MPC + TEEs

Sujaya Maiyya

Slides partially acquired from Shantanu Sharma and Sajin Sasy

Secure Multi-Party Computation (MPC)

- MPC enables multiple parties – each holding their own private data – to **evaluate a computation without revealing any of the private data** held by each party

- Each party can only learn any info based on what they can learn from the output and their own input

- Two families: Garbles circuits (2 party) and secret sharing (multi party)

Garbled circuits

Oblivious transfer

- Two parties: Alice and Bob

- Alice has two messages *m1 & m2* and Bob wants to fetch either *m1 or m2*
  - Alice cannot know if Bob picked *m1 or m2*
  - If Bob picked *m1,* he does not know anything about *m2*

- Want to learn more? [Wiki link]

Garbled circuit

1. An underlying function is translated to a Boolean circuit with 2 inputs (can be done by a third party)
2. Alice ***garbles*** (i.e., encrypts) the circuit
3. Alice sends the *garbled circuit* along with her encrypted input to Bob
4. Bob needs to garble his own input and only the garbler (Alice) knows how to garble/encrypt it
   1. Alice and Bob use oblivious transfer
5. Bob evaluates the circuit and obtains encrypted output and shares with Alice

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Alice replaces 0 and 1 with randomly generated *labels* for 0 and 1 in each circuit

→

a	b	c
X_0^a	X_0^b	X_0^c
X_0^a	X_1^b	X_0^c
X_1^a	X_0^b	X_0^c
X_1^a	X_1^b	X_1^c

Alice encrypts the output column using the 2 input labels

Garbled Table
$Enc_{X_0^a, X_0^b}(X_0^c)$
$Enc_{X_0^a, X_1^b}(X_0^c)$
$Enc_{X_1^a, X_0^b}(X_0^c)$
$Enc_{X_1^a, X_1^b}(X_1^c)$

Output can be decrypted only using two correct input labels

Alice permutes the 4 entries and sends it to Bob along with her labeled input

If Alice's input is $\mathbf{a} = a_4 a_3 a_2 a_1 a_0 = 01101$, she sends $X_0^{a_4}$, $X_1^{a_3}$, $X_1^{a_2}$, $X_0^{a_1}$, and $X_1^{a_0}$

Bob needs the labels for his input that he obtains using Oblivious Transfer

If Bob's input is $\mathbf{b} = b_4 b_3 b_2 b_1 b_0 = 10100$, Bob first asks for b_0=0 between Alice's labels $X_0^{b_0}$ and $X_1^{b_0}$

After the data transfer, Bob evaluates the circuit one gate at a time and tries to decrypt the rows in the garbled circuit, where he can decrypt only one row

$$X^c = Dec_{X^a, X^b}(garbled\_table[i]), \text{ where } 0 \leq i \leq 3.$$

Complexity

1. Take any function and transform it into a Boolean circuit
2. Have the garbler garble the entire circuit – every possible input and output combination per gate in the circuit
3. Communicate between the two parties using OT to transfer labels per bit of plaintext, per gate
4. Evaluate and decrypt the output

Secret sharing

Why Secret-Sharing?

- Encryption techniques are **computationally secure**
  - A powerful adversary can break the encryption technique
    - Google, with sufficient computational capabilities, broke SHA-1 (https://shattered.io/)

- **Information-theoretical security**
  - Secure regardless of the computational power of an adversary
  - Quantum secure

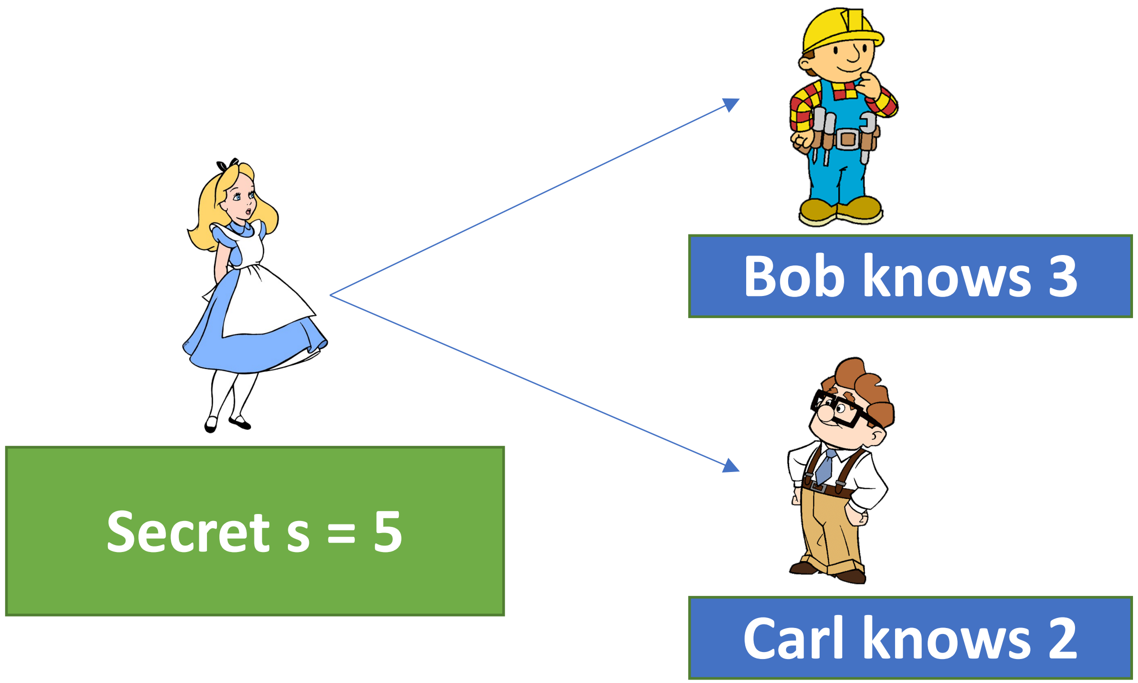
The following slides are from Shantanu Sharma

Additive Secret-Sharing

Assumption: of S servers, at most S-1 servers collude with each other

Split a secret into S shares, store S_i on server *i*

Reconstruct by fetching shares from all and adding them



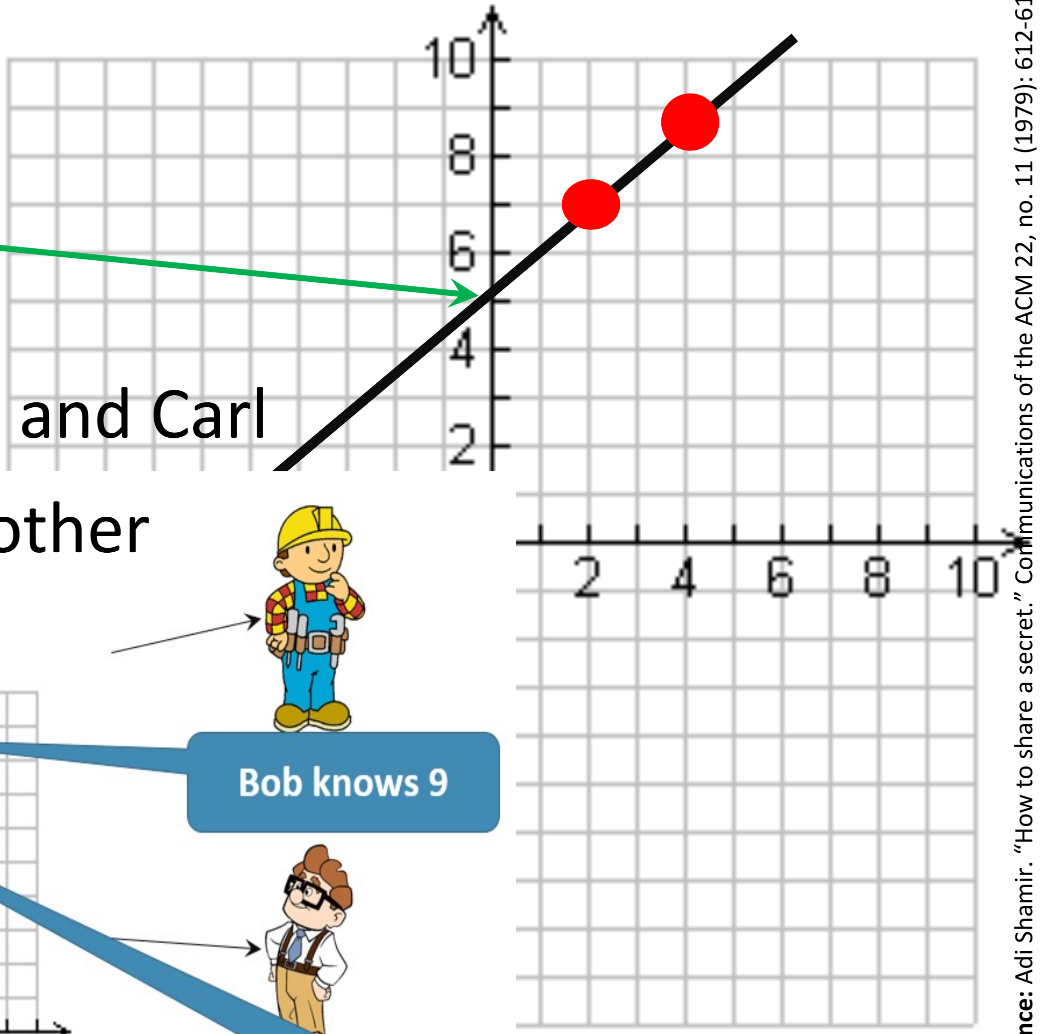
Easy to add (or subtract) secret shared data

$$a + b = \Sigma a_i + \Sigma b_i = \Sigma(a_i + b_i)$$

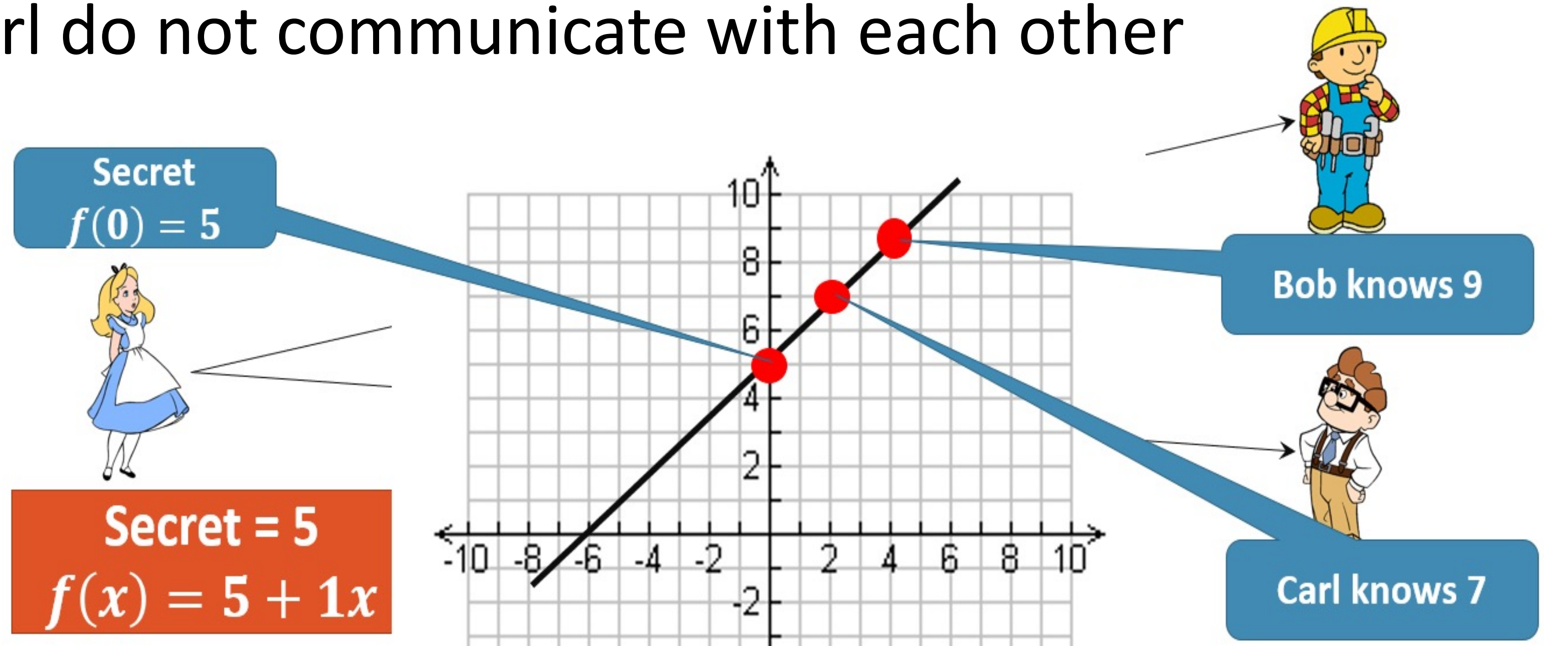
Cons: Even if one party is down, secret cannot be reconstructed

Shamir's Secret-Sharing (SSS) [Shamir79] – Key Idea

- Need at least 2 non-colluding servers
- **One point ⇒ Infinite number of lines**
- **Two points ⇒ Only one line**
  - **Where $f(0)$ is the secret**



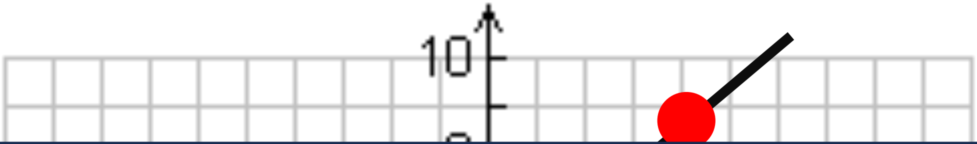
- Alice wants to share her secret value 5 to Bob and Carl
- Bob and Carl do not communicate with each other



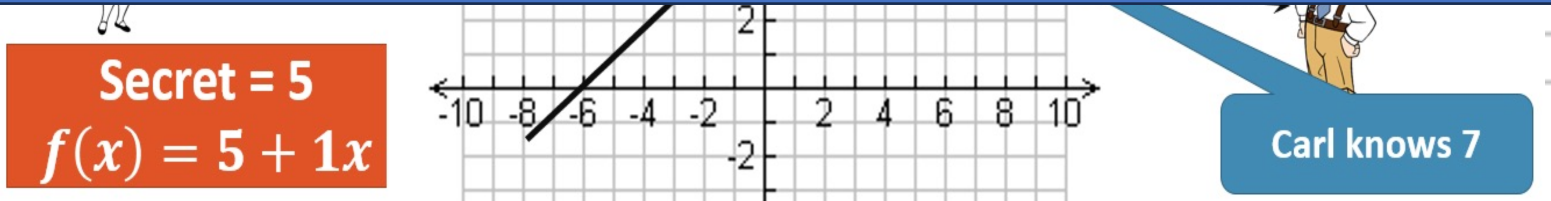
Reference: Adi Shamir. "How to share a secret." Communications of the ACM 22, no. 11 (1979): 612-613.

Shamir's Secret-Sharing (SSS) [Shamir79] – Key Idea

- **One point $\Rightarrow$ Infinite number of lines**
- **Two points $\Rightarrow$ Only one line**



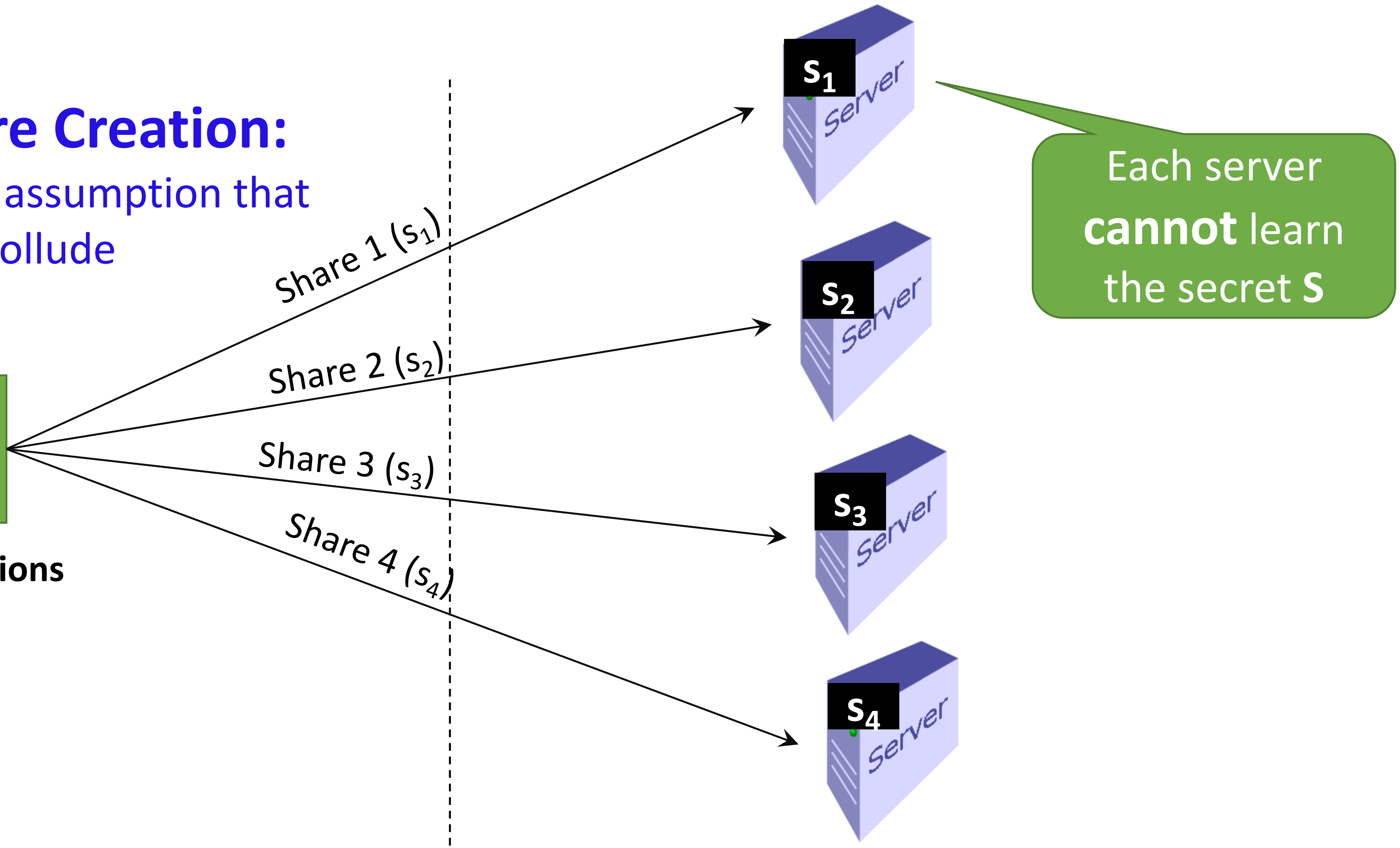
- **Impact of degree of the polynomial vs security**
  - f servers collude $\Rightarrow$ polynomial degree should be $f+1$
    - Servers do not collude $\Rightarrow$ a polynomial of the degree 1
- **Fault tolerant**
  - Due to creating multiple shares



Shamir's Secret-Sharing (SSS)

Secret-Share Creation:
e.g., under the assumption that no server will collude

Secret
S



Mathematical operations
$f(x) = S + ax$

Secret Owner

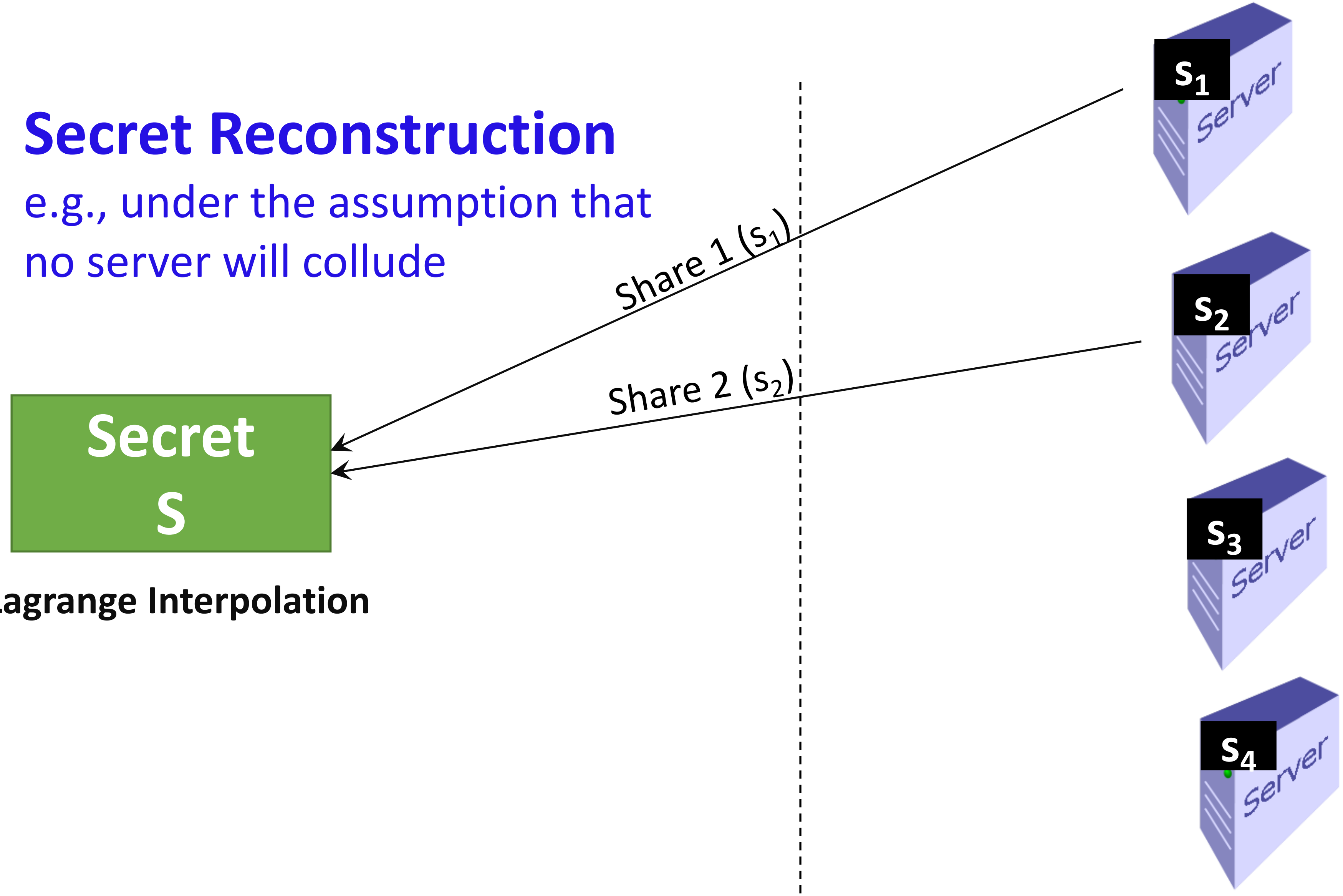
Non-Communicating Public Servers

Reference: Adi Shamir. "How to share a secret." Communications of the ACM 22, no. 11 (1979): 612-613.

Shamir's Secret-Sharing (SSS)

Secret Reconstruction

e.g., under the assumption that
no server will collude



Secret Owner

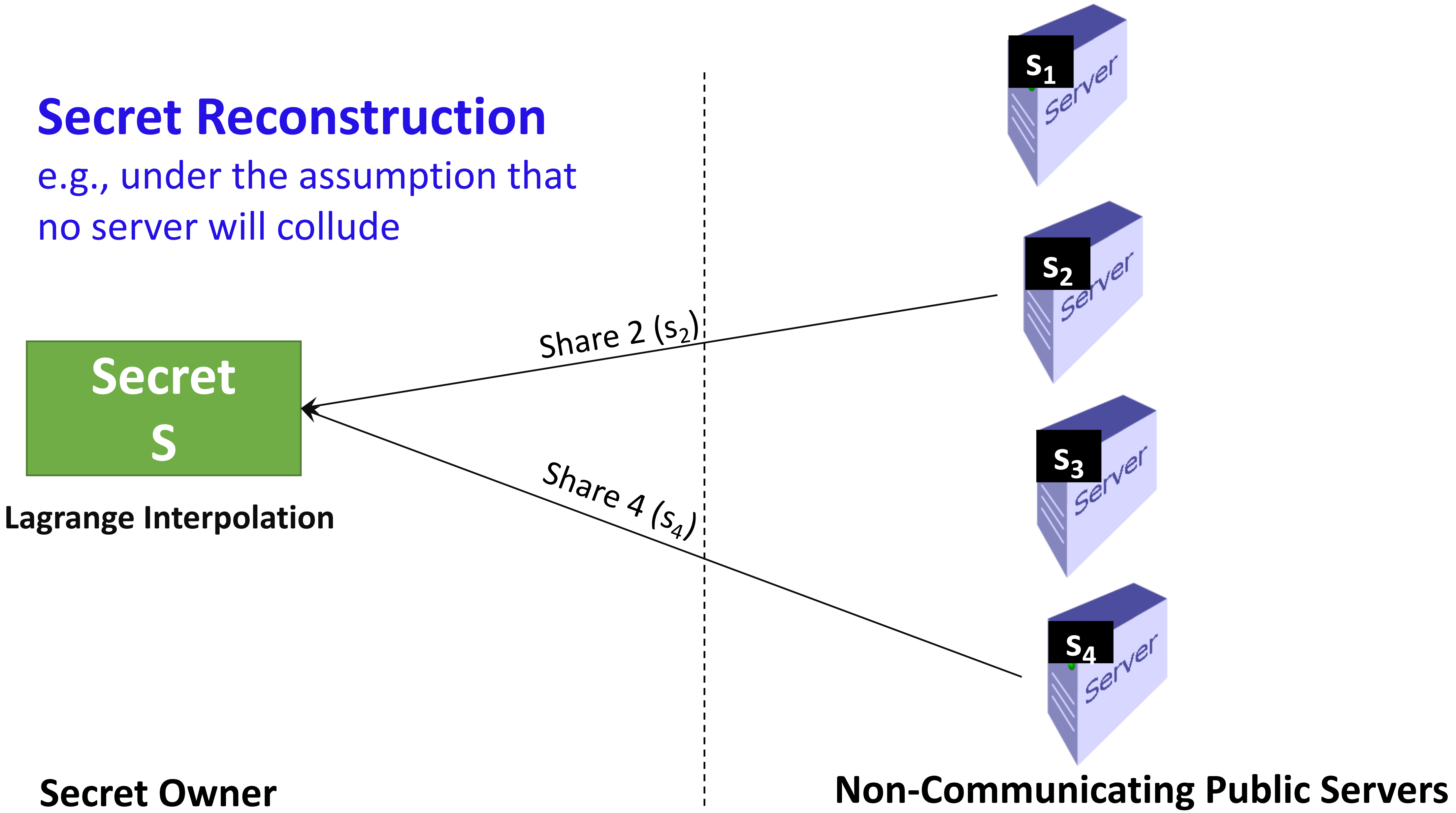
Non-Communicating Public Servers

Reference: Adi Shamir. "How to share a secret." Communications of the ACM 22, no. 11 (1979): 612-613.

Shamir's Secret-Sharing (SSS)

Secret Reconstruction

e.g., under the assumption that
no server will collude



Reference: Adi Shamir. "How to share a secret." Communications of the ACM 22, no. 11 (1979): 612-613.

MPC conclusion marks

- SSS can be used to also support multiplication (how?)
  - SSS supports both addition and multiplication
- Conceptually, GC and SSS can execute most programs
  - However, both have large communication overheads
  - Many solutions to minimize 'online' rounds
- Both techniques are used in developing secure dbs
  - Primarily differs from ORAM dbs in supporting *computations* over columns
  - MPC-based dbs don't always hide access patterns

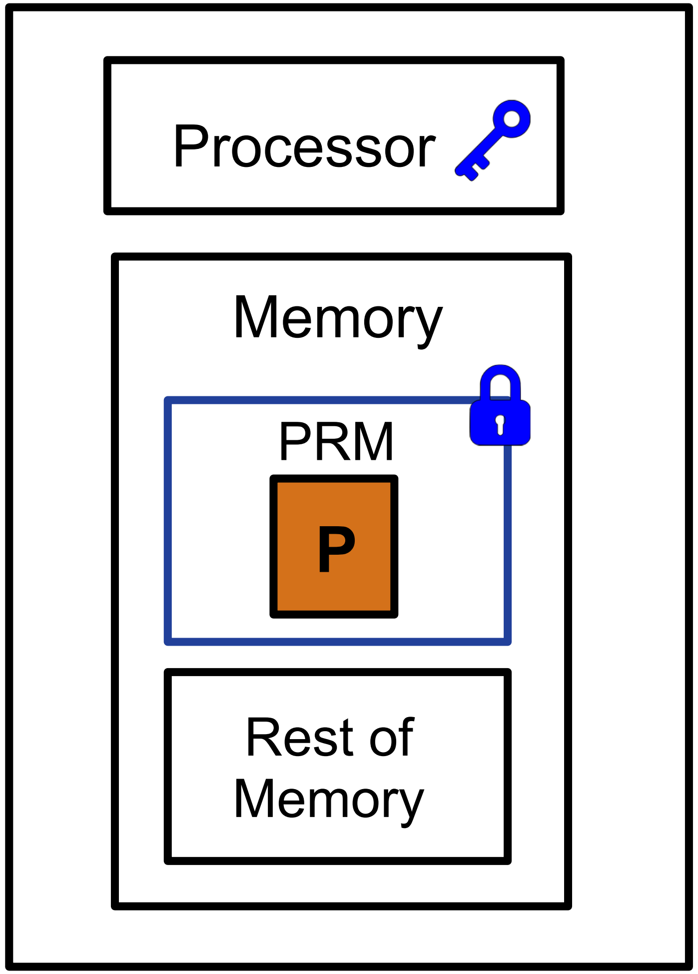
Trusted Execution Environments (TEEs)

Trusted Execution Environments - Intel SGX

- A secure enclave is an **isolated unit of data and code execution** that cannot be accessed even by privileged code (e.g., the operating system or hypervisor)
- *Memory encryption:* only enclave process can access a program's memory
- *Remote attestation:* proof that the code running in the enclave is the one intended, and that it is running on a genuine Intel SGX platform
- *Sealing:* encrypt and authentical the enclave's data to allow stopping and restarting an enclave process w/o losing state
- Developers must partition code as sensitive and non-sensitive. Sensitive code run in the enclave, non-sensitive in host space
- Learn more here

Trusted Execution Environments (TEEs)

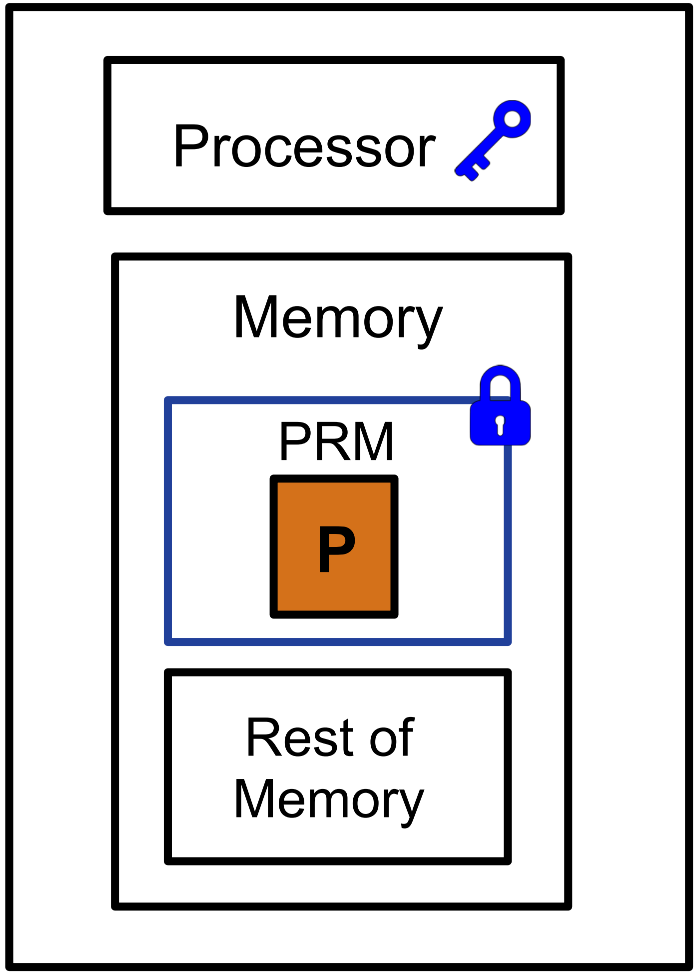
- Processor fused with secret keys at manufacture time
- Enables the processor to set aside Processor Reserved Memory (**PRM**) at boot time
- Able to instantiate secure virtual containers called **enclaves**
- Enclaves can load programs with confidentiality, integrity and freshness guarantees



The following slides are from Sajin Sasy

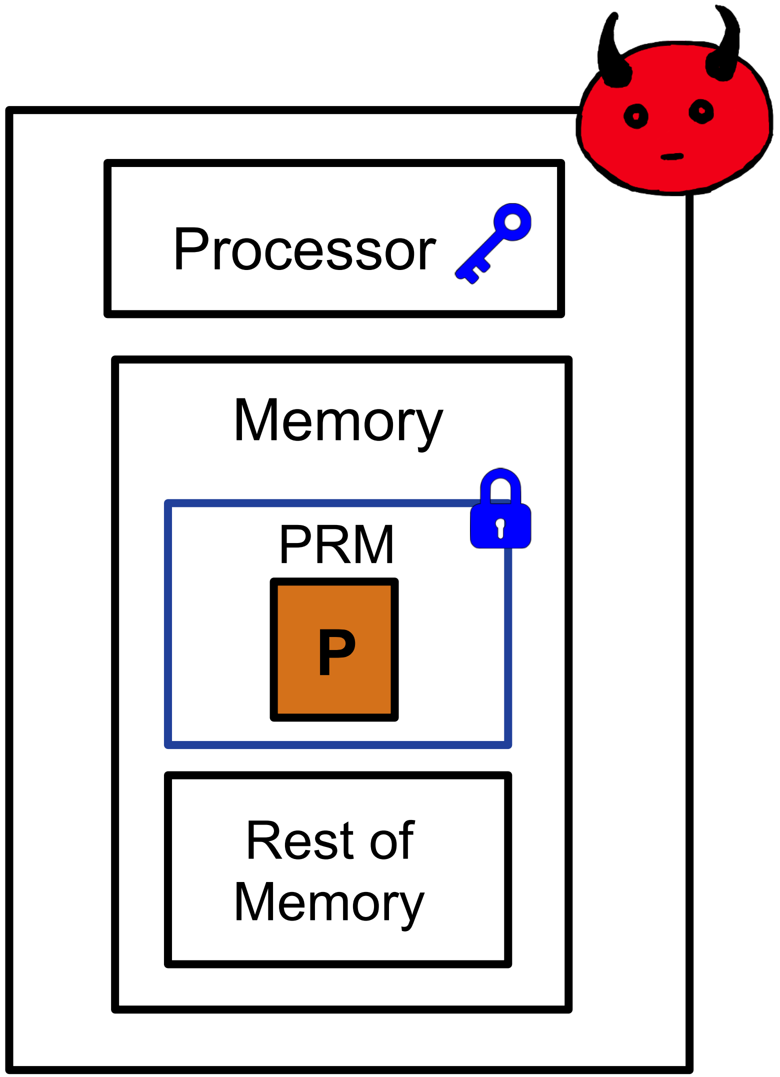
Trusted Execution Environments (TEEs)

- All data within PRM remain encrypted at all times

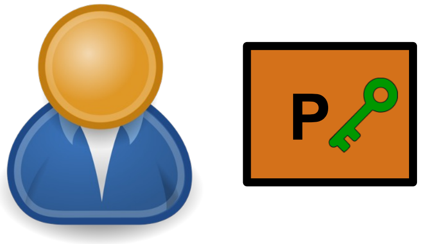


Trusted Execution Environments (TEEs)

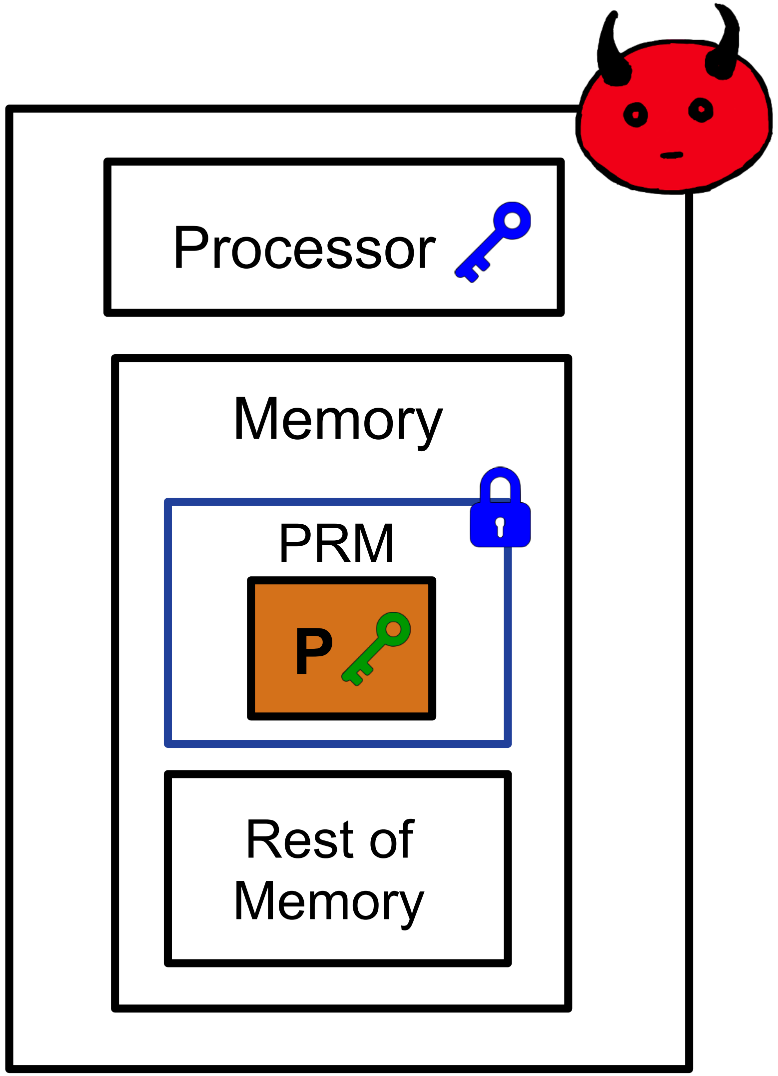
- All data within PRM remain encrypted at all times



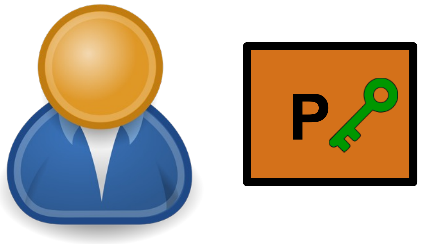
Trusted Execution Environments (TEEs)



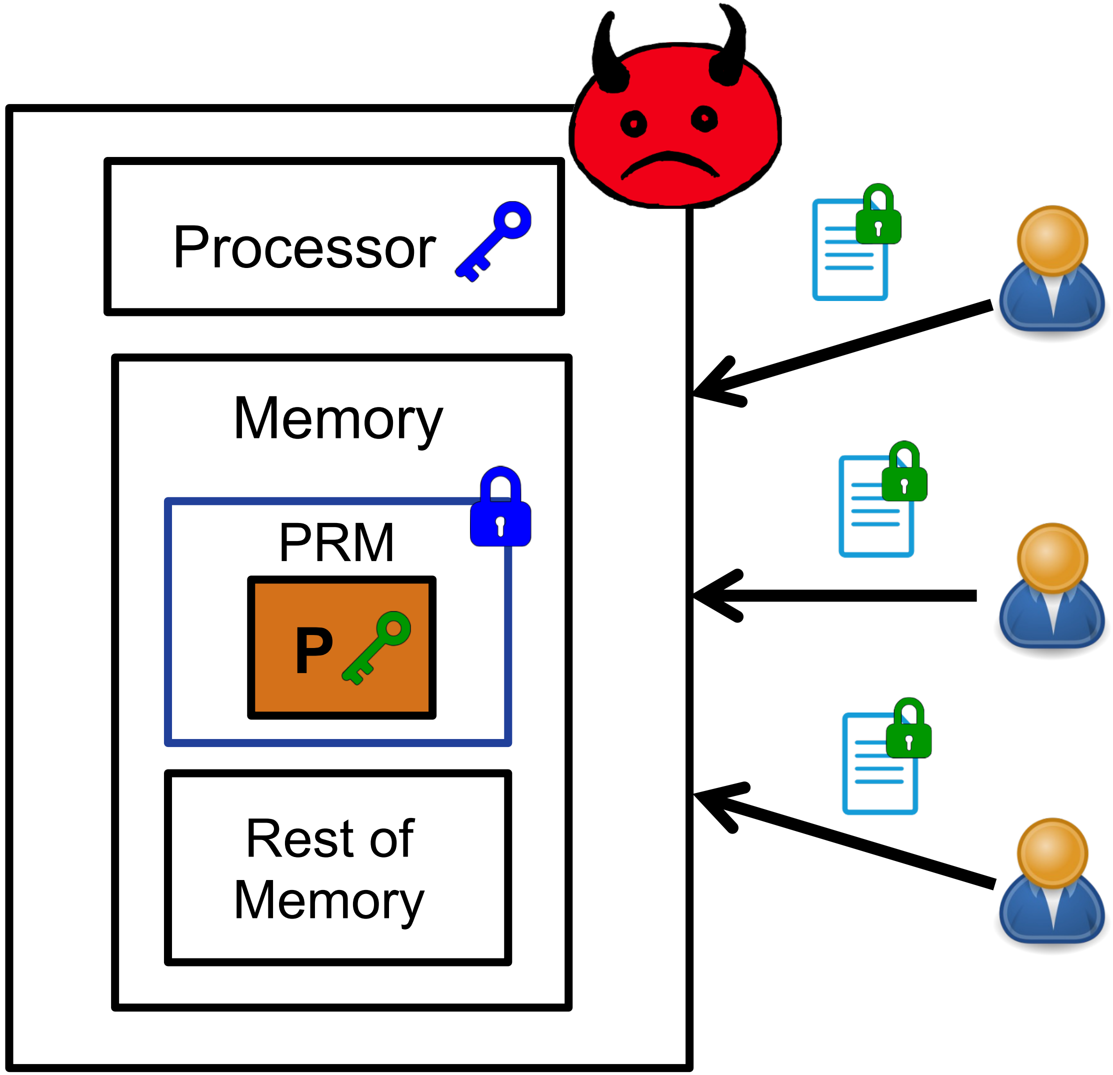
- All data within PRM remain encrypted at all times
- P can have its own key pair enabling users to send private data to P, that only P can decrypt.



Trusted Execution Environments (TEEs)



- All data within PRM remain encrypted at all times
- P can have its own key pair enabling users to send private data to P, that only P can decrypt.

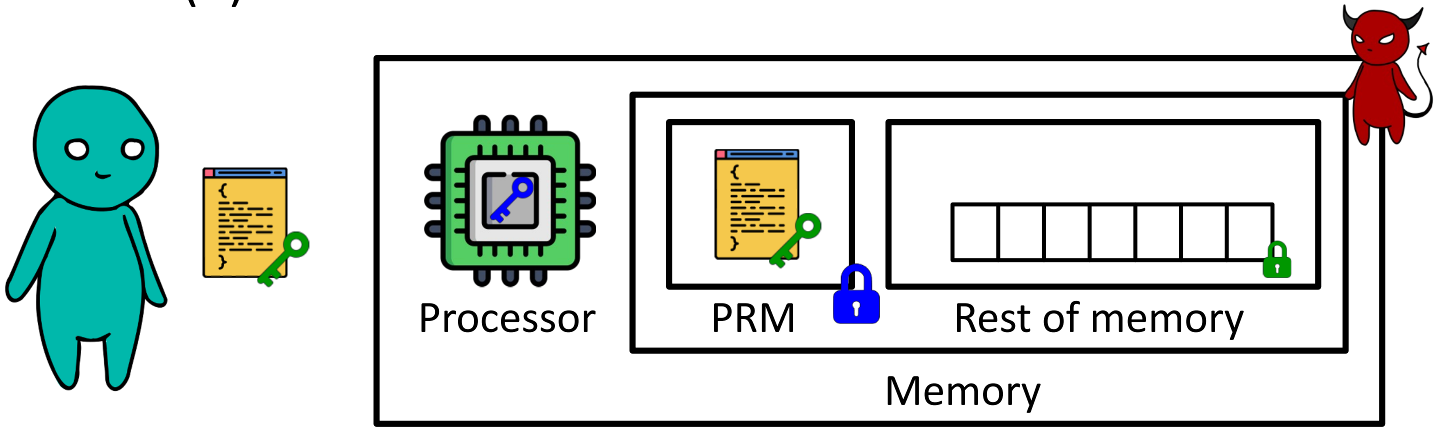


SGX is vulnerable to side channel attacks

```
if (secret==1):
        Branch A
else:
        Branch B
```

Software
Side-channels

(1) Control Flow

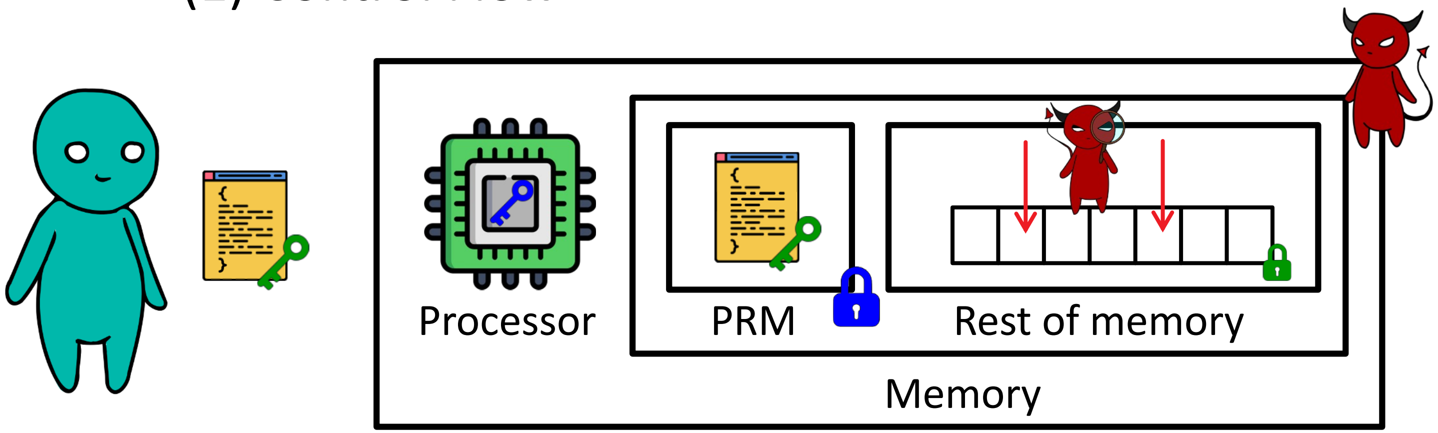


SGX is vulnerable to side channel attacks

```
if (secret==1):
        Branch A
else:
        Branch B
```

Software
Side-channels

(1) Control Flow



Attacks exploit input dependent access behavior on CPU caches, registers, and page faults to uncover plaintext data

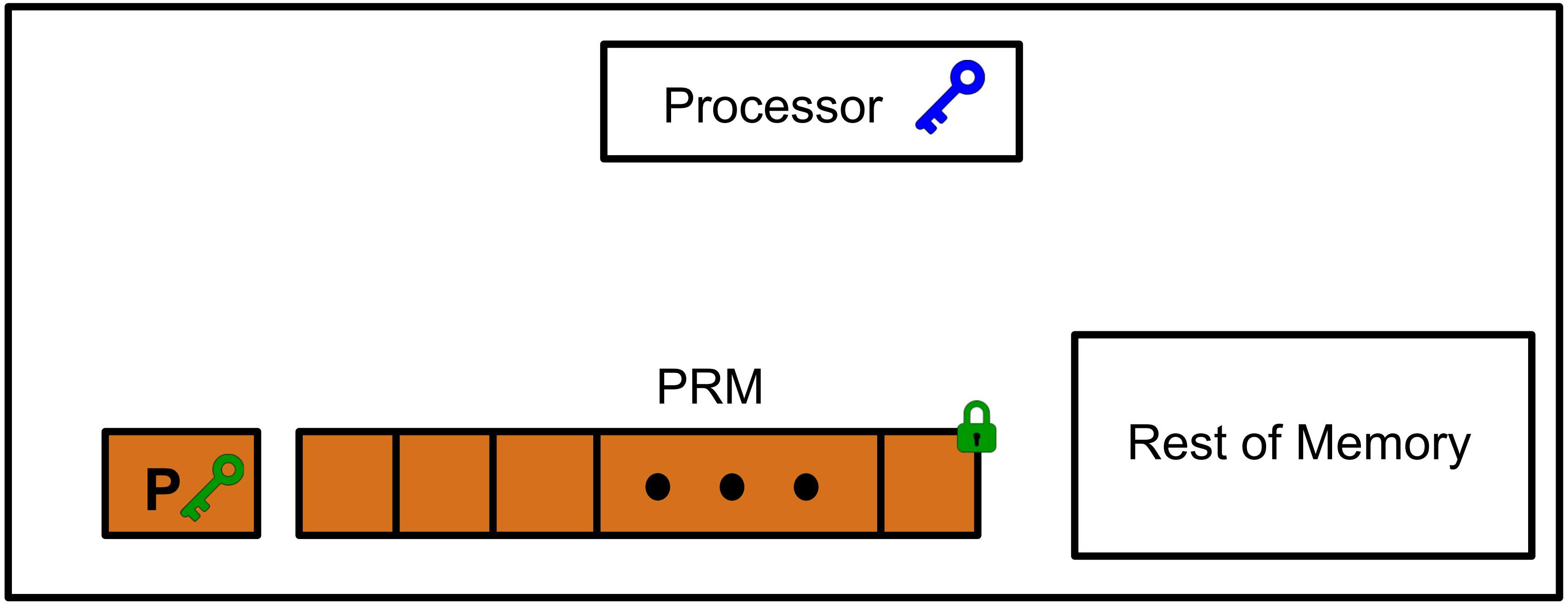
Learn more about attacks: link

(2) Memory Access Patterns

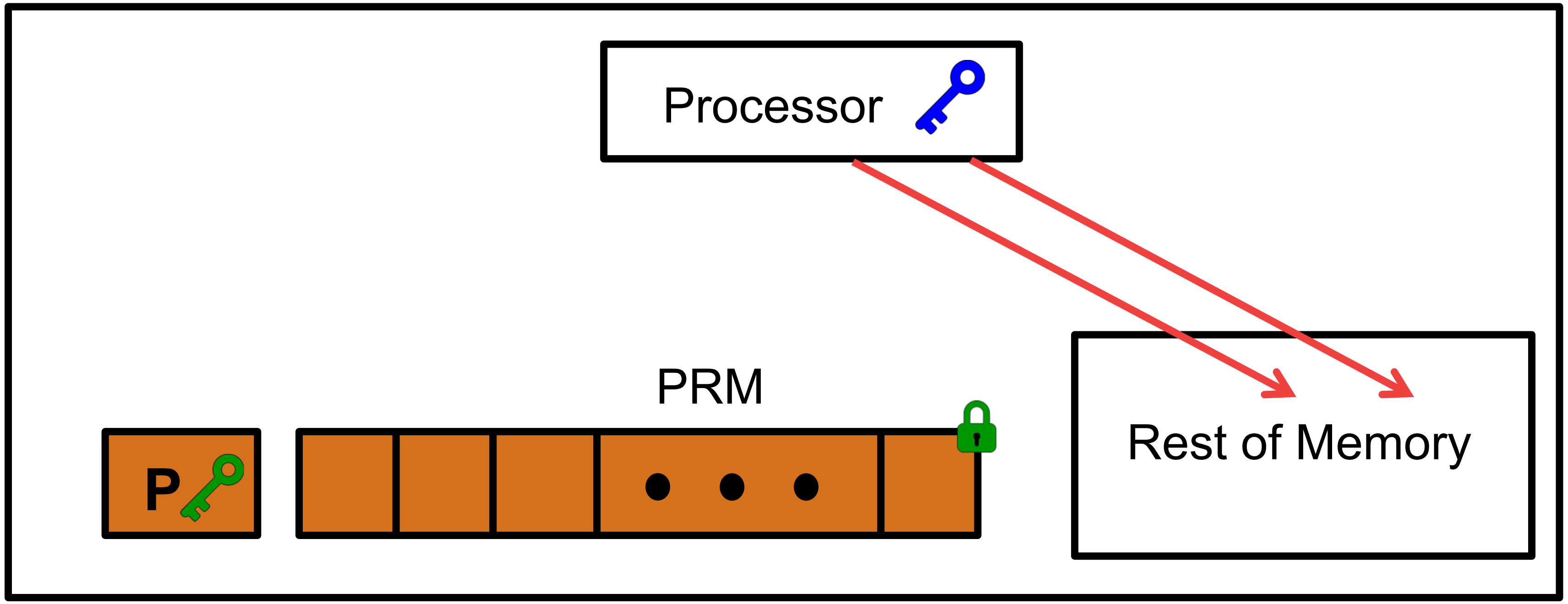
X = A[secret]

We need ***obliviousness***

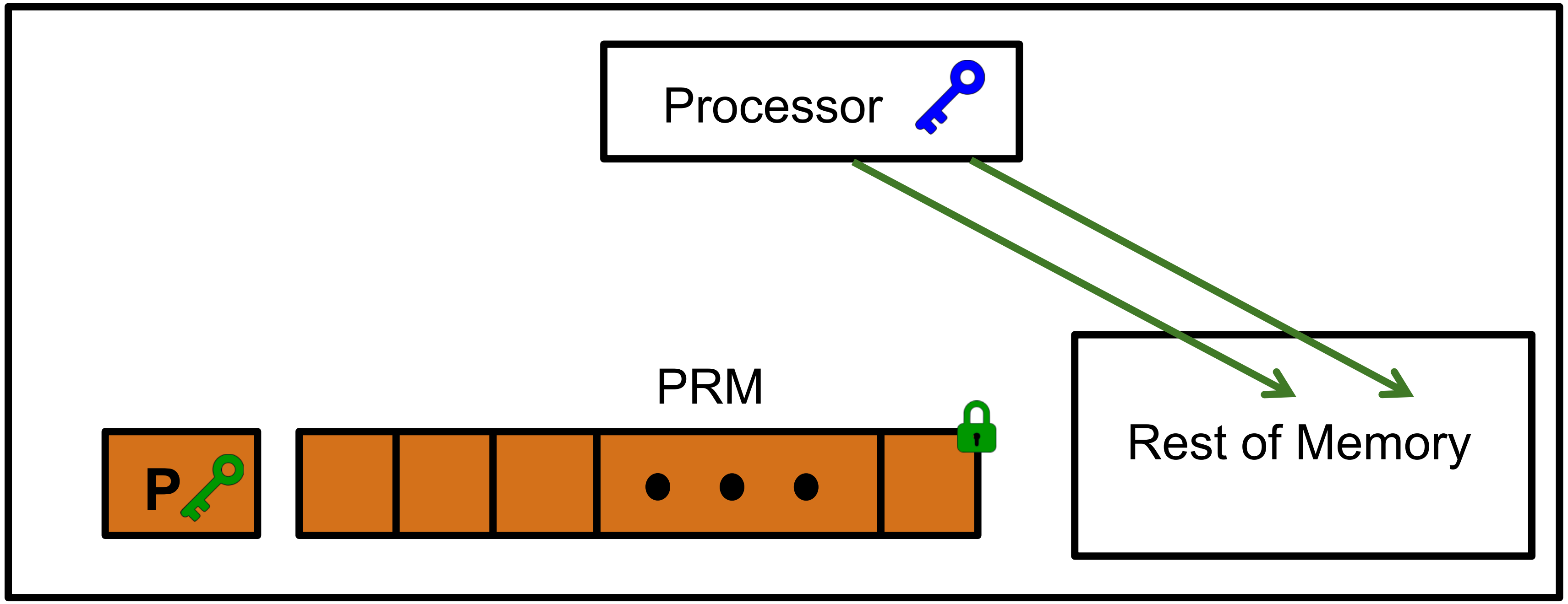
Levels of Obliviousness



Levels of Obliviousness



Levels of Obliviousness

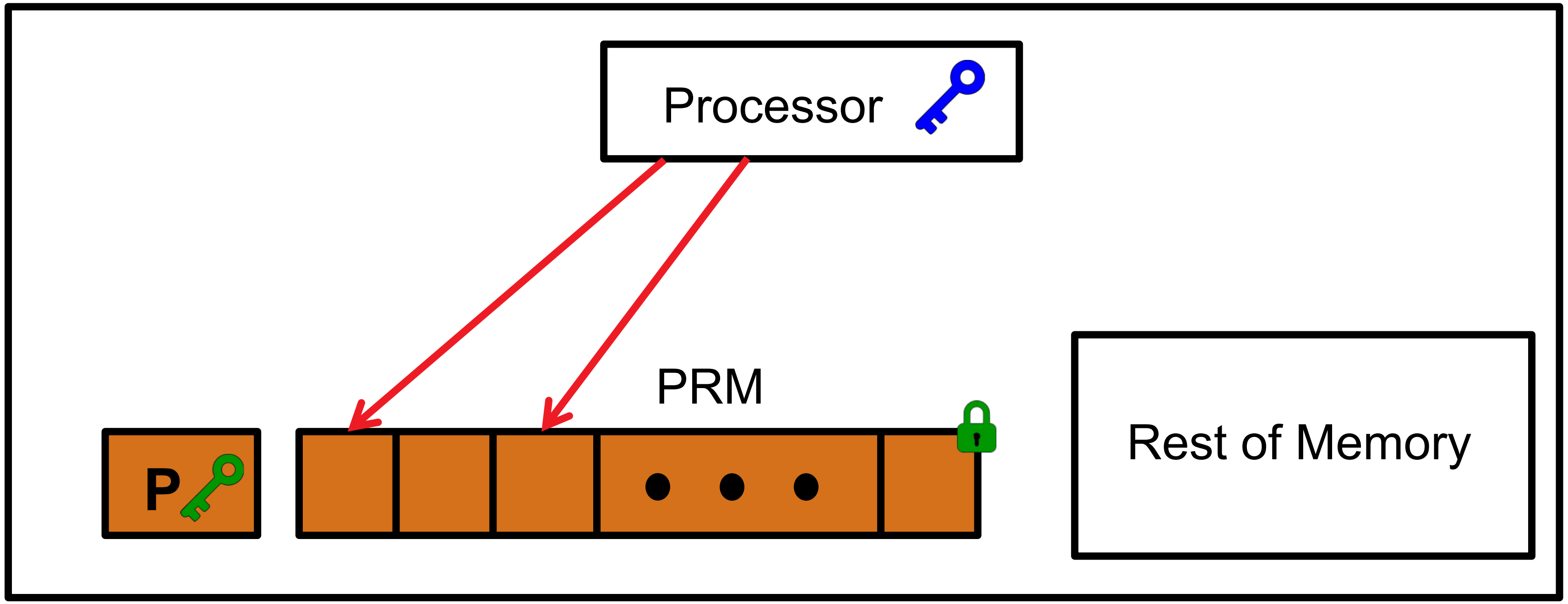


1) External-Memory

External-Memory Oblivious: Access to data outside of the PRM are independent of any secret data.

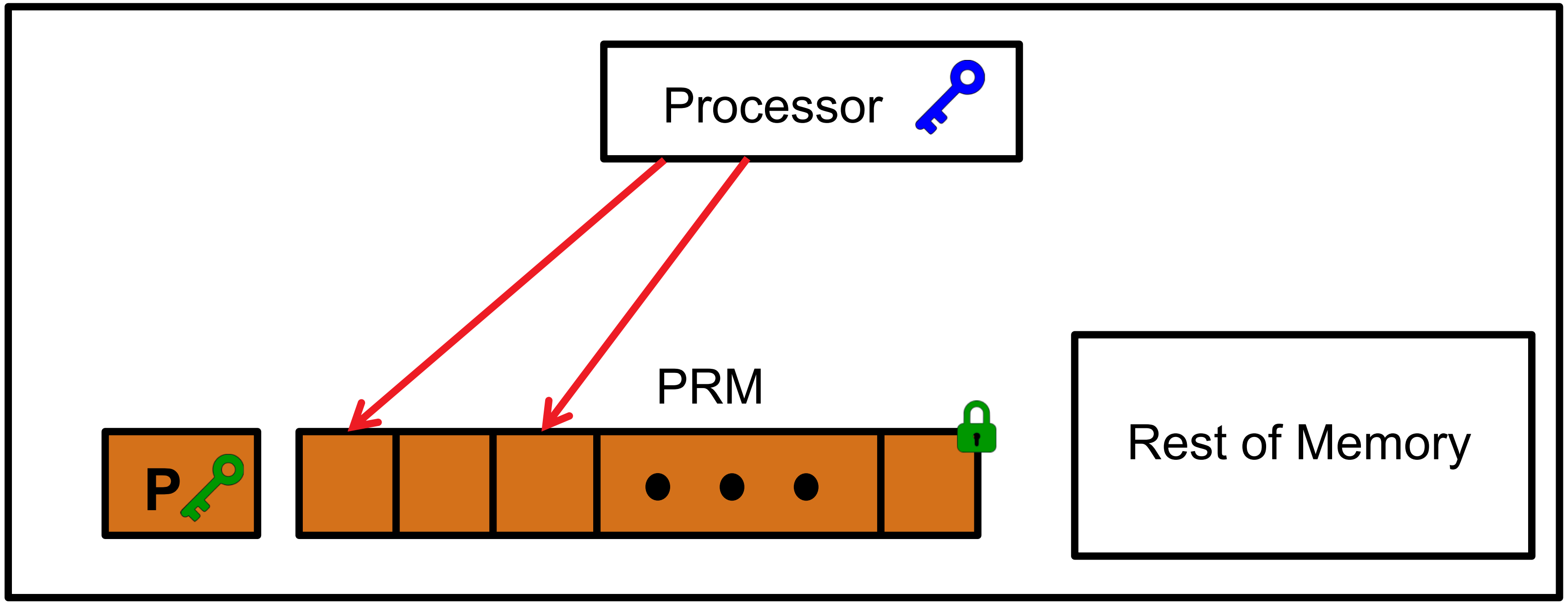
Levels of Obliviousness

1) External-Memory



Protected-Memory Oblivious: Access to data within the PRM are independent of any secret data.

Levels of Obliviousness

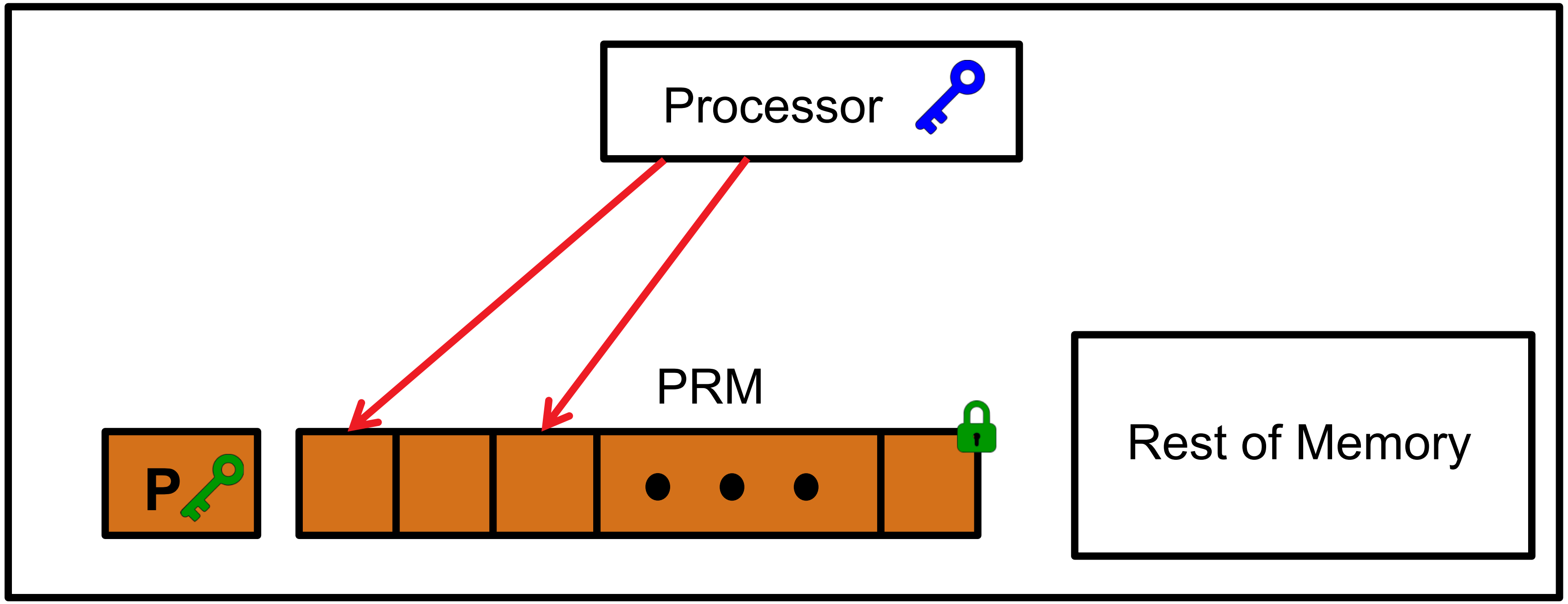


1) External-Memory

2) Protected-Memory

Protected-Memory Oblivious: Access to data within the PRM are independent of any secret data.

Levels of Obliviousness



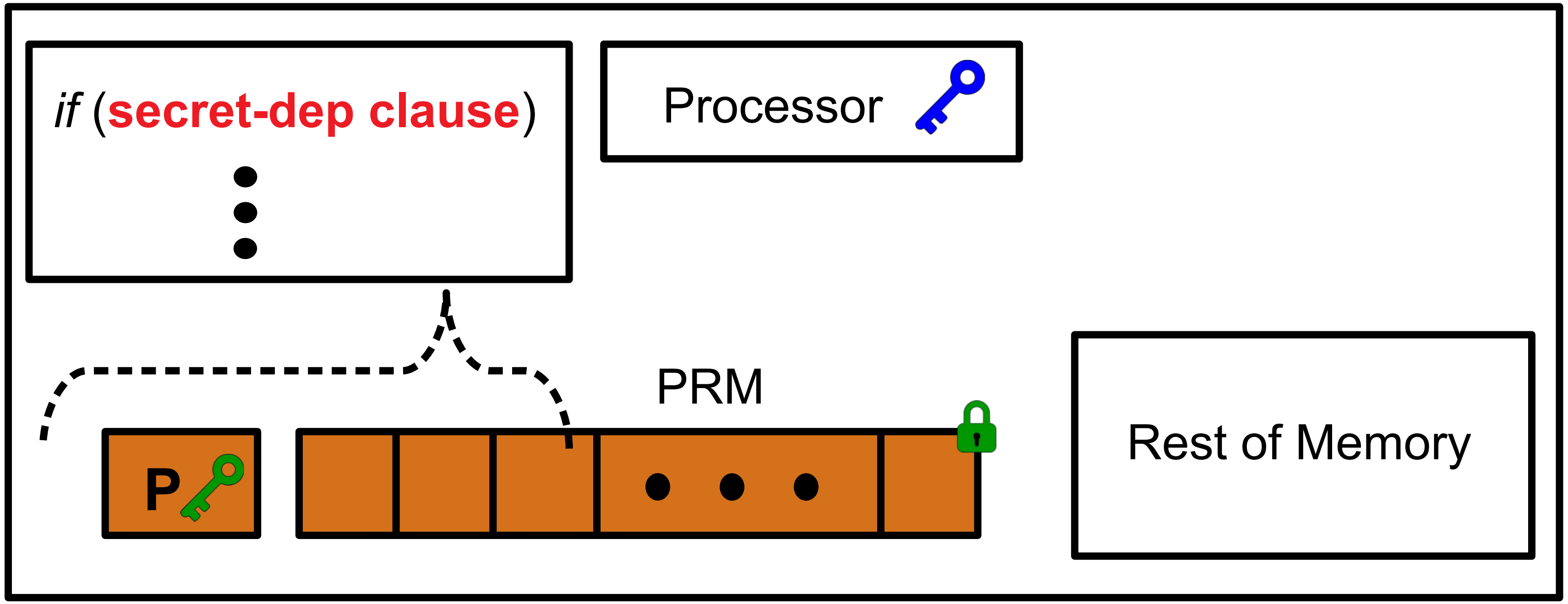
1) External-Memory

2) Protected-Memory
i. Page
ii. Caches

OS is responsible for page table management; Page-granular attacks induce page faults to extract memory locations accessed by the program.

Adversary can observe timing info on caches in the Processor to also launch attacks

Levels of Obliviousness

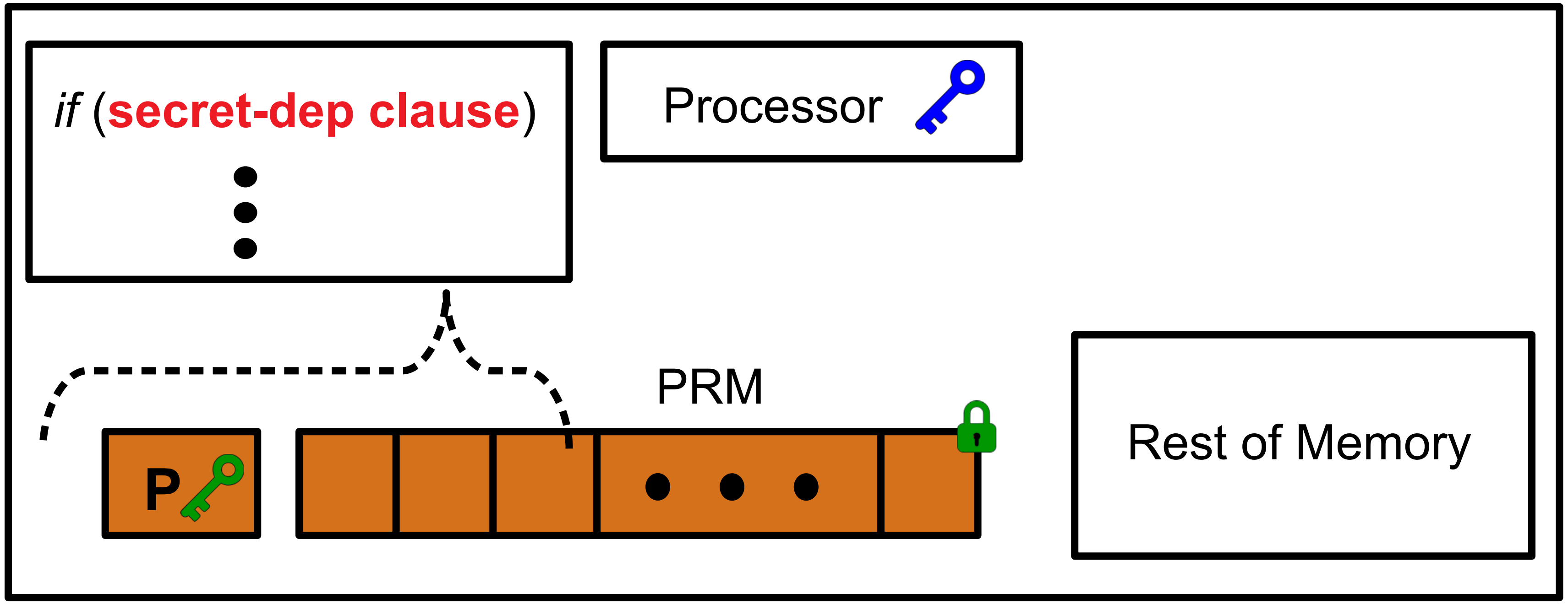


1) External-Memory

2) Protected-Memory
i. Page
ii. Caches

Control-Flow oblivious: Secret-dependent control flow branches leak information about the underlying secret; ensure that the program has no secret-dependent control-flow branches.

Levels of Obliviousness



1) External-Memory

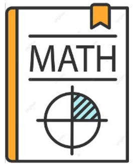
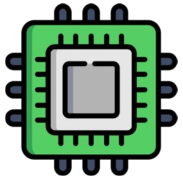
2) Protected-Memory
i. Page
ii. Caches

3) Control flow

Fully Oblivious: A program is fully oblivious if it satisfies all above definitions of obliviousness

Responsibility of the app developer to design oblivious code

Privacy-Preserving Computations

Homomorphic Cryptography	Distributed Trust / Multi-Party Computation	Trusted Execution Environments (TEEs)
Compute directly on encrypted data	Data is secret shared and computed upon by servers	Data computations inside secure containers
Well-understood hardness assumptions 	Non-collusion of computation parties 	Assumes trustworthy hardware 
Impractical overheads	Incurs large bandwidth overheads	Vulnerable to side channel attacks