ETH zürich

BITE: Bitcoin Lightweight Clients Privacy using Trusted Execution

Siniša Matetić (ETH Zurich), Karl Wüst (ETH Zurich), Moritz Schneider (ETH Zurich), Kari Kostiainen (ETH Zurich), Ghassan Karame (NEC Labs) Srdjan Čapkun (ETH Zurich)

28th Usenix Security Symposium, August 14-16, 2019, Santa Clara, CA, USA

Bitcoin - characteristics

BTC Price: 10’344 USD

- *Heavily used*
- *~ 4.1 Tx/s*
- *~ 360k Tx/day*

Bitcoin - characteristics

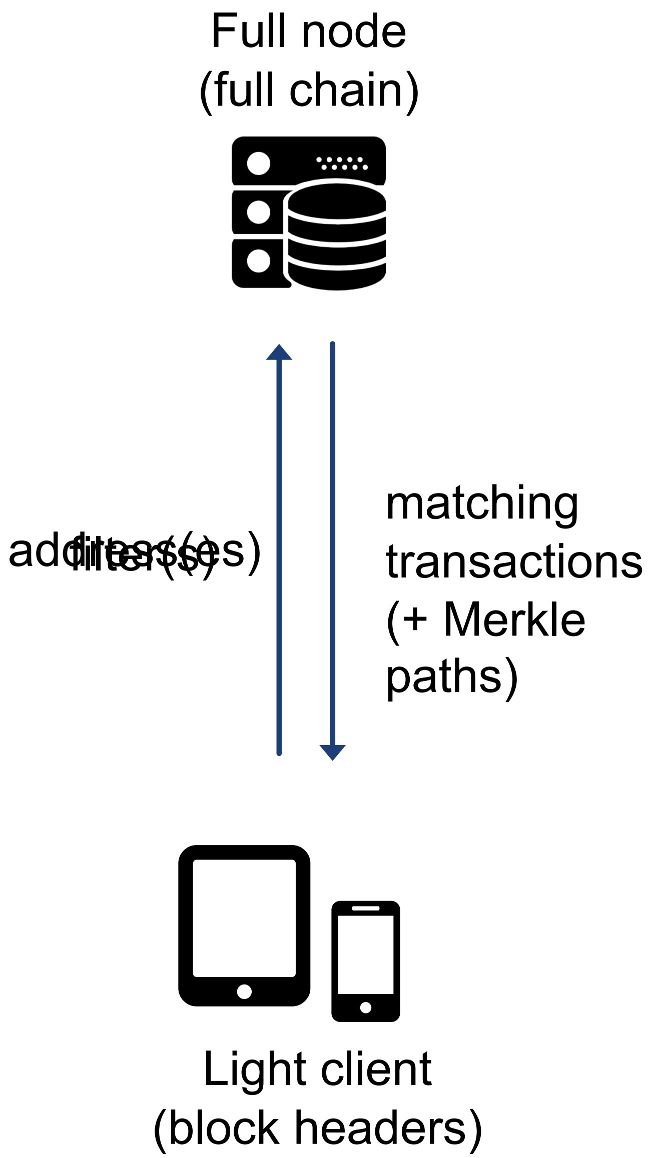
- Significant deployment issue is client requirements
  - Clients need to download and process **entire chain (~230GB)**
  - Participating in the P2P network carries high communication overhead
  - Partial Anonymity achieved through Pseudonymity

- **Implications: using mobile clients for transaction confirmation is infeasible**
  - Many different ”light” clients available for use in mobile (resource constrained) devices

- **Problem:** full reliance on the full node that stores the entire chain
  - Light client stores only block headers, all other information is requested from the full node
  - Fully breaks privacy

Strawman solutions

- Bitcoin supports Simplified Payment Verification (SPV)
  - Works, but sharing the addresses breaks privacy
- Use the same approach with Bloom Filters?
  - Sharing the filters still breaks privacy [1]
- Share addresses with a **TEE?**



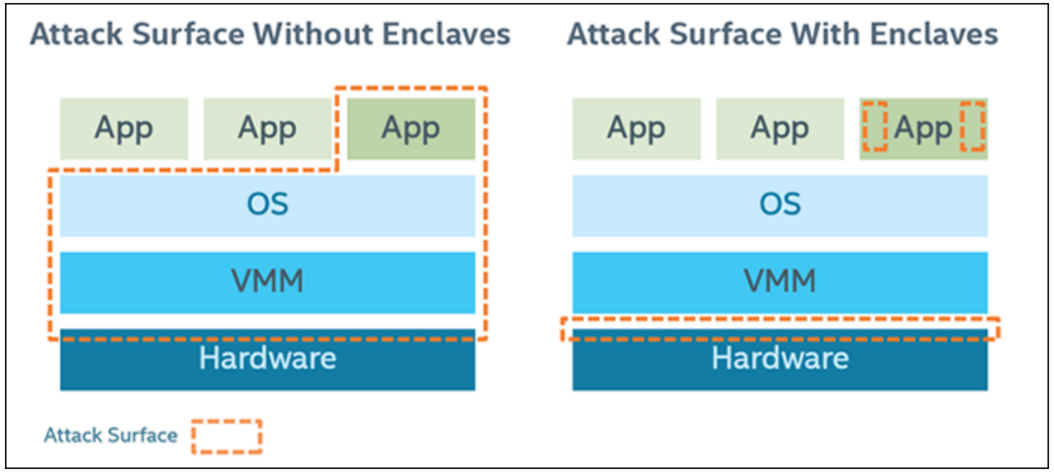
[1] Gervais et al. On the Privacy Provisions of Bloom Filters in Lightweight Bitcoin Clients. ACSAC (2014), ACM.

Trusted Execution Environments

- Enable isolated execution within a user's system
  - Secure, integrity-protected environment
  - Provides processing, memory, and storage capabilities
  - Smart cards, TPM, ARM Trustzone, Keystone, etc.
  - **Intel SGX**

Intel Software Guard Extensions (SGX)

- Intel's architecture containing new instructions, protective mechanisms, and key material in the CPU
  - Runtime isolation, sealing, attestation
  - Memory content encrypted



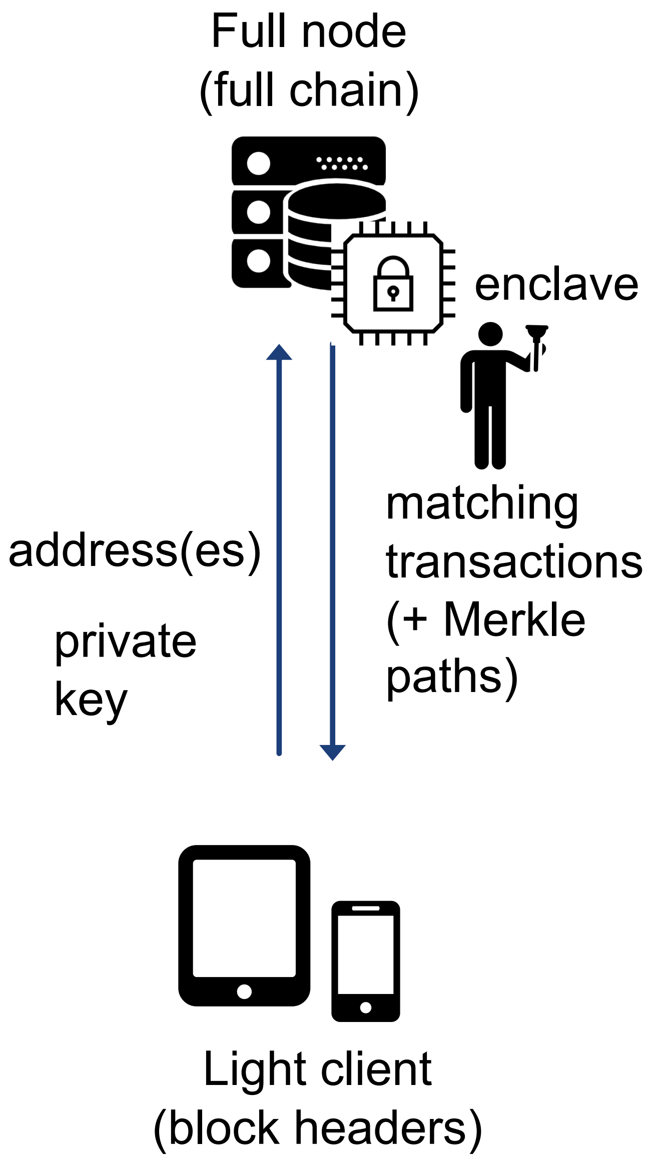
- Trust model
  - CPU and protected enclaves
  - Untrusted system software

- NOTE: Recent works show successful compromise of such environments
  - Side-channel attacks, Spectre, Meltdown, Foreshadow

Images taken from software.intel.com

Strawman solutions - continued

- Bitcoin supports Simplified Payment Verification (SPV)
  - Works, but sharing the addresses breaks privacy
- Use the same approach with Bloom Filters?
  - Sharing the filters still breaks privacy
- Share addresses with a **TEE (SGX enclave)**?
  - Better… but enclaves leak and privacy is still a problem
  - Side-channel attacks
- Send also the private key to the full node?
  - If enclave compromised, client looses all money



Isolated execution and leakage - challenges

- CPU enforces that other software cannot access enclave memory
  - But **physical resources** are shared
- Side-channels were a known threat
  - Original SGX docs: “*software side-channels may be possible*”
  - Page-fault attacks demonstrated soon after release
- Essentially, SGX itself does not provide protection against external and internal information leakage

How to prevent side-channels on SGX?

- Side-channel resilient **implementation** (Intel recommendation)
  - Difficult to apply for all enclaves
- Developer **annotation** (Cloak, Raccoon)
  - Difficult to assess what might leak
- Address **specific attack vectors** (T-SGX, DejaVu)
  - Does not prevent all attacks
- **Private information retrieval** (ORAM) for every memory access
  - Very high overhead
  - Control-flow and timing leakage → **oblivious execution**

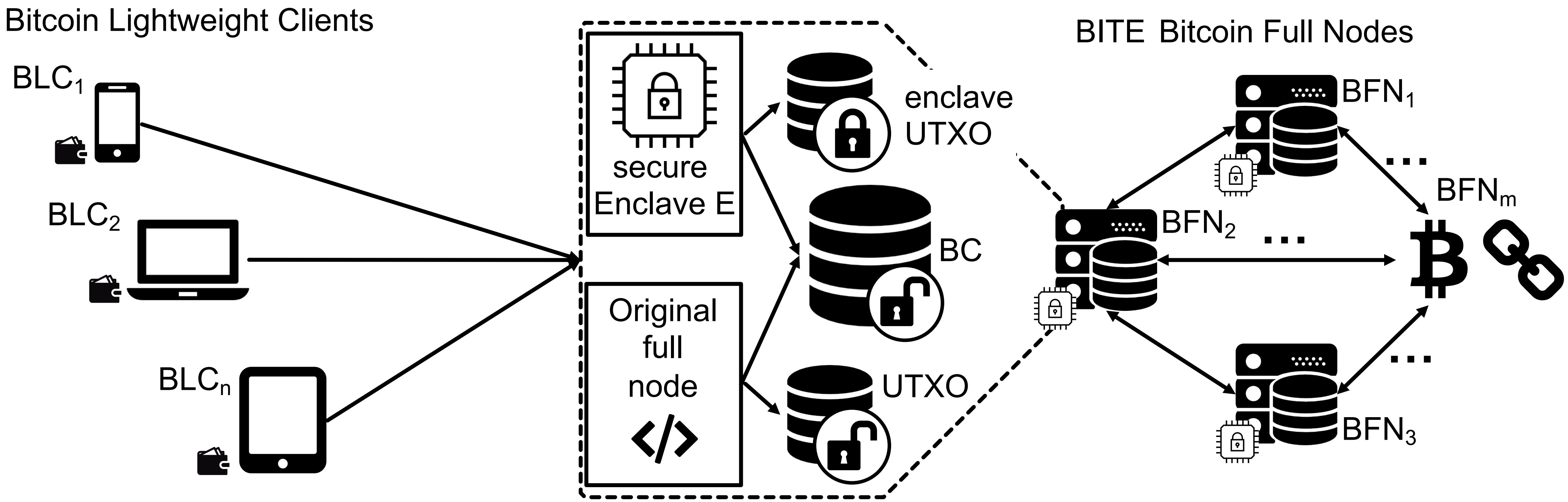
Our solution: BITE – transaction fetching and verification

- Light client shares the **adresses** with the enclave on the full node

- **Enclave hardened** using known techniques
  - **Memory access:** in-memory ORAM to prepare a response
  - **Control flow:** secret-dep branching removed using CMOV [Raccoon]
  - **Response:** Fixed ratio between response size and scanned blocks

- **Two variants** – *Scanning Window* and *Oblivious Database*

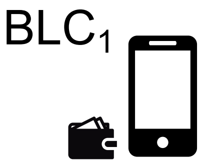
BITE: System Model and protocol overview

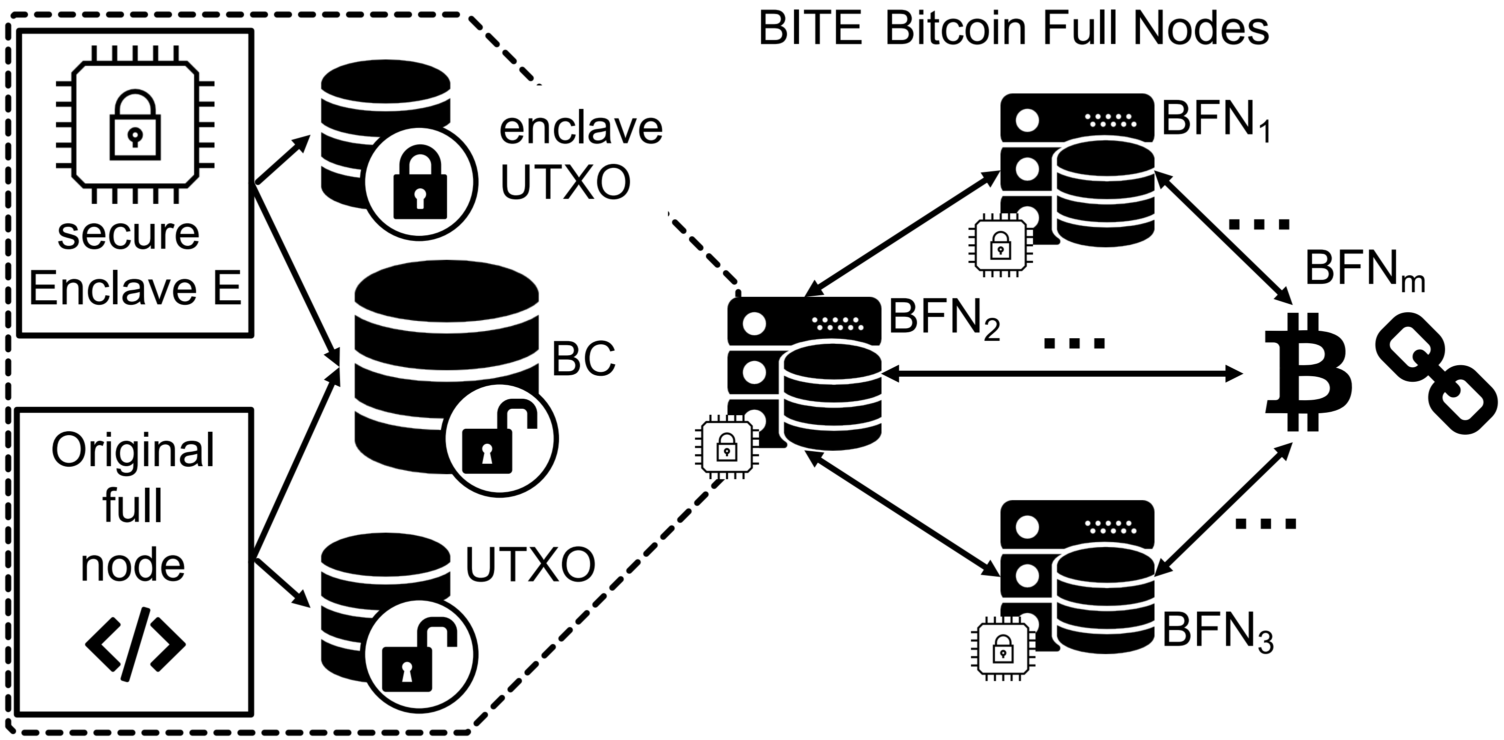


ETH zürich

BITE: System Model and protocol overview

Bitcoin Lightweight Clients

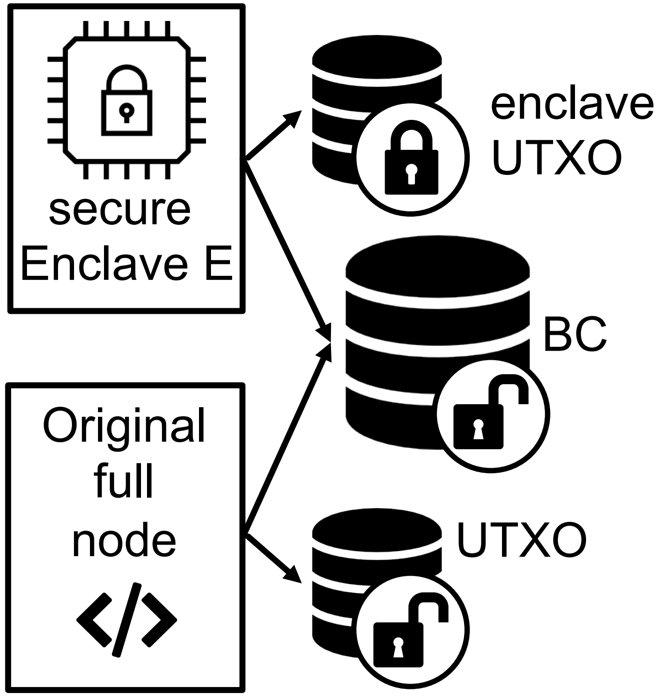




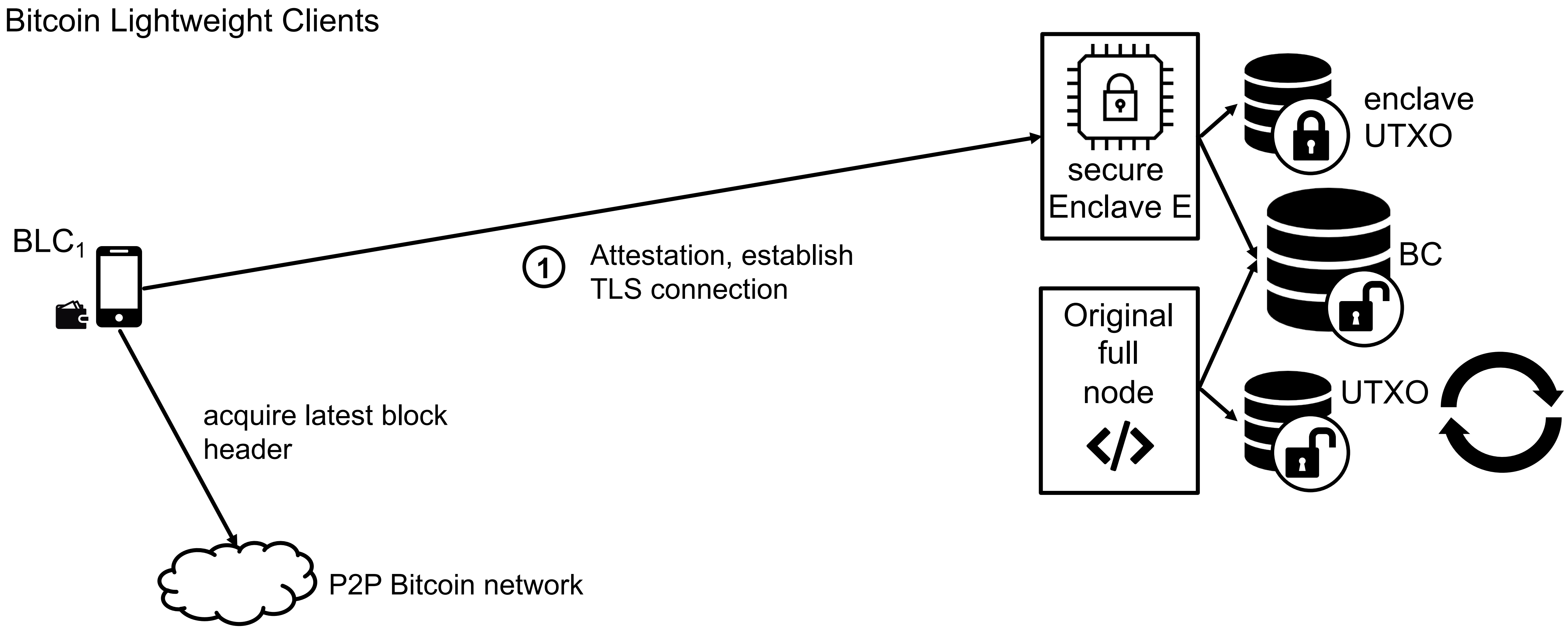
BITE: System Model and protocol overview

Bitcoin Lightweight Clients

BLC$_1$



BITE: System Model and protocol overview

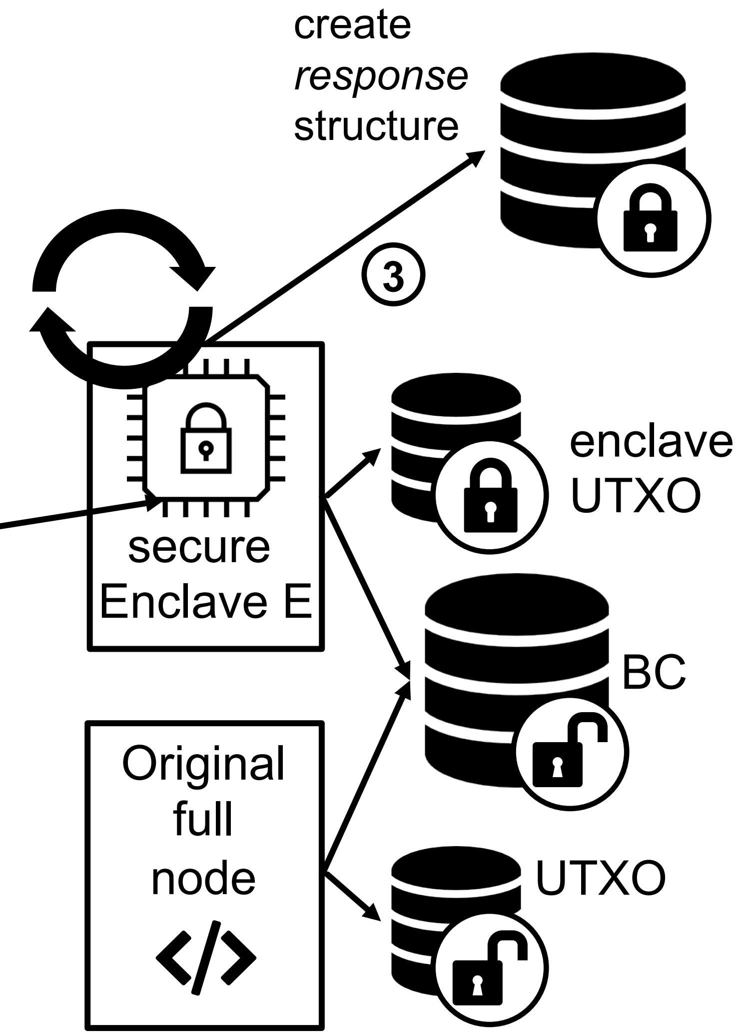


BITE v1: Scanning Window

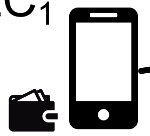
BITE v1: Scanning Window

Bitcoin Lightweight Clients

④ scan BC from last block to h
```
if  block contains transactions
    ->(move TXs to response)
    ->(move MPs to response)
else
    ->(move header to response)
```



BLC_1

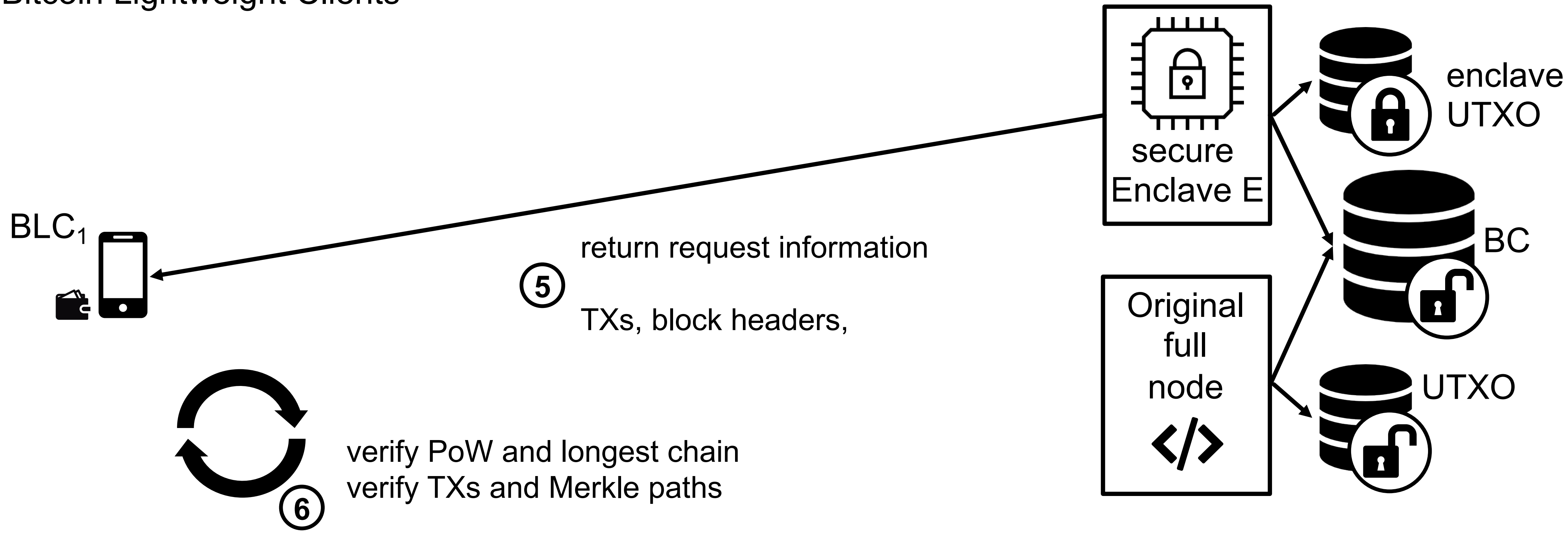


② Information request for transaction retrieval

Address$_{1,2...n}$
Block height **h**

BITE v1: Scanning Window

Bitcoin Lightweight Clients



BITE v2: Oblivious Database

BITE v2: Oblivious Database

Bitcoin Lightweight Clients

Search through enclave UTXO ④

`if` UTXO contains transactions

-›(move TXs to response using ORAM)

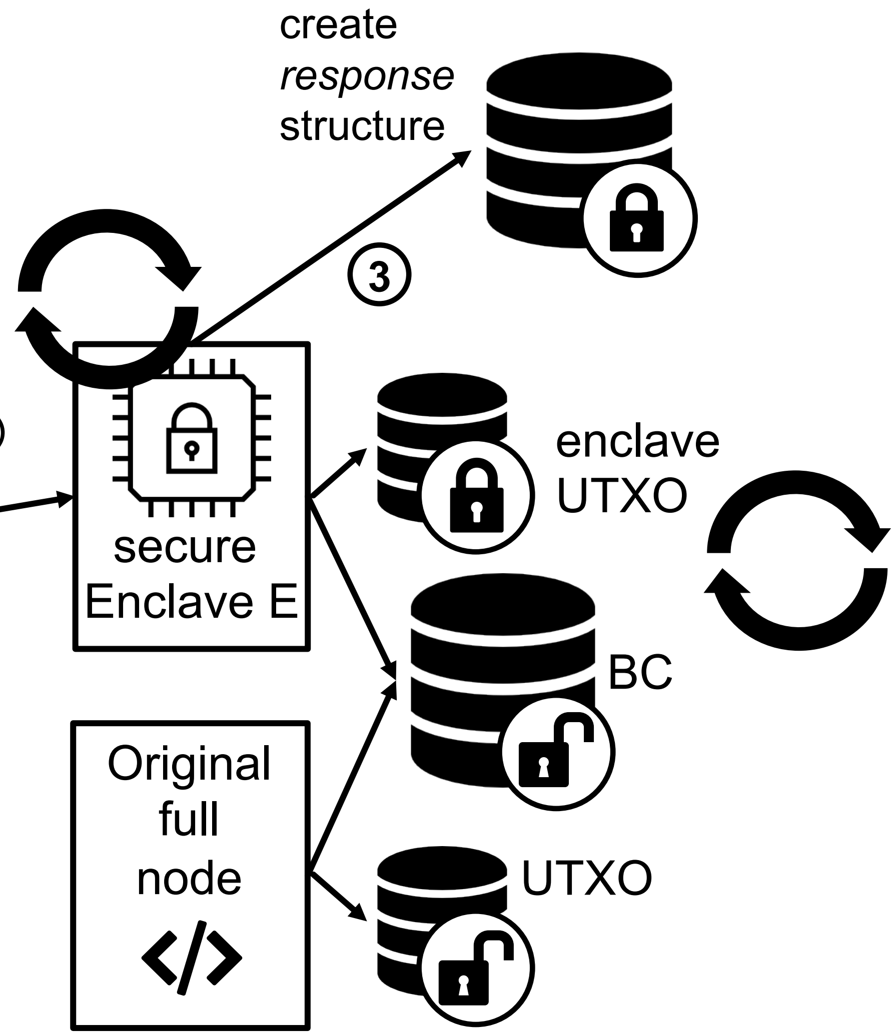
`else`

-›(move nothing to response using ORAM)

BLC_1

② Information request for transaction retrieval

Address$_{1,2\ldots n}$
Block height **h**

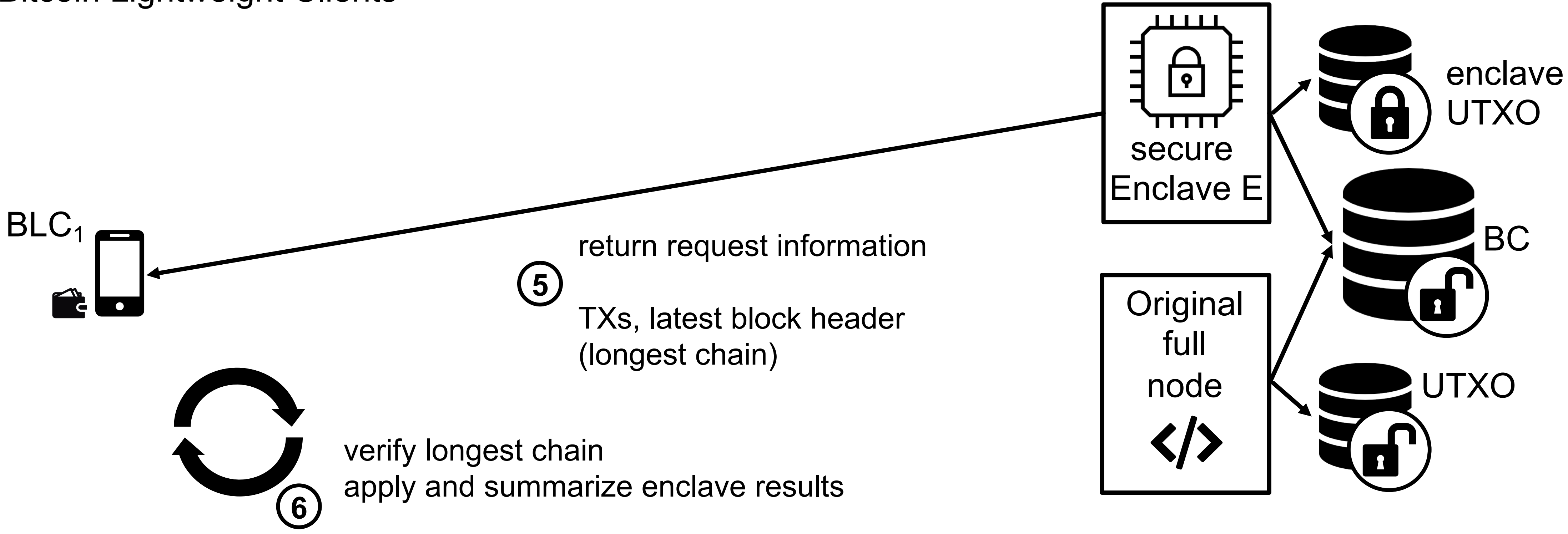


Build *enclave UTXO*

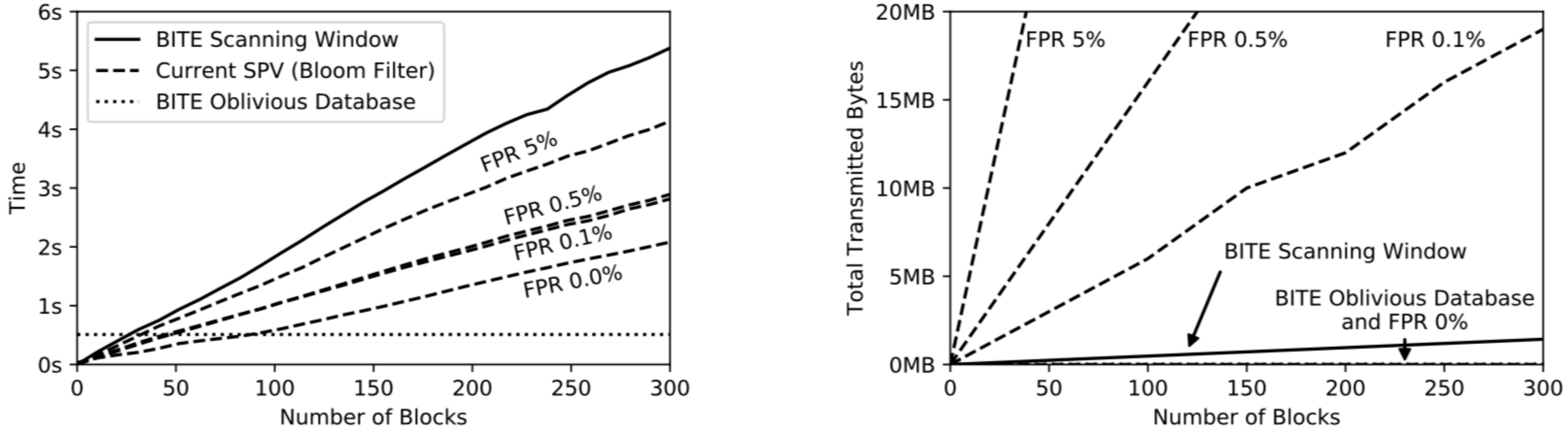
encrypted, indexed and accessed using ORAM

BITE v2: Oblivious Database

Bitcoin Lightweight Clients



Performance



(a) **Processing cost (client request)** for Scanning Window, Oblivious Database and current SPV protocols using Bloom filters.

(b) **Communication cost** for Scanning Window, Oblivious Database and current SPV protocols using bloom filters.

Figure 6: Performance evaluation of Scanning Window and Oblivious Database.

Results

- **BITE is the first practical solution** enabling strong privacy protection for Bitcoin light clients
  - BITE provides all the necessary data for light clients in order to verify and create transactions
- BITE tolerates strong adversary
  - Malicious full node that performs side-channel attacks on enclave
  - Monitors control flow (instruction-level) and data accesses (byte-granularity)
- Graceful failure
  - In the case of full break of SGX, clients don’t lose money

Thank you for your attention! Questions?

Siniša Matetić

ETH Zürich

System Security Group

Institute of Information Security

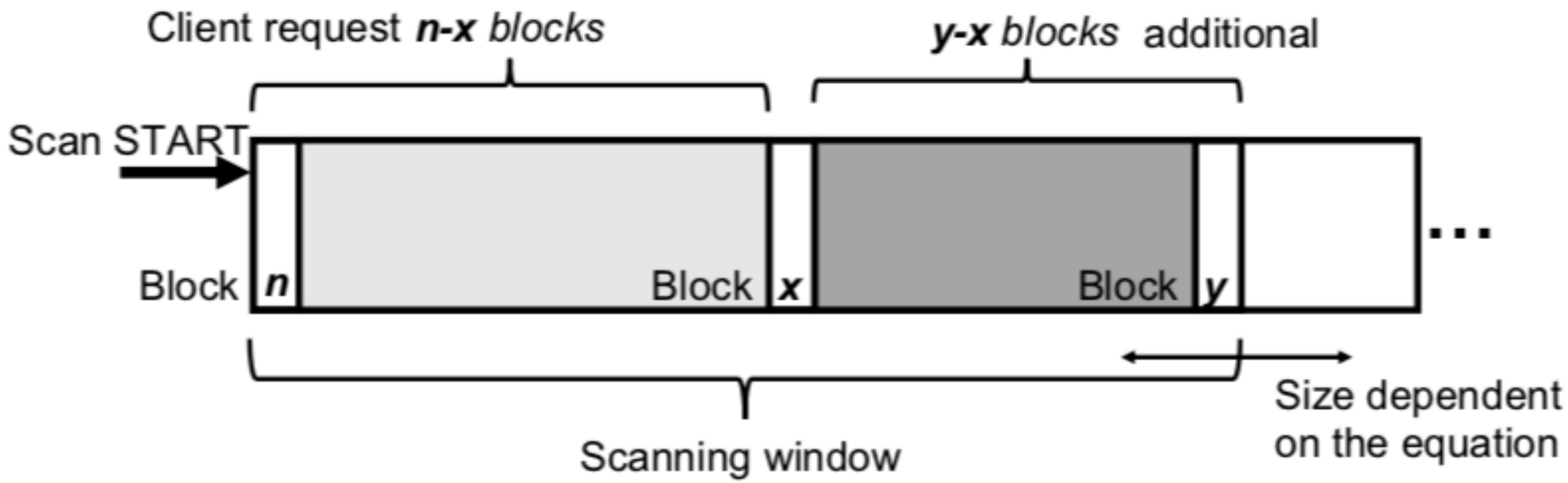
Department of Computer Science

www.syssec.ethz.ch

sinisa.matetic@inf.ethz.ch

Backup

BITE: Scanning Window mechanism



	Processing		Communication	Storage		
	Request	UTXO Update	Response	Blockchain	UTXO	Leakage Protection
Scanning Window[1]	1.9s	-	500kB	200GB	0	✓/✗[4]
Oblivious SW[1]	73s	-	500kB	200GB	0	✓
Oblivious Database[3]	0.5s	78.5s	12kB	50MB[5]	6GB	✓
Stan. SPV FPR 0.5%[1]	1.1s	≈2s	17MB	200GB	2.8GB	✗
Stan. SPV FPR 0.0%[1,2]	0.6s	≈2s	14kB	200GB	2.8GB	✗

[1] For 100 blocks. [2] SPV with no privacy protection. [3] For 10 addresses.
[4] Protects against external leakage but not side-channels.
[5] Only the block headers need to be stored.

Table 3: Performance comparison and requirements on the full node for supporting light clients.

	Leakage			Performance Overhead
	External	Internal	Response Size	
Raccoon[47]	✗	✓	✗	~ 100x[1]
Obliviate[14]	✓	✗	✗	> 4x[1]
Raccoon[47] + Obliviate[14]	✓	✓	✗	$100x - 400$x[2]
BITE Scanning Window [3]	✓	✓	✓	40x
BITE Oblivious Database	✓	✓	✓	1x

[1] Based on the performance evaluation of [47] and [14].
[2] Combination of the two primitives can yield an overhead in this range.
[3] Fully oblivious Scanning Window variant.

Table 4: Performance overhead and security comparison between existing primitives and BITE.

		t_m		
		5kB	10kB	20kB
Blocks	100	0.7s (± 0.2s)	1.3s (± 0.5s)	2.7s (± 0.9s)
	200	0.7s (± 0.2s)	1.4s (± 0.5s)	2.8s (± 0.9s)
	300	0.7s (± 0.2s)	1.5s (± 0.5s)	3.0s (± 0.9s)

Table 2: Processing time per block with oblivious execution for Scanning Window depending on the number of requested blocks and the temporary size, averaged over 100 blocks.

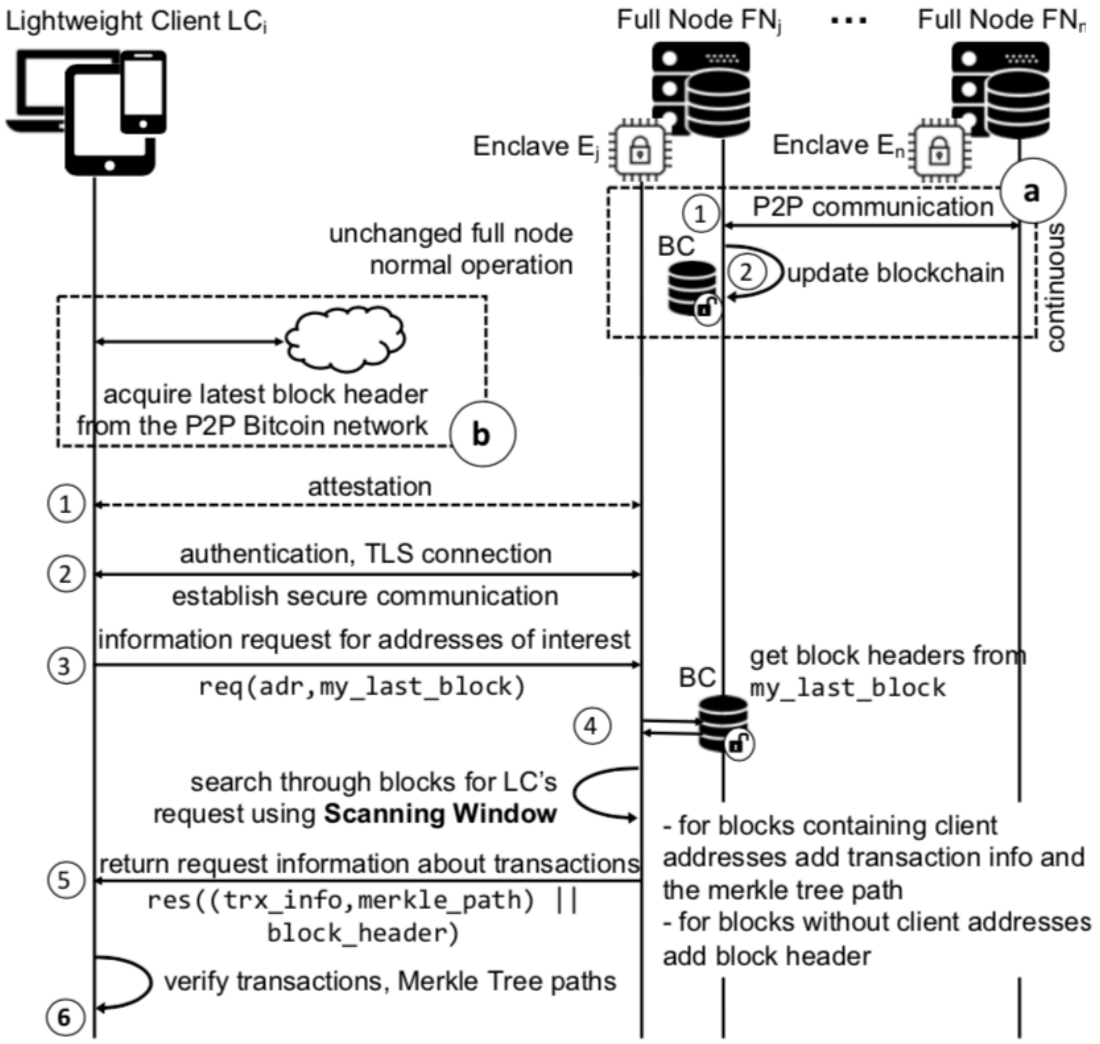


Figure 2: **Scanning Window operation.** Light client creates a secure connection to an enclave on full node and sends a request with its address and last known block. The enclave scans the locally stored chain and prepares a response with the size proportional to the number of scanned blocks.

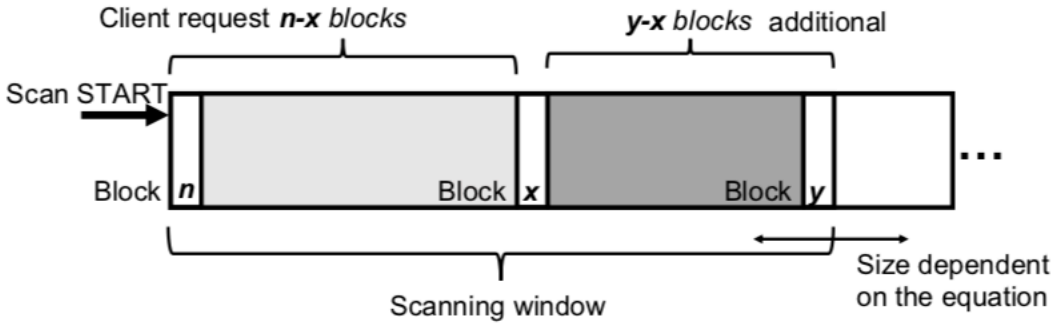


Figure 3: **Block reading in Scanning Window**. Depending on the number of requested blocks (up to x) and the number of matching transaction in them, we read potentially extraneous blocks (up to y) to keep the ratio between the read blocks and the response message size constant.

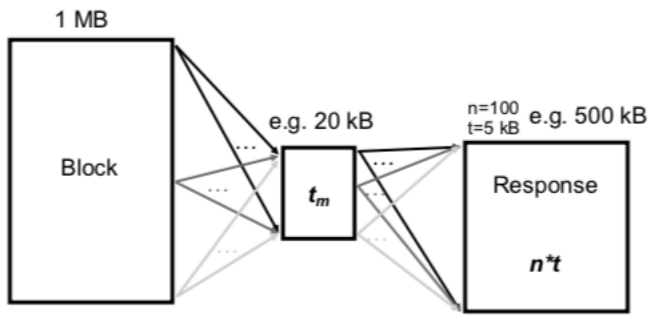


Figure 4: **Oblivious copying in Scanning Window.** The data is copied in an oblivious fashion from the block to a temporary array, i.e., every transaction is conditionally moved using `cmov` to every possible destination. The data contained in the temporary array is then copied to the response in an oblivious fashion, again using `cmov` to conditionally copy everything to all possible locations in the response.

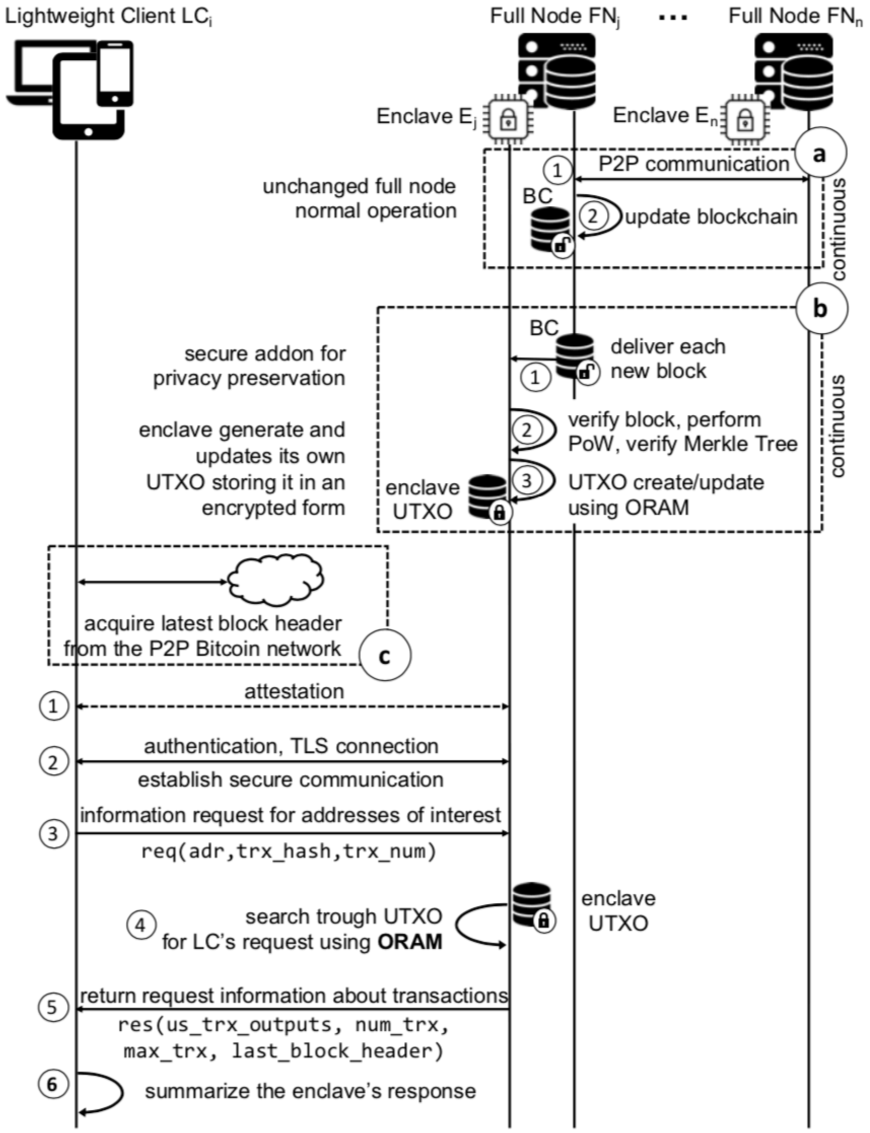


Figure 5: **Oblivious Database operation.** Lightweight client sends a request containing its address and the last transaction to an enclave on full node. Enclave queries a specially-constructed UTXO database using ORAM and provides a response back to the client.