SSP: Self-Sovereign Privacy for Internet of Things using Blockchain and MPC

Tiffany Hyun-Jin Kim Joshua Lampkins
HRL Laboratories, LLC
{hjkim,jdlampkins}@hrl.com

***Abstract*—With the proliferation of the Internet of Things (IoTs), their devices have been collecting our personal data to enhance our life styles and their purposes. Care must be taken when handling such sensitive data to protect the data owners' privacy while guaranteeing the integrity when the data is accessed by a third party. Furthermore, data should be available without much delay to support medical emergencies. Unfortunately, IoT devices may be limited with storage and computational resources to operate highly secure operations and respond to data requests. In this paper, we propose SSP, a novel mechanism that protects the privacy and integrity of the data collected by IoT devices without having a single point of failure, while minimizing the cryptographic operations that must be performed on IoT devices. SSP utilizes the blockchain data structure for availability and integrity, and Multi-Party-Computations (MPC) for privacy protection. We implemented SSP in C/C++ and according to our evaluation results, SSP is efficient in computation, communication, and storage operations on IoT devices.**

I. Introduction

The proliferation of smart sensors and devices has been impacting our daily lifestyles, providing convenient and seamless experiences while raising some concerns. With the estimation of 8.4 billion connected Internet-of-Things (IoT) devices in use worldwide in 2017 and projection of the number to increase to 50 billion by 2020,[1] one critical aspect is to reconsider the security and privacy issues from their operations. In fact, Gartner expressed a concern that the biggest inhibitor to growth will be the *absence of security by design* [1].

Indeed, one of the most dangerous security concerns is that users are constantly being monitored, oftentimes surrendering their privacy since they are unaware of what types of data the smart devices are collecting and how the data is being used. For example, EPIC identified the unlawful surveillance of "Always On" IoT devices under federal wiretap law, which included Samsung SmartTV, Google Chromium browser, Mattel "Hello Barbie", Microsoft Kinect, Amazon Alexa, Nest Labs "Nest Cam", and Canary Connect home security systems.[2] All these devices collected the audio or video recordings, and some devices transferred the data without encryption to a third-party voice-to-text encoder. Such an unlawful surveillance was possible by abusing the well-known behavior that people rarely read the privacy policy statement.

While it is crucial to protect the privacy of the data, empowering the data owners to control its usage is desirable.

[1]https://gtnr.it/2Mcqz56
[2]https://epic.org/privacy/internet/ftc/EPIC-Letter-FTC-AG-Always-On.pdf

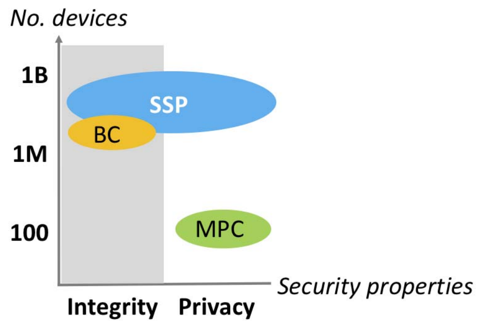

Fig. 1. SSP's security properties. By combining the benefits of blockchain (BC) and Multi-Party Computation (MPC), SSP aims at providing both integrity and privacy guarantees on the data being generated by the IoT devices.

This paper attempts to provide such security features and introduces Self-Sovereign Privacy (SSP) for IoT devices. SSP utilizes the benefits of blockchain and multi-party computation (MPC) for integrity and privacy guarantees of personal data while enabling the data owner to assign access permissions. More specifically, SSP uses the blockchain (BC) infrastructure, where the BC nodes store the encrypted data in a distributed manner, and uses the MPC, where the MPC servers control the release of the keys and the access rights to enable only the authorized parties to access the private data. As long as the majority of the participating blockchain nodes and MPC servers are not compromised, SSP guarantees to protect the integrity and privacy of the personal data collected by the IoT devices (see Figure 1). Oftentimes, providing strong security guarantees require significant computational and communication requirements. To show that SSP is lightweight and suitable for IoT devices that are oftentimes limited in storage, computational, and communication capacities, we implemented our protocol in C/C++. Our experiments indicate that SSP incurs minimal overhead in computations, communications, and storage.

Contributions. We introduce Self-Sovereign Privacy (SSP) as a mechanism to provide the secrecy on the personal data that is being collected by the smart IoT devices.

- SSP is a lightweight protocol using blockchain and MPC to provide the integrity and privacy of the data collected by IoT devices.
- SSP guarantees to provide the security guarantees as long as no more than $1/2$ of the blockchain nodes and $1/2$ of the MPC servers are compromised.

- SSP has been implemented in C/C++ and the evaluation results confirm its suitability for IoT devices.

II. Background

Blockchain. A blockchain is a collection of data that is segregated into a list of blocks. Each block is cryptographically linked to the previous block in such a way that one cannot edit a block without editing all subsequent blocks. This cryptographic linkage is normally achieved by including a hash digest of the previous block (or a hash digest of some portion of the previous block) in the subsequent block.

A blockchain is stored by a group of blockchain nodes. The block generation protocol will vary depending on the blockchain protocol being used, but once a new block is generated, it is sent to one or more of the blockchain nodes. The recipients then confirm that the received block is valid, and if so, they distribute it to other blockchain nodes; invalid blocks are ignored. It may take some time for all blockchain nodes to receive the new block, but it is expected that eventually all blockchain nodes will agree on the contents of a given block in the chain. The method by which the blockchain nodes reach a consensus on the contents of a given block will vary depending on the blockchain protocol.

Consensus and mining. Various consensus and mining mechanisms exist [2,3]. For example, Bitcoin uses a *Proof-of-Work (PoW)* consensus and mining, where a node must demonstrate that it has performed some amount of work in the form of computing a hash with specific properties to add a block to the ledger. Another mechanism is *Proof-of-Stake (PoS)*, where each node is allowed to vote as to which blocks are added to the blockchain, according to the amount of currency the node owns. *Proof-of-Elapsed-Time (PoET)* requires each node to wait some time from a trusted piece of hardware, and the node with the shortest wait time generates the next block. *Round Robin (RR)* enables the participating nodes to take turns to create new blocks. Note that in RR, nodes that maintain the ledger are not anonymous. *Byzantine Fault Tolerance (BFT)* variations require each node to keep a list of nodes that they trust and perform a BFT-like algorithm to interact and determine the next block. The selection of the consensus and mining mechanisms may depend on the actual service applications of the IoT devices, and our protocol works with any consensus and mining mechanisms.

Multi-party computation (MPC). A Multi-Party Computation (MPC) protocol allows a group of servers to store secret data in a distributed fashion and perform computations on the distributed data without revealing the secret data to any individual server. Once the computation is complete, the servers can provide the output of the computation to one or more recipients.

During the execution of an MPC protocol, some of the servers may become corrupted to reveal secret data or to behave in a manner not specified by the protocol description.

MPC protocols are designed to be secure under the assumption that no more than a threshold number of servers are corrupted. The number of servers is denoted n and the corruption threshold is denoted as t. (This notation is common in MPC literature, and is used throughout this disclosure.) For example, an MPC protocol may state that it is secure so long as less than one third of the servers are corrupted. In this case, the protocol requires that $t < n/3$, so that if, for instance, n=7, then the protocol would require that t is no more than 2, meaning that no more than 2 out of the 7 servers are corrupted. The specific values of n and t for this invention will depend on the MPC protocol being used.

In this paper, MPC subprotocols are used to distribute private keys, and any MPC protocol from the literature will suffice (e.g., [4]), as long as it contains subprotocols that allows a dealer to distribute shares of a secret among a group of servers, and possibly deliver auxiliary information to the servers as well. Whether or not the auxiliary information is included will depend on which subprotocol from the literature is being used. A secret reconstruction protocol, which allows the servers to use their shares and possibly auxiliary information to reveal a secret to a recipient.

III. Problem Definition

Our goal is to design a security mechanism that provides privacy guarantees on the data collected by the IoT devices, and that enables the data owners to be in control which third parties can access what types of data in an intuitive manner. While some IoT devices have enough memory to store all the data that it collects, the majority of the IoT devices have limited memory. Hence, we are interested in a security solution that protects the data as it is transferred to and stored on an external server. In this scenario, the core challenge is to minimize the overhead (i.e., computation, communication, and storage) on the IoT devices, while providing the security on their data.

A. Attacker Model

Two types of attackers are relevant for our paper. First, an attacker may want to access the protected data being collected by IoT devices without proper access permissions. An example is stealing a key to access the remote controller access history to burglar the home when its owner is not present. Second, an attacker may want to modify or delete the protected data from IoT devices. For example, an attacker may attempt to modify a victim's heart beat record to jeopardize or threaten his life.

B. Desirable Properties

- **Privacy:** The data should only be accessible by the authorized entities that the data owner grants.
- **Integrity:** Once the data is transferred and stored on an external server, any modification or deletion should be detected.
- **Efficient forward and backward secrecy:** It is necessary to restrict (1) which data an entity can access, and (2) how long he/she can access it. Specifically, an entity, who has been granted to access certain data at time t, should be restricted from accessing the data before time t (forward secrecy). Similarly, an entity, who loses

access to the data at time t, should not be able to access the data after time t (backward secrecy). Furthermore, forward and backward secrecy should not incur additional computational, communication, and storage overhead on IoT devices.
- **Availability:** The data should be accessible as needed, without significant delays. Also, the data should be stored in a distributed manner to mitigate a single point of failure.

IV. Insights behind SSP

Some IoT devices may have access to the dedicated backend servers to store their data as part of specific services. Other IoT devices may require some fees if the owners desire to store their data on the dedicated servers, or not offer such services at all. Based on the accessibility of the service-specific backend servers, we introduce two variants of Self-Sovereign Privacy (SSP). We explain the insights behind SSP to provide integrity, availability, and self-sovereign privacy.

Integrity and availability guarantees. The blockchain has been a successful distributed data structure where the participating blockchain nodes maintain and synchronize the block contents across the network using a consensus and mining mechanism. SSP utilizes the distributed nature and the consensus mechanism to provide the consistent view of the committed data as long as the majority of the nodes (i.e., 50%) are honest to obey the protocol.

More specifically, once the IoT devices collect data, they commit to the blockchain the data, or its hash if the data can be stored on the service-specific backend servers, with encryption. With the correct commitments from the participating blockchain nodes, anyone, including the data owner, can always verify any illegitimate updates or data deletions from the blockchain.

Key management. IoT devices may be vulnerable to attackers that try to steal the keys to decrypt the data on the blockchain. Since IoT devices may be limited with its key management capabilities (e.g., lack of special hardware to securely store the keys), we propose that the keys are managed by MPC servers. In this case, unless an attacker compromises more than $1/3$ of the MPC servers, the keys remain secure and can be disclosed to those authorized by the data owners.

A. *Entities for SSP*

IoT devices. IoT devices are responsible for collecting the data, for example using their embedded sensors. Some IoT devices may have sufficient capabilities for the cryptographic key generation, in which case they perform the key generations. Note that an IoT device may collect multiple types of data (e.g., a surveillance camera can record both audio and video files), in which case the device may use different keys for different types of data. Then the IoT device can upload the data in the form of a block to the blockchain.[3]

(Option) Gateway devices. In the case of an IoT device that is incapable of generating cryptographic keys due to its low computational capabilities and storage capacity, it may rely on a gateway device that can be paired, such as a WiFi router at home or an external server offered by the service. These gateway devices will then generate the keys on behalf of the IoT devices and upload the data that is forwarded by the IoT devices. More specifically, IoT and gateway devices can setup a symmetric key to exchange the collected data, and the gateway device can use another key that it shares with MPC to encrypt and upload the data to the blockchain. In this paper, we describe the protocols using symmetric keys that requires minimal cryptographic overhead on IoT devices, but symmetric keys may require a special hardware on these gateway devices to store them securely. Public-key cryptography can be used instead if IoT devices can support the public-key operations.

(Option) Service-specific backend server. When the IoT devices are packaged with a service to store the data on a dedicated, service-specific backend server, the backend server is responsible for storing the data with encryption, and forwarding the hash of the data to the blockchain, such that anyone can validate the integrity of the stored data.

Blockchain and mining nodes. Blockchain (and mining) nodes are responsible to add the received data to the blockchain. More specifically, the blockchain may store the encrypted data, or the hash of the data if the service-specific backend server exists, and distribute them across participating blockchain nodes to maintain a consistent view of the data that have been forwarded by IoT devices or backend servers.

MPC servers. MPC servers are responsible to manage the keys to decrypt the data. To serve this task, MPC servers work with IoT devices or the gateway devices to receive the private keys associated with the encrypted data. To support the self-sovereign privacy, MPC servers coordinate with the data owner (or the IoT device that generates the data) to update the keys such that only authorized parties can access the data while ensuring the forward and backward secrecy. The MPC mechanism preserves its security guarantees given multiple servers that participate in the MPC protocol. Hence, we assume that multiple MPC servers exist that are managed by possibly mutually-exclusive entities, such as IoT manufacturers.

For the rest of this paper, we consider the worst-case scenario of IoT devices that are limited in computational, communication, and storage capabilities and hence delegate all data-handling tasks to the gateway devices.[4] For two variants

[3]We leave it up to the data owner to determine the size of the uploaded data based on the frequency/urgency of the desired data upload. For example, the owner can configure the device to collect the data to the maximum supported block size and instruct the device to upload it to the blockchain, or upload the data at certain frequency (e.g., at every 12am) even if the data size is less than the maximum block size.

[4]Otherwise, readers can consider IoT devices to conduct all tasks that the gateway devices perform.

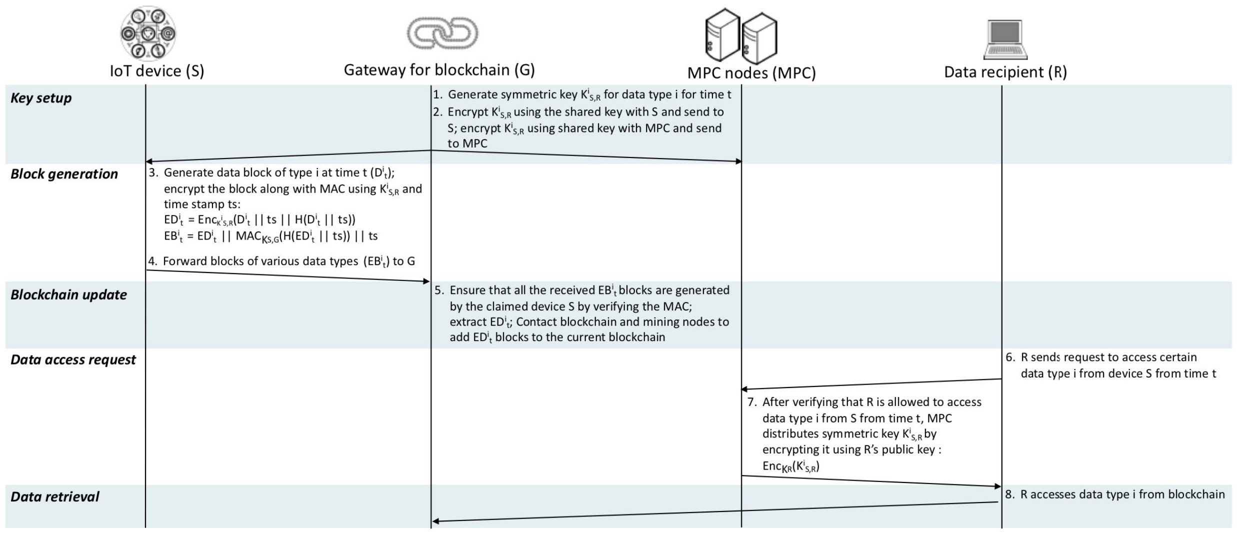
IoT device (S)
Gateway for blockchain (G)
MPC nodes (MPC)
Data recipient (R)
Key setup
1. Generate symmetric key $K^i_{S,R}$ for data type i for time t
2. Encrypt $K^i_{S,R}$ using the shared key with S and send to S; encrypt $K^i_{S,R}$ using shared key with MPC and send to MPC
Block generation
3. Generate data block of type i at time t (D^i_t); encrypt the block along with MAC using $K^i_{S,R}$ and time stamp ts:
$ED^i_t = Enc_{K^i_{S,R}}(D^i_t \| ts \| H(D^i_t \| ts))$
$EB^i_t = ED^i_t \| MAC_{KS,G}(H(ED^i_t \| ts)) \| ts$
4. Forward blocks of various data types (EB^i_t) to G
Blockchain update
5. Ensure that all the received EB^i_t blocks are generated by the claimed device S by verifying the MAC; extract ED^i_t; Contact blockchain and mining nodes to add ED^i_t blocks to the current blockchain
Data access request
6. R sends request to access certain data type i from device S from time t
7. After verifying that R is allowed to access data type i from S from time t, MPC distributes symmetric key $K^i_{S,R}$ by encrypting it using R's public key : $Enc_{KR}(K^i_{S,R})$
Data retrieval
8. R accesses data type i from blockchain

Fig. 2. SSP protocol overview.

of the SSP mechanisms, we vary the existence of the service-specific backend server.

V. Protocol Description

We now describe two variants of SSP: SSP and SSP-light.

A. *Self-Sovereign Privacy (SSP)*

SSP assumes that the IoT-device owner has not signed up for a service that provides a dedicated, service-specific server that can securely store the data collected by the IoT device. This scenario involves the following entities: An IoT device (S) that collects possibly multiple types of its owner's data, a gateway device (G) that uploads the encrypted data to the blockchain, MPC servers (MPC) that manage and distribute the secret keys for data decryption, and a data recipient (R) that desires to access the owner's specific type of data. We assume that S and G have pre-established shared symmetric key $K_{S,G}$.

Key setup. This initial stage involves G to generate a symmetric key $K^i_{S,R}$ for data type i for time t. Then G shares the symmetric key with MPC and S. Note that whenever the data owner desires to update the data access privilege for any recipients for this data type (i.e., revoke the current recipient's access permission, or add another recipient to access the same data type), the key setup process repeats to generate a new symmetric key. In Section V-C, we present how such key updates can be efficiently performed without requiring the data recipients to request MPC servers.

Block generation. Upon receiving symmetric key $K^i_{S,R}$, S collects the data D^i_t of type i at time t, and generates a block by encrypting it along with a message authentication code (MAC) using $K^i_{S,R}$ as follows:

$$ED^i_t = Enc_{K^i_{S,R}}(D^i_t \| ts \| H(D^i_t \| ts)); \qquad (1)$$

$$EB^i_t = ED^i_t \| MAC_{K_{S,G}}(H(ED^i_t \| ts)) \| ts, \qquad (2)$$

where ts and H stand for a timestamp and a hash operation, respectively. Note that the MAC is to help G authenticate that the received data is indeed generated by G at time ts. Then S forwards the block EB^i_t to G.

Note that we use MAC to minimize the computational overhead on the IoT devices. If IoT devices are capable of computing public-private encryption, MAC can be replaced by a signature on the data as follows:

$$EB^i_t = ED^i_t \| Sign_{K_S^{-1}}(H(ED^i_t) \| ts) \| ts, \qquad (3)$$

where $Sign$ and K_S^{-1} stand for a signature and S's private key, respectively. Using the signature may provide an additional security benefit for those that download the data to ensure that the data was indeed generated by the claimed IoT device S.

Blockchain update. Upon receiving the the encrypted block (EB^i_t), G ensures that the block is generated by the claimed S by verifying the MAC. When the MAC verification succeeds, G simply extracts the encrypted data from EB^i_t and add it to the current blockchain by contacting blockchain and mining nodes.[5] At this point, the encrypted data collected by S is available for an authorized third party to access it.

Data access request. MPC servers manage the keys to decrypt the data on the blockchain, and releases them according to the access permission that the data owner defines. To access the data generated by the IoT device S, the data recipient R must contact the data owner. More specifically, R must specify the data type, and the duration of the access (e.g., from time t, from t until t_1, etc.). In return, the data owner updates the access permission to add R for data type i for the time duration

[5]Depending on the types of the blockchain system, G may receive a confirmation that the requested data has been appended to the blockchain.

as agreed on, and informs the MPC servers with the updated the access permission list.

R sends a request to MPC servers to access D_i^t generated by S, and MPC distributes the symmetric key, encrypted using R's public key K_R such that R can decrypt the data:

$$Enc_{K_R}(K^i_{S,R}). \tag{4}$$

Data retrieval. R can now retrieve the data block from the blockchain, and use the received key $K^i_{S,R}$ to access the data generated by S at time t. There are cases where keys need to be updated for various reasons. For example, R's public-private key pair can be compromised, in which case K^i_{SD} can no longer stay secure. Other examples include addition or deletion of the data recipient to provide forward and backward secrecy. In Section V-C, we describe how SSP handles key updates to protect the privacy of the data stored in the blockchain.

B. Self-Sovereign Privacy-Light (SSP-light)

When IoT devices can utilize the service-specific backend server to store the data, the blockchain can be used to protect the integrity of the stored data. Similar to above, we use a scenario that involves the following entities: An IoT device (S) that collects possibly multiple types of its owner's data, a backend server (B) that stores the data with encryption, MPC servers (MPC) that manage and distribute the secret keys for data decryption, and a data recipient (R) that desires to access the owner's specific type of data. We assume that S and B have pre-established shared symmetric key $K_{S,B}$.

The majority of the steps are similar to SSP, except that (1) the backend server B takes the role of the gateway device G and that (2) the blockchain stores the hash of the data instead of the encrypted data.

Key setup. S and MPC can establish a shared symmetric key $K^i_{S,R}$ for data type i for time t. Note that whenever the data owner desires to update the data access privilege for any recipients for this data type (i.e., revoke the current recipient's access permission, or add another recipient to access the same data type), the key setup process repeats to generate a new symmetric key. In Section V-C, we present how such key updates can be efficiently performed without requiring the data recipients to request MPC servers.

Block generation. Upon receiving symmetric key $K^i_{S,R}$, S collects the data D^i_t of type i at time t, and generates a block by encrypting it along with a message authentication code (MAC) using $K^i_{S,R}$ as follows:

$$ED^i_t = Enc_{K^i_{S,R}}(D^i_t, ts); \tag{5}$$

$$EB^i_t = ED^i_t || MAC_{K_{S,B}}(H(ED^i_t||ts))||H(D^i_t||S||ts)||ts, \tag{6}$$

where ts and H stand for a timestamp and a hash operation, respectively. Note that the MAC is to help B authenticate that the received data is indeed generated by G at time ts. Then S forwards the block EB^i_t to B.

As mentioned in Section V-A, if IoT devices are capable of computing public-private encryption, MAC can be replaced by a signature on the data.

Blockchain update. Upon receiving the the encrypted block (EB^i_t), B ensures that the block is generated by the claimed S by verifying the MAC (or the signature). When the MAC/signature verification succeeds, B simply extracts the encrypted data ED^i_t from EB^i_t and stores it on its own server. B then contacts blockchain and mining nodes to add the hash of the data $H(D^i_t)$ generated by the IoT device S with the data type i at time t to the blockchain.

Data access request. This step is the same as presented in Section V-A. MPC servers manage the keys to decrypt the data on the blockchain, and releases them according to the access permission that the data owner defines. To access the data generated by the IoT device S, the data recipient R must contact the data owner. More specifically, R must specify the data type, and the duration of the access (e.g., from time t, from t until t_1, etc.). In return, the data owner updates the access permission to add R for data type i for the time duration as agreed on, and informs the MPC servers with the updated the access permission list.

R sends a request to MPC servers to access D^t_i generated by S, and MPC distributes the symmetric key, encrypted using R's public key K_R such that R can decrypt the data:

$$Enc_{K_R}(K^i_{S,R}||ts). \tag{7}$$

Sending ts ensures that R uses the correct key to access the data that was generated at time ts.

Data retrieval. R now contacts the backend server B, which in return forwards the data encrypted using R's public key:

$$Enc_{K_R}(ED^i_t||ts). \tag{8}$$

After retrieving data, R can ensure the integrity of the data (i.e., backend server has not manipulated the data) by checking the hash of the data that is stored on the blockchain.

C. Efficient Key Updates

It is crucial to limit the duration of the data access per recipient to prevent unauthorized entities from accessing the data with forward and backward secrecy, as mentioned in Section III-B. To provide forward and backward secrecy, we propose that different keys are used to encrypt different blocks, where the succeeding key can be derived from the previous key using a hash function.

More specifically, after encrypting a block of data type i using $K^i_{S,R}$, S simply hashes the key such that the next block is encrypted using $H(K^i_{S,R})$. In this manner, forward secrecy can be guaranteed, since R cannot decrypt S's data that was generated before time t, due to the one-wayness of the hash operation. Furthermore, R can continue retrieving S's data after time t without contacting MPC all the time.

R may lose access to the current key (e.g., $H^n(K^i_{S,R})$ where the hash has been applied n times on the first key that it received from MPC at time t). For example, the access grant period agreed with the data owner is over, or R loses the current key. In such cases, the encryption key for the data must be updated to provide backward secrecy, such that R or an unauthorized entity in possession of the key cannot access the data from the current time (i.e., $t+n$). SSP and SSP-light then updates the key to a new value (i.e., $K^{i}_{S,\neg R}$) and continues applying the hash operation until another recipient loses access. When the key is updated, MPC servers may contact other legitimate recipients to inform the updated key (i.e., push keys). Alternatively, legitimate recipients always validate the correctness of the downloaded data, by computing the hash on the decrypted data and timestamp ts and ensure that it matches with the enclosed hash value inside the block (Equation 1). In case of a mismatch, the legitimate recipients can contact MPC servers to receive the updated key (i.e., pull keys).

VI. Security Analysis

In this section, we analyze how SSP and SSP-light mitigate potential attacks.

A. Security Analysis of SSP

Key compromise on IoT devices (S). When an attacker compromises S's key $K^{i}_{S,R}$ at time t, the attacker may be able to retrieve the data from time t on the blockchain until S detects its key compromise and updates the key. Although the attacker can eavesdrop the data, the attacker cannot manipulate the data, which requires compromising $K_{S,G}$ or K^{-1}_{S} to update the MAC or the signature and to convince G. On the other hand, compromising S invalidates any data that it collects anyways. SSP ensures that the attacker does not retrieve any data before time t, thanks to its forward secrecy guarantee.

Key compromise on gateway devices (G). An attacker does not gain any information about S's protected data by compromising G's shared key with S ($K_{S,G}$). Compromising both G and S, on the other hand allows the attacker to not only gain access to the data, but also modify the data.

Key compromise on data recipient (R). When an attacker successfully compromises the key that R has previously received from MPC ($K^{i}_{S,R}$) at time t, the attacker can gain access to the block contents that have been accessible by R since time t until either R notifies its key compromise to MPC/S, or R's access permission time expires. Since old blocks were encrypting using $K^{i}_{S,R}$ and its hashes, these blocks can be exposed to the attacker without a proper access permission. Consequently, once R finishes retrieving the information (e.g., at time $t+n$), R should ensure to store the next key (e.g., at time $t+n+1$) in a tamper-resistant security module, and never store the previous keys (e.g., from time t until $t+n$).

Key compromise on MPC servers (MPC). Unless an attacker successfully compromises the majority of the MPC servers, the attacker cannot acquire the keys to decrypt the data on the blockchain.

TABLE I
Overhead measurement on IoT devices.

IoT device overhead	Self-Sovereign Privacy
Executable size	21KB
Computation	12ms per MB of data
Communication	256b per key update of a data type
Storage	256(n+1)b for n data types

B. Security Analysis of SSP-Light

The security analysis of SSP-light is similar to SSP, except that the backend server (B) replaces the gateway devices (G).

Key compromise on IoT devices (S). Similar to above in Section VI-A, an attacker that successfully compromises S's key $K^{i}_{S,R}$ at time t can retrieve the data from time t if the attacker can observe the communication between S and the backend server B. Although the attacker can eavesdrop the data, the attacker cannot manipulate the data, which requires compromising $K_{S,B}$ or K^{-1}_{S} to update the MAC or the signature and to convince G. On the other hand, compromising S invalidates any data that it collects anyways. With the forward secrecy guarantee, SSP-light also ensures that the attacker does not retrieve any data before time t.

Key compromise on backend server (B). An attacker does not gain any information about S's protected data by compromising B's shared key with S ($K_{S,B}$). Compromising both B and S, on the other hand allows the attacker to not only gain access to the data, but also modify the data.

Key compromise on data recipient (R). An attacker may successfully compromises R and acquire $K^{i}_{S,R}$. However, compromising $K^{i}_{S,R}$ alone does not enable the attacker to retrieve the data that B forwards. To successfully retrieve the data without the access grant, the attacker must compromise R's private key in addition.

Key compromise on MPC servers (MPC). Similar to above, an attacker must successfully compromise the majority of the MPC servers in order to decrypt the protected data on the blockchain.

VII. Implementation Results

To illustrate the benefits of our mechanism, we implemented both the client and server codes using C. More specifically, we implemented the IoT-side blockchain code on Raspberry Pi (quadcore, ARM Cortex-A53, 1.2GHz). We selected SHA224 for hash operations and AES-256 for symmetric-key encryption.

As shown in Figure 3, SSP incurs small time overhead on the resource-limited IoT devices. More specifically, SSP incurs .1ms up to 10s to encrypt and compute a MAC of a data size up to 1 GB on the Raspberry Pi. Our executable size is 21KB and takes 12ms per MB of data to process the data on the IoT side. In terms of the communication overhead, SSP incurs 256bits per key update per data type, and storage overhead is $256(n+1)$ bits for n data types. Table I summarizes the space, computational, and communication overhead on IoT devices.

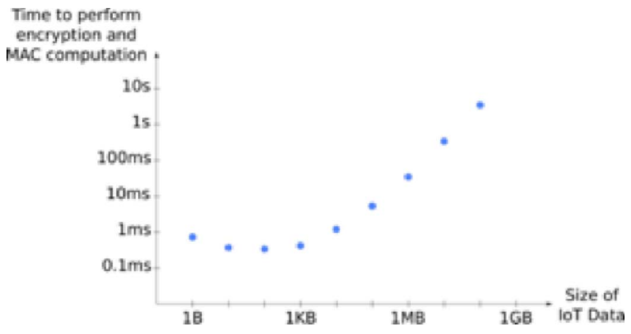

Fig. 3. SSP's time overhead on IoT devices.

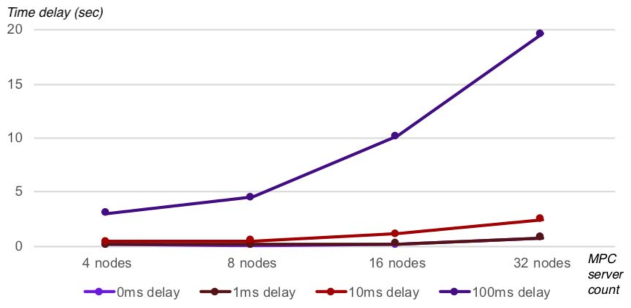

Fig. 4. SSP's time delay for key recovery from MPC servers.

We also implemented the server-side MPC code in C, and measured the overhead using an Intel Xeon 3.50GHz server. Since the data recipient R retrieves a key that is being shared by MPC servers, we measured the time delay for each key being recovered with 4, 8, 16, and 32 MPC servers. To emulate the transmission from MPC servers to a data recipient, we added time delays of 0, 1, 10, and 100ms. As shown in Figure 4, the key recovery process is quite fast, taking from 0.16sec with 4 MPC servers to 0.75sec with 32 MPC nodes, given 1ms delay. Even with 10ms delay, the recovery takes 0.41sec to 2.46sec with 4 and 32 MPC nodes, respectively. When the transmission delay increases to 100ms, it dominates the computation overhead from MPC servers.

VIII. Discussion

Many design decisions remain open for SSP or SSP-light. In this section, we dicuss the benefits and cost of the design options.

Incentives for blockchain nodes. Blockchain nodes mine and maintain either the protected data collected by IoT devices for SSP, or the hash of the protected data for SSP-light. Consequently, the data owner, IoT device manufacturers, or the service providers can provide incentives to the blockchain mining and consensus nodes for their efforts to maintain the blockchain.

Access pattern tracking. Since all the data recipients must acquire the keys to access the data, IoT device owners can acquire the tracking analysis from MPC nodes (e.g., accessed entities, frequency of their access, etc.). Such a tracking capability is important for the data owners to review and update their access control policies when they identify data misusage.

Public vs. private blockchain. If the blockchain is public, anyone can contribute the data to, maintain, or download the data from the blockchain. While SSP and SSP-light provides security and privacy guarantees, care must be taken for the key compromises as described in Section VI, especially on the recipient side. Alternatively, the blockchain can be private if the data owner or the service provider desires to limit who can access the encrypted data.

IX. Related Work

We review the most relevant work that aims at protecting sensitive information using blockchain or MPC. Enigma is a decentralized computation platform using a highly optimized version of secure MPC to secret-share sensitive data and store it across cloud servers using distributed hashtable [5]. In Enigma, blockchain serves as a tamper-proof log of events, including access records and the data locations. Both Enigma and SSP provides privacy protection on the data using MPC and blockchain in different ways. Lightweight Scalable Blockchain (LSB) has been proposed for IoTs, utilizing external nodes to manage intensive blockchain operations [6,7]. However, LSB does not provide privacy protection on the data.

Several researchers have proposed using smart contracts to provide privacy protections using access control mechanisms. Hawk is a proposed framework for building privacy-preserving smart contracts [8], and FairAccess proposes an access control authorization mechanism by transferring access tokens using transactions [9]. Zhang et al. proposes a distributed access control for IoT [10]. Unlike these approaches, SSP and SSP-light support data owners to define access controls while preserving the data privacy without smart contracts.

X. Conclusions

This paper introduces mechanisms that utilize blockchain and MPC to achieve data integrity and privacy for IoT devices in a lightweight manner. Our protocols have been shown to be fast without incurring much overhead on resource-constrained IoT devices and MPC servers. Our future work includes applying SSP and SSP-light to a variety of use cases, including supply chain management, sensor network data privacy management, and identity management. We hope that our paper supports the usage of blockchain and MPC towards bringing self-sovereign privacy for IoT devices.

References

[1] R. Contu, P. Middleton, S. Alaybeyi, and B. Pace, "Forecast: IoT Security, Worldwide, 2018," Gartner, Tech. Rep., 2018.

[2] A. Baliga, "Understanding Blockchain Consensus Models," Persistent Systems Ltd, https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf, Tech. Rep., 2017.

[3] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," in *In Proceedings of the 2015 IEEE Symposium on Security and Privacy*, 2015, pp. 104–121.

[4] I. Damgard and J. B. Nielsen, “Scalable and unconditionally secure multiparty computation,” *Advances in Cryptology - CRYPTO 2007*, vol. 4622, 2007.

[5] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: decentralized computation platform with guaranteed privacy, 2015,” *URL: https://enigma.co/enigma_full.pdf*.

[6] A. Dorri, S. S. Kanhere, and R. Jurdak, “Towards an Optimized Blokchain for IoT,” in *Proceedings of the IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2017, pp. 173–178.

[7] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Blockchain for IoT security and privacy: The case study of a smart home,” in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017.

[8] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” in *Proceedings of the IEEE Symposium on Security and Privacy*, 2016.

[9] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, “FairAccess: a new Blockchain-based access control framework for the Internet of Things,” *Security and Communication Networks*, vol. 9, pp. 5943–5964, 2017.

[10] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, “Smart Contract-Based Access Control for the Internet of Things,” arXiv: 1802.04410v1, 2018.