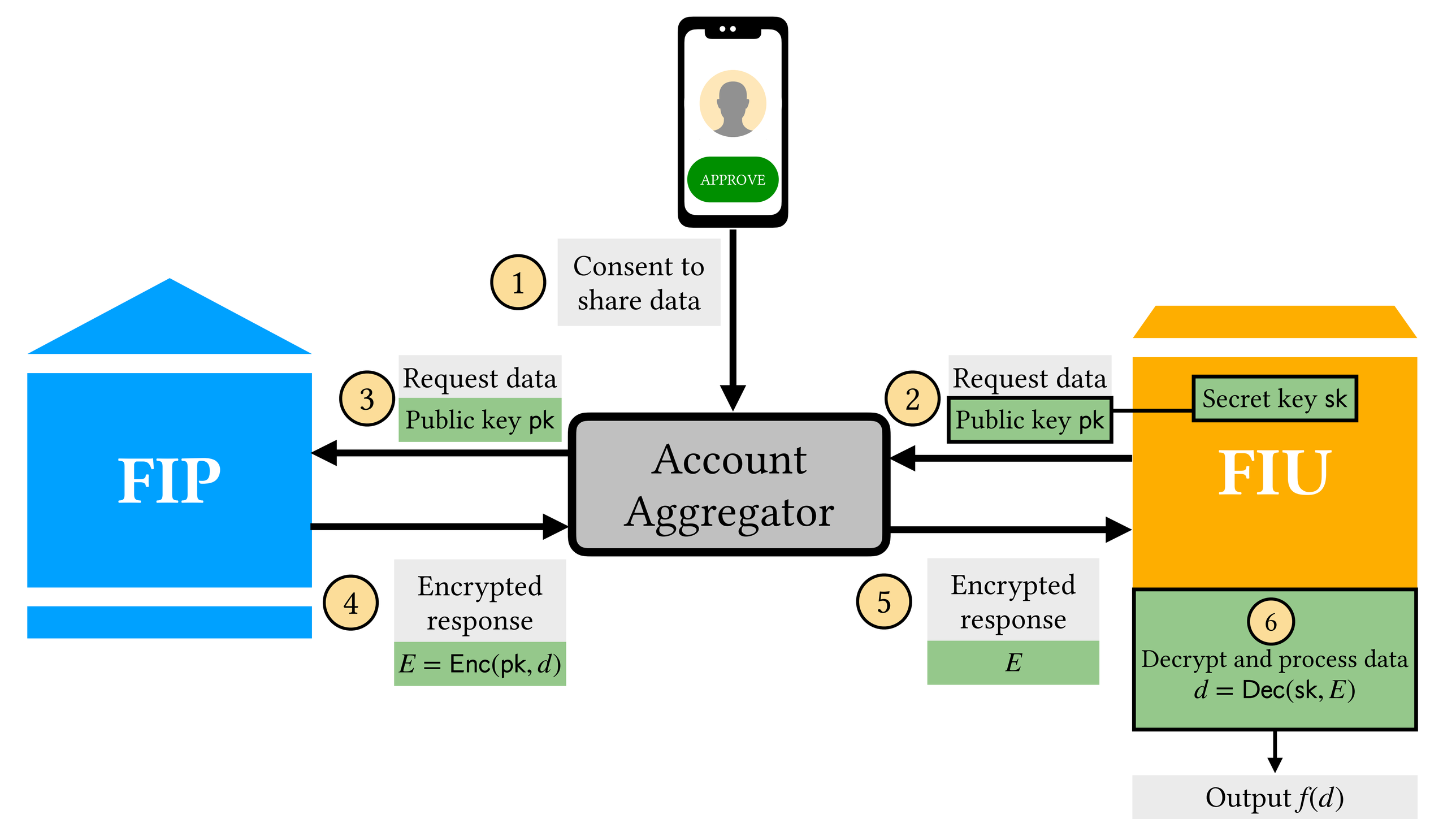
- Data breach or insider attack on FIU puts data in the wrong hands, affects the whole ecosystem

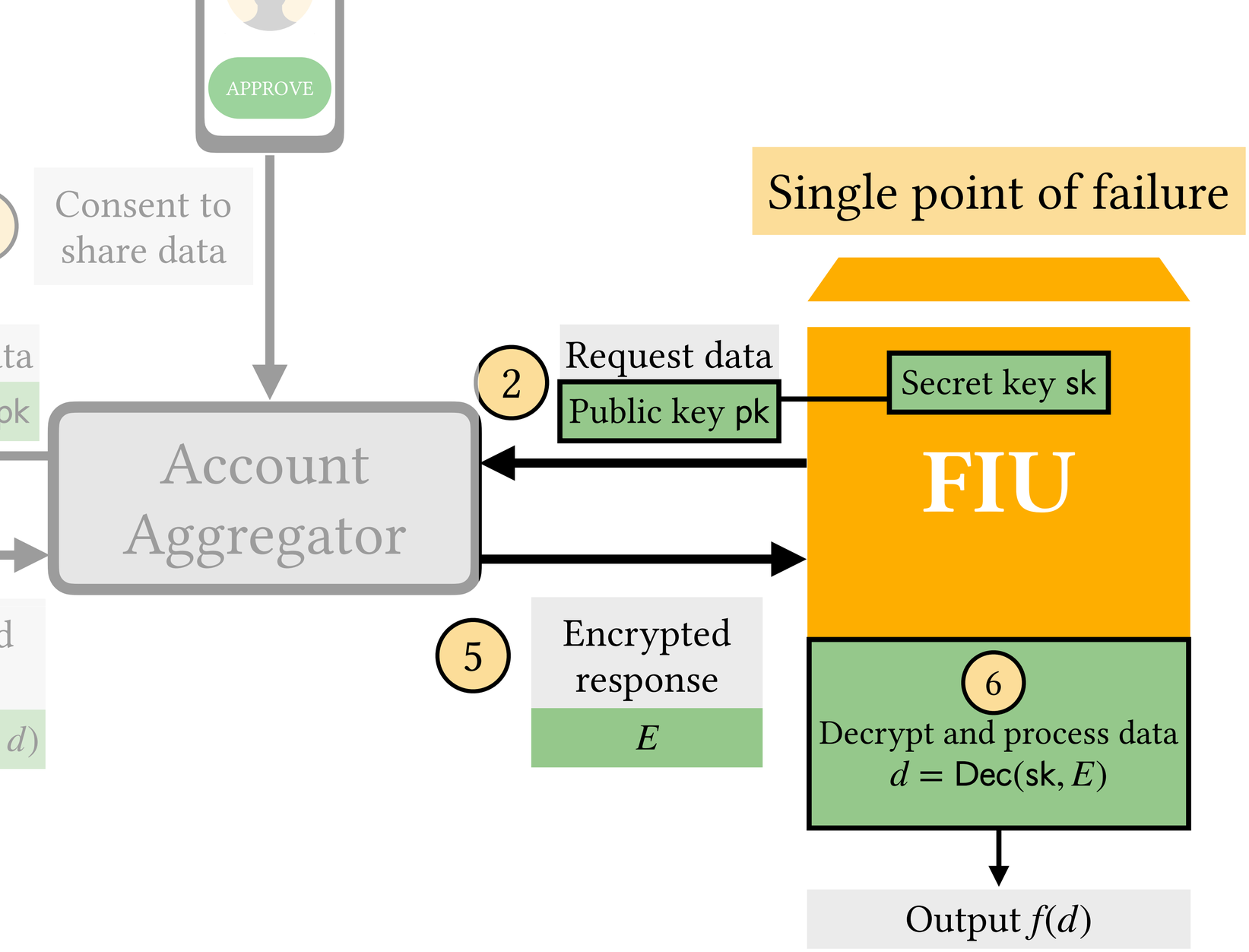
  - Users lose control over their personal financial information

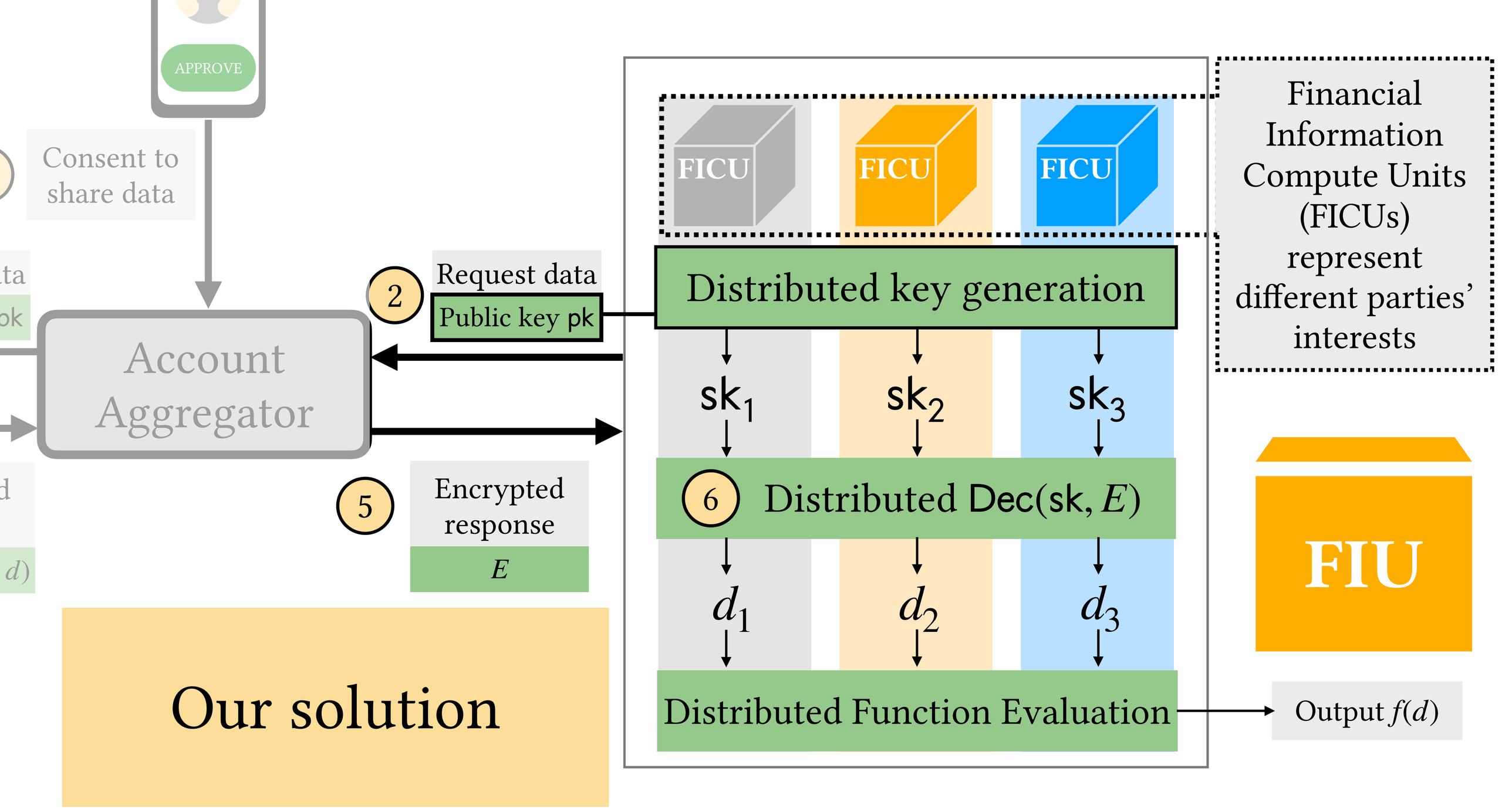
  - FIPs affected on a macro level: loss of proprietary data

- Heuristic FIP solution: throttle response rate
  - Undermines ecosystem, and still leaks data anyway

- A compliance-by-design approach is needed


APPROVE
1
Consent to share data
FIP
3
Request data
Public key pk
Account Aggregator
2
Request data
Public key pk
Secret key sk
FIU
4
Encrypted response
$E = \mathsf{Enc}(\mathsf{pk}, d)$
5
Encrypted response
E
6
Decrypt and process data
$d = \mathsf{Dec}(\mathsf{sk}, E)$
Output $f(d)$


APPROVE
Consent to share data
Account Aggregator
2
Request data
Public key pk
Single point of failure
Secret key sk
FIU
5
Encrypted response
E
6
Decrypt and process data
d = Dec(sk, E)
Output f(d)


APPROVE
Consent to share data
Account Aggregator
2
Request data
Public key pk
5
Encrypted response
E
FICU
FICU
FICU
Financial Information Compute Units (FICUs) represent different parties' interests
Distributed key generation
sk1
sk2
sk3
6
Distributed Dec(sk, E)
d1
d2
d3
Distributed Function Evaluation
Output f(d)
FIU
Our solution

The FICU Paradigm

- Three FICU nodes, with naturally non-colluding operators:
  - FIU, interested in utility of data
  - FIP (or proxy), interested in minimizing data exposure
  - Infrastructure provider, interested in credibility of the AA ecosystem
- **Duplication protection**. Data is secret shared, and accessible only via MPC
- **Fixes participation incentives**.
  Value of data $\propto$ quality and quantity of usage
  Fine-grained compensation model for FIPs

Technical Details

- MPC model: 3 parties, no colluding pairs (1 active corruption)
- Ideal functionality to realize:
    1. Generate pk and $[\mathsf{sk}]$
    2. Obtain $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, d)$ from AA
    3. Compute $[d] = \mathsf{Dec}([\mathsf{sk}], \mathsf{ct})$
    4. Operate on $[d]$ as required

Technical Details

- MPC model: 3 parties, no colluding pairs (1 active corruption)
- Ideal functionality to realize:
  1. Generate pk and $[\mathsf{sk}]$
  2. Obtain $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, d)$ from AA
  3. Compute $[d] = \mathsf{Dec}([\mathsf{sk}], \mathsf{ct})$
  4. Operate on $[d]$ as required

Standard queries

Technical Details

- MPC model: 3 parties, no colluding pairs (1 active corruption)
- Ideal functionality to realize:

1. Generate pk and $[\mathsf{sk}]$
2. Obtain $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, d)$ from AA
3. Compute $[d] = \mathsf{Dec}([\mathsf{sk}], \mathsf{ct})$
4. Operate on $[d]$ as required

Standard queries

Threshold decryption

-Well studied problem

-Simple solution: tweak Enc to directly encrypt secret shares

Technical Details

- MPC model: 3 parties, no colluding pairs (1 active corruption)
- Ideal functionality to realize:

1. Generate pk and [sk]
2. Obtain $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, d)$ from AA
3. Compute $[d] = \mathsf{Dec}([\mathsf{sk}], \mathsf{ct})$
4. Operate on $[d]$ as required

Standard queries

Threshold decryption

-Well studied problem

-Simple solution: tweak Enc to directly encrypt secret shares

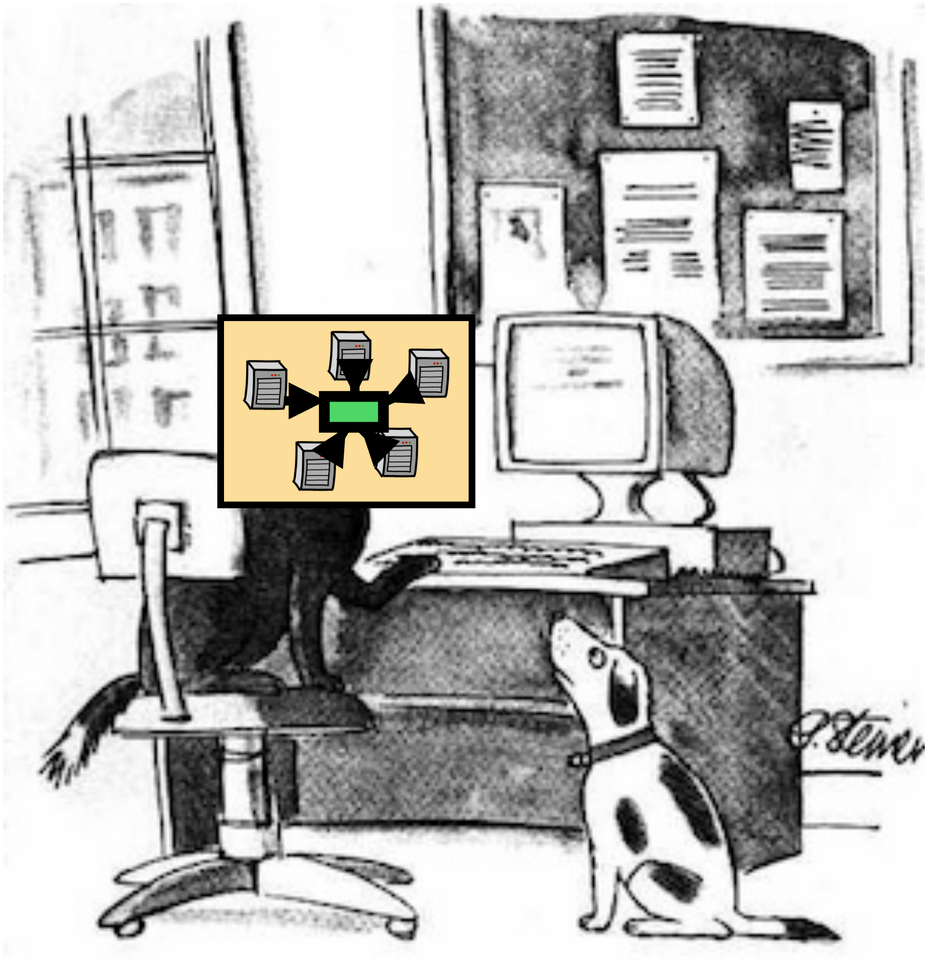
Standards set externally; can't be changed

MPC-agnostic Context

Peter Steiner,
The New Yorker
July 5, 1993 issue

"On the internet, nobody knows you're a dog"

MPC-agnostic Context



Peter Steiner,
The New Yorker
July 5, 1993 issue

"On the internet, nobody knows you're RUNNING MPC

Our Approach

- **Guiding principle**: Protocol specs are set agnostic to MPC, unrealistic to wait for MPC-friendly standards
- We design our solution to be a drop-in replacement for FIU, for integration with AA ecosystem as it exists *today*
- Two protocol design challenges:
  - $\mathsf{Dec}([\mathsf{sk}], \mathsf{ct})$ is a difficult function to handle in MPC
  - FIP (via AA) delivers data d as an XML file
    $\Rightarrow [d]$ must be **parsed** before running any queries

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

2. Use key K to evaluate AES in GCM mode

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Boolean circuit

Elliptic curve group ops

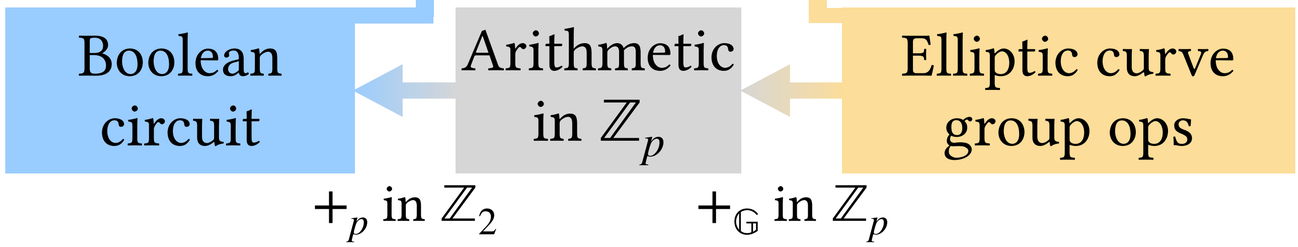
2. Use key K to evaluate AES in GCM mode

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]



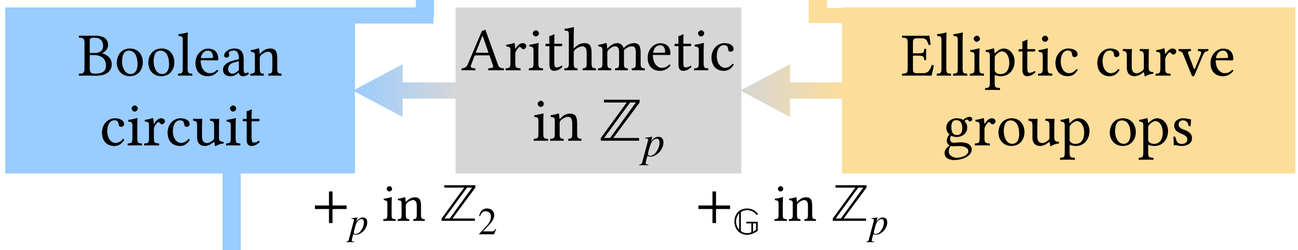
2. Use key K to evaluate AES in GCM mode

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]



2. Use key K to evaluate AES in GCM mode

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]

Boolean circuit

Arithmetic in $\mathbb{Z}_p$

Elliptic curve group ops

$+_p$ in $\mathbb{Z}_2$

$+_{\mathbb{G}}$ in $\mathbb{Z}_p$

2. Use key K to evaluate AES in GCM mode

Which general Boolean MPC paradigm?	
Garbled Circuits $O(1)$ rounds 100s bits/gate	"Secret-sharing" $O(\text{depth})$ rounds few bits/gate

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]

Boolean circuit

Arithmetic in $\mathbb{Z}_p$

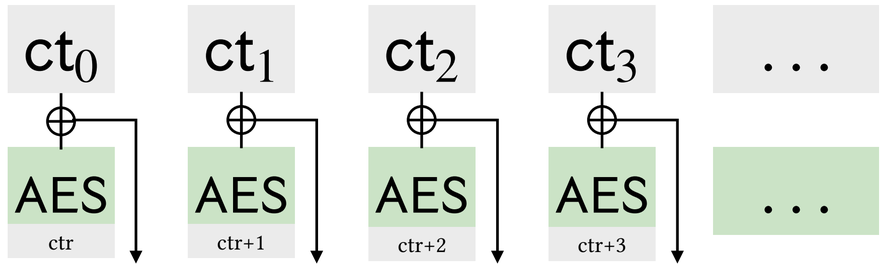
Elliptic curve group ops

$+_p$ in $\mathbb{Z}_2$

$+_{\mathbb{G}}$ in $\mathbb{Z}_p$

2. Use key K to evaluate AES in GCM mode

AES invocations in parallel



Which general Boolean MPC paradigm?	
Garbled Circuits $O(1)$ rounds 100s bits/gate	"Secret-sharing" O(depth) rounds few bits/gate

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]

Boolean circuit

Arithmetic in $\mathbb{Z}_p$

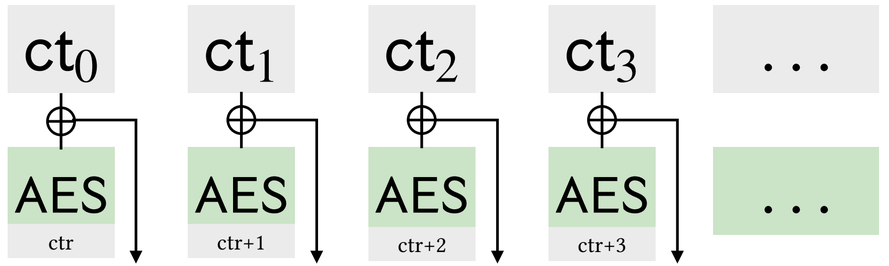
Elliptic curve group ops

$+_p$ in $\mathbb{Z}_2$

$+_{\mathbb{G}}$ in $\mathbb{Z}_p$

2. Use key K to evaluate AES in GCM mode

AES invocations in parallel



Circuit depth independent of |ct|

Which general Boolean MPC paradigm?

Garbled Circuits	"Secret-sharing"
$O(1)$ rounds	$O(\text{depth})$ rounds
100s bits/gate	few bits/gate

Dec([sk], ct) in MPC

Decryption involves two steps:

1. Establish shared key $K = \mathsf{SHA}(g^{\mathsf{sk} \cdot r})$

Along the lines of
[Abram Damgård Scholl Trieflinger 21]
[Mohassel Rindal 18]

Boolean circuit

Arithmetic in $\mathbb{Z}_p$

Elliptic curve group ops

$+_p$ in $\mathbb{Z}_2$

$+_{\mathbb{G}}$ in $\mathbb{Z}_p$

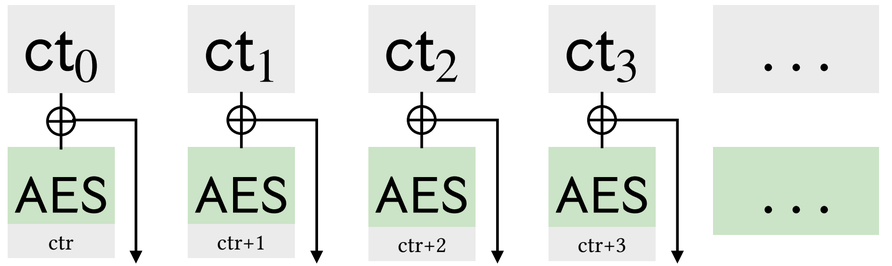
2. Use key K to evaluate AES in GCM mode

AES invocations in parallel

Which general Boolean MPC paradigm?

Garbled Circuits	"Secret-sharing"
$O(1)$ rounds	$O(\text{depth})$ rounds
100s bits/gate	few bits/gate

Wins for $|\mathsf{ct}| >$ few KB



Circuit depth independent of $|\mathsf{ct}|$

Our Approach

- **Guiding principle**: Protocol specs are set agnostic to MPC, unrealistic to wait for MPC-friendly standards
- We design our solution to be a drop-in replacement for FIU, for integration with AA ecosystem as it exists *today*
- Two protocol design challenges:

✓ Dec([sk], ct) is a difficult function to handle in MPC

- FIP (via AA) delivers data d as an XML file
  $\Rightarrow$ $[d]$ must be **parsed** before running any queries

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format

```
<Transactions>
    ⋮
  <Transaction id="S1842" amt="54.2" nar="UPI/ZMTO" ts="13:22" >
    ⋮
</Transactions>
```

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure

```
<Transaction|id="S1842" |amt="54.2" |nar="UPI/ZMTO"|ts="13:22" >
```

```
transaction_struct  tr = {string id, int amt, string nar, time_t ts}
```

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Step 1: identify delimiters

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

<Transaction XXXXXXXX XXXXXXXXXX XXXXXXXXXXXXX XXXXXXXXXX>

Step 1: identify delimiters

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

<Transaction XXXXXXXX XXXXXXXXXX XXXXXXXXXXXXX XXXXXXXXXX>

Leakage!

Step 1: identify delimiters

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

10 chars

<Transaction XXXXXXXX XXXXXXXXXX XXXXXXXXXXXXX XXXXXXXXXX>

Leakage!

Step 1: identify delimiters

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

10 chars

<Transaction XXXXXXXX XXXXXXXXXX XXXXXXXXXXXXX XXXXXXXXXX>

Leakage!

Step 1: identify delimiters

amt="54.2"
10 chars

vs.

amt="104.3"
11 chars

Why is Parsing Challenging?

- Plaintext is a bank statement that consists of a list of transactions, provided in XML format
- Plaintext parsing: identify delimiters, cast into structure
- What about parsing in secret shared form?

<Transaction XXXXXXXX XXXXXXXXXX XXXXXXXXXXXXXX XXXXXXXXXX>

10 chars

Leakage!

Step 1: identify delimiters

✓

amt="54.2"

10 chars

vs.

✗

amt="104.3"

11 chars

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

<Transaction|id=“S1842” |amt=“54.2” |nar=“UPI/ZMTO”|ts=“13:22” >

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

```
id=S1842  amt=54.2  nar=UPI/ZMTO ts=13:22
```

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

```
id = S1842
amt= 54.2
nar= UPI/ZMTO
ts= 13:22
```

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

```
id = S1842
amt= 54.2
nar= UPI/ZMTO
ts= 13:22
```

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

```
 id  S 1 8 4 2
amt  5 4 . 2
nar  U P I / Z M T O
 ts  1 3 : 2 2
```

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

id	S	1	8	4	2				
amt	5	4	.	2					
nar	U	P	I	/	Z	M	T	O	
ts	1	3	:	2	2				

Max row length = 10

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

id	S	1	8	4	2	#	#	#	#	#
amt	5	4	.	2	#	#	#	#	#	#
nar	U	P	I	/	Z	M	T	O	#	#
ts	1	3	:	2	2	#	#	#	#	#

Max row length = 10

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

id	S	1	8	4	2	#	#	#	#	#
amt	5	4	.	2	#	#	#	#	#	#
nar	U	P	I	/	Z	M	T	O	#	#
ts	1	3	:	2	2	#	#	#	#	#

Max row length = 10

Max string length = 40

Actual string length = 22

Padding chars = 18

Secure Parsing

- Ideal situation: each entry is “padded” to a fixed max length
- Our approach is to securely derive this padding

```
id  S 1 8 4 2 # # # # #
amt 5 4 . 2 # # # # # #
nar U P I / Z M T O # #
ts  1 3 : 2 2 # # # # #
```

Max string length = 40

Actual string length = 22

Padding chars = 18

Max row length = 10

Secure Parsing

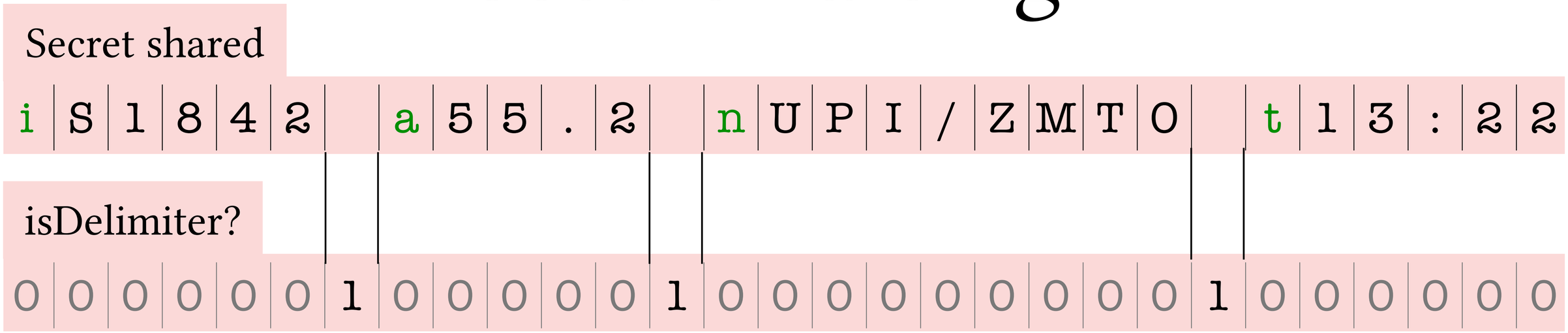
i S 1 8 4 2 a 5 4 . 2 n U P I / Z M T O t 1 3 : 2 2

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Secure Parsing



Secure Parsing

Secret shared

i	S	l	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	l	3	:	2	2

isDelimiter?

0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

relative_index (distance from last delimiter-1)

0	1	2	3	4	5	-1	0	1	2	3	4	-1	0	1	2	3	4	5	6	7	8	-1	0	1	2	3	4	5

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2
0	1	2	3	4	5	-1	0	1	2	3	4	-1	0	1	2	3	4	5	6	7	8	-1	0	1	2	3	4	5

relative_index (distance from last delimiter-1)

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2
0	1	2	3	4	5	-1	0	1	2	3	4	-1	0	1	2	3	4	5	6	7	8	-1	0	1	2	3	4	5

relative_index (distance from last delimiter-1)

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Public Pad array

id	#	#	#	#	#	#	#	#	#	#
amt	#	#	#	#	#	#	#	#	#	#
nar	#	#	#	#	#	#	#	#	#	#
ts	#	#	#	#	#	#	#	#	#	#

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2
0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

Public Pad array

Actual requirement (secret)

id						#	#	#	#	#
amt					#	#	#	#	#	#
nar									#	#
ts						#	#	#	#	#

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2
0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

Public Pad array

Actual requirement (secret)

isPadRequired?

id						1	1	1	1	1
amt					1	1	1	1	1	1
nar									1	1
ts						1	1	1	1	1

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

Public Pad array

Actual requirement (secret)

absolute_index of required pads

id						6	7	8	9	10
amt					15	16	17	18	19	20
nar									29	30
ts						36	37	38	39	40

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

absolute_index of required pads

6	7	8	9	10	15	16	17	18	19	20	29	30	36	37	38	39	40
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Public Pad array

Two arrays with secret shared *absolute* indices of values within target array

Secure merge yields the desired final result

Secure Parsing

Secret shared

i	S	1	8	4	2		a	5	5	.	2		n	U	P	I	/	Z	M	T	O		t	1	3	:	2	2

Post padding, absolute_index = (max_row_len × last_delim_index) + relative_index

0	1	2	3	4	5	-	10	11	12	13	14	-	20	21	22	23	24	25	26	27	28	-	30	31	32	33	34	35

absolute_index of required pads

6	7	8	9	10	15	16	17	18	19	20	29	30	36	37	38	39	40
#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

Public Pad array

Two arrays with secret shared *absolute* indices of values within target array

Secure merge yields the desired final result

Secure Parsing

	1	2	3	4	5	6	7	8	9	10
id	S	1	8	4	2	#	#	#	#	#

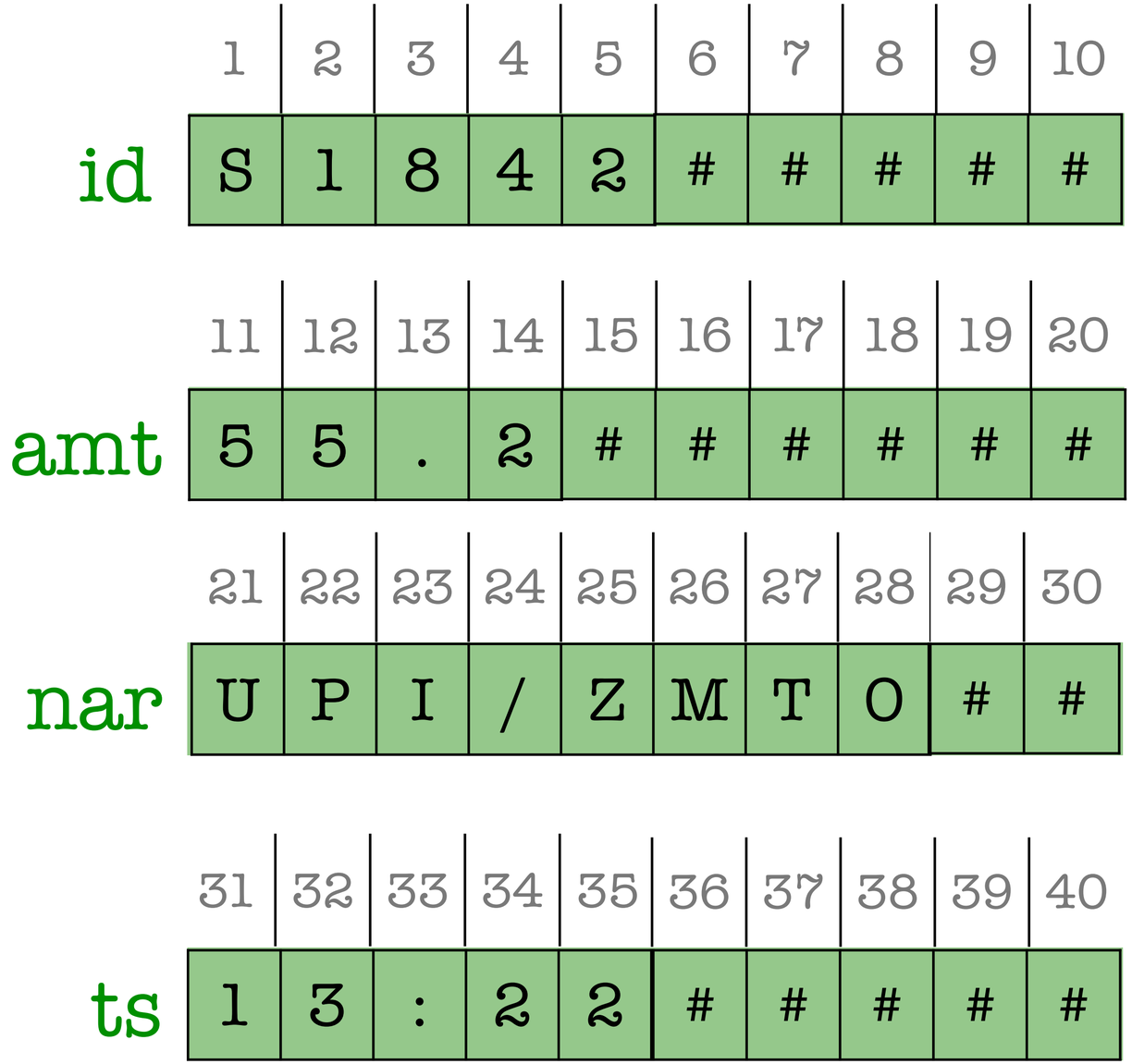
	11	12	13	14	15	16	17	18	19	20
amt	5	5	.	2	#	#	#	#	#	#

	21	22	23	24	25	26	27	28	29	30
nar	U	P	I	/	Z	M	T	O	#	#

	31	32	33	34	35	36	37	38	39	40
ts	1	3	:	2	2	#	#	#	#	#

Secure merge yields the desired final result

Secure Parsing



Secure merge yields the desired final result

Efficiency / Practicalities

- Rough cost profile:
  - Per character: constant number of additions, comparisons, share conversions
  - Constant number of shuffles (simple custom protocol)
  - Overall round complexity independent of file size
- Currently the most expensive component in our integration with the AA ecosystem

Efficiency / Practicalities

- End-to-end: Order of minutes to process a small encrypted bank statement, obtained via actual API
- Plenty of scope to improve the protocol and implementation
- Pilot deployment with partners:

SahamatiNet

Infrastructure provided by Sahamati

fold

Financial Information User (FIU)

Finfactor

Technology Service Provider

Conclusion

- Open Finance frameworks are increasingly popular worldwide
  - eg. India's regulator-driven Account Aggregator (AA) ecosystem

- We propose an MPC-based solution to close gaps in incentives and trust
  - Duplication protection from rogue data fiduciaries (AA: FIUs)
  - Fair compensation model for data custodians (AA: FIPs)

- Design principle: drop-in replacement for immediate use

- Coming soon: full specs on eprint, report on pilot deployment

Thanks!

silencelaboratories.com/open-banking