A Resource-Efficient Smart Contract for Privacy Preserving Smart Home Systems

Nazmus Saquib, Fatih Bakir, Chandra Krintz, Rich Wolski
Department of Computer Science
University of California, Santa Barbara
{nazmus, bakir, ckrintz, rich}@cs.ucsb.edu

***Abstract*—Due to the proliferation of IoT and the popularity of smart contracts mediated by blockchain, smart home systems have become capable of providing privacy and security to their occupants. In blockchain-based home automation systems, business logic is handled by smart contracts securely. However, a blockchain-based solution is inherently resource-intensive, making it unsuitable for resource-constrained IoT devices. Moreover, time-sensitive actions are complex to perform in a blockchain-based solution due to the time required to mine a block. In this work, we propose a blockchain-independent smart contract infrastructure suitable for resource-constrained IoT devices. Our proposed method is also capable of executing time-sensitive business logic. As an example of an end-to-end application, we describe a smart camera system using our proposed method, compare this system with an existing blockchain-based solution, and present an empirical evaluation of their performance.**

***Index Terms*—IoT, smart contract, smart home, ethereum, blockchain**

I. Introduction

In recent years, the advancements and pervasive application of the Internet of Things (IoT) have influenced home automation, resulting in the increased popularity of smart home systems. In a smart home, low-cost, resource-constrained IoT devices control domestic appliances depending on the change in a physical property (e.g., temperature, light, etc.) or the occurrence of an event (e.g., opening a door). The massive amount of data collected by these IoT devices can be personal and sensitive; and are often transmitted over an insecure network to untrusted service providers for further analysis [1]–[3]. This raises concerns about data security and privacy, as data can be used and altered by service providers, such as the cloud, where it is stored. Therefore, a decentralized system where the end-user and any untrusted party can share immutable sensitive information is desirable.

A blockchain, which is an emergent peer-to-peer, immutable digital ledger technology is such a system. Unsurprisingly, the application of blockchain in smart home has garnered interest from the research community for addressing security and privacy concerns recently [4]–[11]. Some blockchains such as Ethereum [12] support smart contracts – executable programs stored on a blockchain that run when some predetermined condition is met. This enables smart home systems to record data on the blockchain and embed business logic in the blockchain that is contingent upon that data.

Although blockchain can be an effective tool to ensure data privacy and security, its application in an IoT environment has its own challenges [13]. First, most blockchain algorithms require significant computational power and as such are not suitable for *direct* use on resource-constrained IoT devices. Typically, such devices must communicate with high-end (resource-rich) devices that essentially act as proxies to leverage the features provided by a blockchain. However, this proxy architecture introduces new challenges in terms of reliable network connectivity to the proxy and the complexity associated with securing the channel between the devices and their proxies. Reliable and fast network connecting the devices to their blockchain contact points can be power intensive to implement. Further, a separate protocol for securing the connection to the blockchain proxy must be correctly integrated with the blockchain protocols, further adding to the heterogeneity and interoperability complexities that "simple" IoT devices must support using resource-constrained architectures. Third, time-sensitive actions are difficult to perform in a blockchain due to the time required to mine a block using blockchain algorithms. For example, at the time of writing this paper, the public Ethereum mainnet on average mines a block approximately every 13 seconds [14], meaning we can expect this much delay on average for a transaction to be recorded in the ledger. Historical values higher than 30 seconds have also been observed in the past. Other blockchains such as bitcoin which mines a block every 10 minutes [15] can be more time-consuming. Further, typically there is no guarantee that a transaction will be included in the next mined block, so the wait time can be much greater than the average time to mine a block.

As a result, many blockchain-based smart home systems use a private blockchain that has a faster mining rate to minimize this delay [5], [6], [8], [11]. However, this optimization introduces a new trust relationship between the devices and the entity that operates the private blockchain that public blockchains do not require. That is, in a private blockchain, whoever is maintaining the blockchain has ultimate authority over the data. Such data sharing is undesirable in a smart home system, especially in sharing economy in which a property owner rents rooms/apartments to the tenants, or the "traditional" economy where real estate agents require temporary access to a property that is for sale, thus sharing control over the home's intelligent electronics (e.g.

surveillance cameras, electronic door locks, alarm systems, etc.). These challenges ultimately accrue to the inability of resource-constrained devices to directly implement the security guarantees and tamper resistance offered by blockchains and smart contracts.

In this work, we describe the design and implementation of a blockchain-independent, smart contract that is suitable for direct implementation on resource-constrained IoT devices. Our proposed design is independent of any particular blockchain technology and does not share the aforementioned limitations. However, it is still able to provide shared security and privacy desired in a smart home system, with the possibility of its application in other fields as well. The proposed method is based on access control using *capability tokens* [16]. Unlike traditional tokens, our token system can contain bytecodes that are executable in virtual machines that are lightweight enough for implementation on resource-restricted devices and secure enough to implement smart contracts. This capability-based approach is device local, resulting in a more time-efficient system than its blockchain-based counterpart. The key to our approach is that any user in possession of a token originally generated by some device can create *derivations* of it, which are tokens with constrained privileges relative to that of the current token that the user possesses. Additionally, any device can cryptographically verify a token that it generated originally along with *any* derivation of this token, transitive or immediate. To implement this verification feature, a token carries a "chain" of derivations (each modifying the one before it) and the device traverses a chain of derivations and compares computed and stored hash values. This is similar in concept to the way in which blockchains implement verification: the protocols compare the stored and computed hash values of the transactions in the latest block and iteratively perform this operation up to the first block by following reference to the parent block. We present the details of our proposed approach in Section IV.

As an example end-to-end smart home application, we present a detailed implementation of a smart camera in sharing economy using our proposed method. We then compare our system against an existing blockchain-based solution [4] and present an empirical evaluation of the two systems. We find that our system can operate in resource-constrained devices, be used in time-sensitive operations, and has *5 orders of magnitude better* user perceived latency than the blockchain-based solution, and is thus suitable for on-device implementation.

II. Background

In this section, we give a brief overview of blockchain, types of blockchain, and smart contracts. We also explain how each of these is related to smart home systems.

A. Blockchain

A blockchain is a distributed, peer-to-peer, immutable digital ledger consisting of a chain of blocks. Each block contains one or more transactions along with a hash of these transactions. These blocks are *mined* (i.e. added to the chain) by solving cryptographically hard problems requiring significant computational power. High-end devices that are used to solve these problems are called *miners*. There are generally multiple miners in a blockchain forming a peer-to-peer network.

The ordering of blocks in a blockchain is achieved using a consensus algorithm. Each block contains a hash of the previous one, which means the entire chain can be traversed and validated starting from any block up to the first one (called *genesis*). The cryptographic algorithms are designed in such a way that for an adversarial entity to tamper with a block and still get validated would require an impractical amount of computational power. A blockchain provides multiple features that are desirable to ensure privacy and security in a smart home system:

- *Decentralization*: As a blockchain is a peer-to-peer system, users need not rely on an untrusted third party to store their data. Moreover, the same system can be used to allow tenants to maintain their data separately from that of homeowners.
- *Immutability*: A blockchain can ensure the integrity of transactions through its immutable ledger. Transactions can be performed between a homeowner and a tenant using the blockchain, which cannot be disputed.
- *Transparency*: A transaction mined in one node is propagated to multiple nodes in the blockchain. Moreover, these transactions are validated by the receiving nodes. This adds to the confidence of the involved parties regarding the correctness of the transactions.

B. Types of Blockchain

There are primarily two types of blockchain: (i) public and (ii) private. Any user can join a public blockchain network and add and verify data. On the other hand, only certain "permissioned" entities can take part in a private blockchain network. As a result, the consensus algorithms and validation process used in a public blockchain tend to be more resource-intensive than that of a private blockchain. Most specifically, private blockchains typically implement faster mining and validation rates than their public counterparts [17]. However, a private blockchain requires trust among the participants. Moreover, it is considered to be less secure than its public counterpart, as in a public network the number of nodes involved is high, resulting in a low probability of a majority attack being successful. Therefore, a public blockchain provides better security and privacy features compared to a private one [18]. Although the use of private blockchains in smart home systems has been extensively researched in the literature [5], [6], [8], [11], relatively few focus on the use of public blockchains [4], [19].

C. Smart Contract

A smart contract is an executable code that is triggered when predefined conditions are met. Despite smart contracts not being new to the research community [20], they have

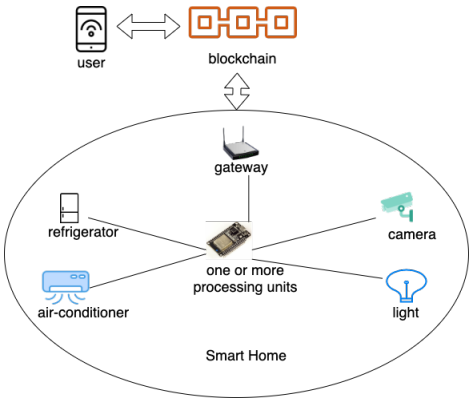

Fig. 1. A high-level architecture of existing blockchain-based smart home systems. Domestic appliances are controlled by one or more low-cost, resource-constrained processing units. Processing units can connect to the blockchain network using light clients through a gateway or proxy. Users can interact with the blockchain network to read/record different parameters required by smart-home applications, e.g., threshold values of temperature, light intensity, etc. Processing units can receive directives from the blockchain, e.g., whether to turn a light on/off, etc.

gained renewed attention after blockchains supporting them have been introduced — Ethereum [1] being the most significant. Functions in a smart contract are executed when a transaction calling that function is successfully added to a mined block. In this way blockchains allow business logic to be executed based on data stored securely in the blockchain.

III. Blockchain-Based Smart Home Systems

The traditional smart home system consists of one or more processing units that control home appliances [2]. These processing units are often low-cost and resource-constrained, e.g., microcontrollers or single-board computers. If there are multiple processing units, they usually communicate wirelessly. The processing units are connected to the Internet through a gateway. In addition, systems that use a private blockchain often include high-end devices acting as miners within the home network [5], [6], [8], [11]. The homeowner deploys smart contracts in the blockchain through a blockchain transaction and receives the address of the smart contract.

The basic architecture of a blockchain-based smart home system shown in Figure 1 has been adopted in multiple previous efforts. In [5], the authors use a single machine private Ethereum blockchain to implement an air-conditioning system. In this work, the authors use a smart contract to store a threshold temperature value and do not provide any experimental results. In [6], the authors use a private Ethereum blockchain with two miners to create an alert system that lights LEDs if the temperature/humidity rises above a threshold. Their results reveal that setting the threshold values takes 18.55 seconds on average. In [8], the authors create a private Ethereum blockchain with two machines and use the smart contract to store threshold values like the previous works. Although the authors do not present the execution time,

[1]https://ethereum.org/en/

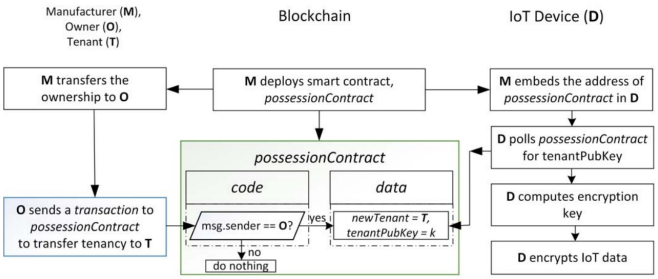

Fig. 2. Role of different users in the blockchain-based smart camera system [4].

they compare CPU usage between two well-known consensus algorithms. In [21], the authors propose a blockchain-enabled, capability-based access control that uses smart contracts to convey rights from client to service provider. The authors create a private Ethereum blockchain with six high-end miners and report that a client observes a delay of 243 milliseconds on average before receiving a response from the server, including the time required by the server to read from the smart contract. However, they do not report the time required to store token data. In [4], the authors propose a privacy-preserving smart camera system for sharing economy, such as Airbnb [2]. This work uses a public blockchain (to obtain greater security) and embeds business logic in the smart contract (e.g., checking conditions to transfer tenancy) instead of using it simply for storage. However, the authors do not provide any experimental evaluation of their proposed system. Our work presents a detailed explanation and an implementation of the system described in [4] as well as a comparative implementation using our approach (viz. Section V).

A. Smart Camera System

We consider a scenario where a homeowner rents a room to a tenant. The room has a camera that encrypts data while streaming video. Ideally, only the owner should be able to decrypt this data when the room is unrented, but a tenant should be able to take over this exclusive decryption capability for the duration of the rental. In [4], the authors provide a blockchain-based solution to this problem using smart contracts as described next.

The manufacturer of the camera records the address of the homeowner (every user account and smart contract in a blockchain has an address) within a smart contract (*PossessionContract*) and deploys it to a public blockchain. This contract is responsible for tenancy transfers and tenancy polling. The manufacturer embeds the address of the smart contract in the camera, along with the private key of the camera. The public key of the camera is recorded in the smart contract during deployment.

PossessionContract consists primarily of two functions: *transferTenancy* and *pollTenancy*. *transferTenancy* is used to update tenant's information such as the public key of the tenant, tenancy period, cost, etc. This function performs a

[2]https://www.airbnb.com

check at the beginning to make sure the entity calling the function is indeed the owner. A malicious actor (including a malicious owner) may call the function at any time during the valid tenancy period, and this function will return without affecting any value, thus respecting the original agreement between the owner and the tenant.

pollTenancy is used by the camera to find the public key of the current tenant. A camera polls the tenant's public key on a daily basis according to the proposed method in [4]. As both the camera and the tenant know each other's public key, they can establish a symmetric key using the Diffie-Hellman protocol. It is at this moment that the tenancy (with respect to the camera) begins as the camera encrypts its video stream data using this key which only the tenant can decrypt. Once the camera detects a tenancy change through a call to this function, a new key is established. Figure 2 shows the interaction among the different users and the functions of *PossessionContract* as proposed in [4].

The authors in [4] suggest the use of Ethereum as a public blockchain for this system. We highlight a few caveats of using the public blockchain that we explored while investigating this approach. First, functions that update or store data (data stored in a smart contract comprise its *state*) in the smart contract can be executed only as a result of a blockchain transaction getting mined [22]. As described in Section I, a block in Ethereum public network is mined every 13 seconds on average but practically, this delay can be much longer. This delay might prevent even a well-intended execution of this function. For example, consider a scenario where an owner sets the wrong information first (cost, e.g.) and then attempts to do it correctly just before the tenancy begins by calling the *transferTenancy* function. As the transaction containing this function call might get mined after the tenancy period starts, the function will return without making any change.

Second, Ether (ETH) is the currency of Ethereum, which can be bought using real money or mined by solving cryptographically hard problems. Every transaction costs some amount of ETH. Therefore, executing the *transferTenancy* function requires some amount of money, however small it might be. Note that the *pollTenancy* function does not cost any ETH, as it is reading a value rather than updating the state [23]. Functions that only read the state are known as *view* functions.

Third, in the version of the application described in [4], a smart contract function can not run on its own – it must be called explicitly (through a transaction if it updates state). Therefore, the tenancy will not revert automatically back to the owner after an agreed-upon tenancy period is over. Moreover, the owner has to make sure he/she transfers tenancy to a new tenant before the camera performs its daily poll but after the end of the old tenancy.

IV. Capability-Based Smart Home Systems

In this section, we describe an alternative approach to building smart home applications where access control is enforced through very computationally efficient capabilities. Our goal is to achieve feature parity with blockchain-based systems while reducing power consumption and operation latency sufficiently to permit direct on-device implementation. To enable this, we have designed and implemented a distributed capability framework for heterogeneous distributed systems like those used in IoT and smart homes. The complete approach, called CAPLETS is described comprehensively in [24] and overviewed here.

A *capability* is a communicable, unforgeable token of authority. The holder of a token is entitled to the privileges held in that token. While capabilities have many potential implementations, we are interested in network capabilities, which allow the tokens to be passed around freely over a network, as opposed to local capabilities such as those found in [25]. While local capabilities are protected from tampering and forging by kernel-user space separation, network capabilities are protected through cryptographic means, often in the form of a digital signature, signed by the origin of the token. For example, the tokens defined in [26] are simply JSON objects with a signature field. The contents are application-defined, but when the server receives a token, it can ensure that it is a token it generated before and the token has not been tampered in any way. The best analogy is they work like a truly secure concert ticket. Clients receive/buy a token at some point, and at a later point, they present it as proof that they are entitled to perform a request/enter the venue. Unlike physical tickets, it is impossible to forge a new token or tamper with it to gain access to a more expensive area.

Some implementations of network capabilities (e.g. [26]) do not allow any modification to the token by clients, while others [27], [28] allow controlled reduction of privileges in varying levels of expressiveness. The tokens defined by Macaroons [28] allow attaching richer constraints (e.g. beyond simple downgrades from read/write to read) to tokens. This *derivation* operation occurs without the knowledge or participation of the server, which forms the basis of the distributed authorization framework capabilities enable.

Alongside the use of capabilities for authorization, the most important difference from the blockchain-based system is that this version is completely device local. For instance, for the smart camera application, the entire state needed for operation, the stream encryption key, the tenant public key, etc. are maintained by the camera itself thus removing the costs of external activities, mainly blockchain mining, from the end-to-end system.

We construct a CAPLETS application as a group of RPC services running on a device. An application-dependent set of *capabilities* carried within tokens grant access to these services. Another set of application-dependent *constraints* limit the use of the capabilities in different ways. An example of a capability is the privilege to call a specific function of a service. An example of a constraint is limiting this function call to be made only from a specific network endpoint. Unlike Access Control Lists, Role-Based Access Control [29], or Attribute-Based Access Control [30] models, this information is not stored on the device as part of a database, but rather

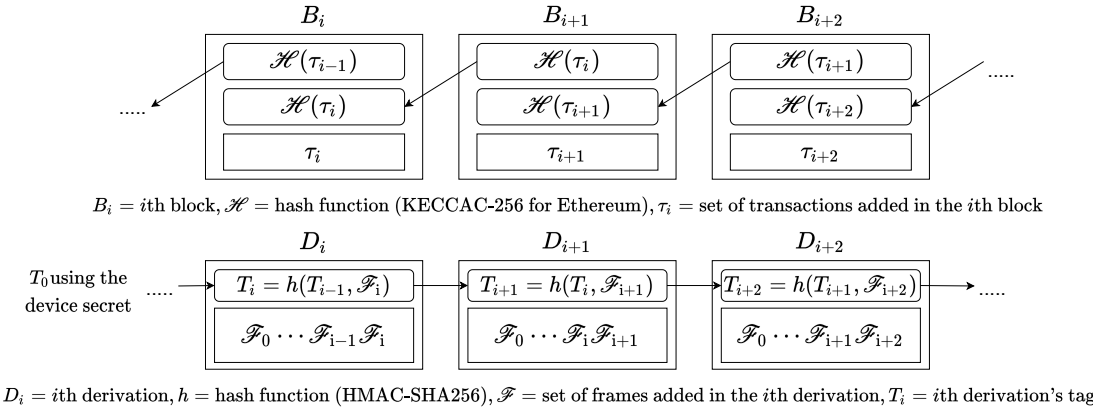

Fig. 3. Chain structures of blockchain and CAPLETS.

carried in cryptographically secure tokens and stored by the clients themselves. Its cryptographic construction prevents malicious clients from forging tokens.

Constraints of CAPLETS are implemented through programs for an application-specific byte code virtual machine or native code. This allows for absolutely flexible policy implementations, similar to smart contracts. Unlike smart contracts, the programs are efficient enough to be executed securely by resource-restricted devices.

Capabilities and constraints in CAPLETS are strongly typed objects. They have agreed upon structures and they maintain type information across the network. They can carry arbitrary information for use in authorization. CAPLETS also defines a mechanism to encode RPC invocations as part of its capability encoding.

We provide a distributed and secure method for sharing privileges in CAPLETS, called a *derivation*. Capabilities and constraints are carried in blocks called *frames* to support derivations. A client can append a new frame to an existing token while maintaining verifiable cryptographic proof that it indeed held the token to which it is appending the new frame. The capabilities in a frame can only be reduced and constraints can only be increased, so it is impossible to gain new privileges through derivations. Finally, derivations are irreversible, so a client holding a token with reduced privileges cannot recover the original, more privileged token.

We tag tokens with a computationally inexpensive HMAC-SHA256 [31] rather than signing with asymmetric digital signature functions. The first token of a device is tagged with a secret generated by the device. The token is shared with the device owner, but the secret never leaves the device. Changing the internal secret renders all existing tokens invalid.

To summarize, a token is a chain of frames and a single 32-byte HMAC-SHA256 tag. The initial tag is computed using the secret on the device and the content of the first frame, therefore the device must create any token with a single frame. MAC construction guarantees that it is impossible to recover the secret given the tag and the contents. Tags of derivations (i.e. tokens with multiple frames) can be computed by *any* entity holding a valid token using the existing tag as the secret to the same MAC operation with the content of the next frame. In other words, a token carries an immutable log of derivations where each link protects the next one. Once a device receives a token, it can verify it by replaying the MAC functions as described above over the log of derivations and comparing the computed tag value with the tag stored in the token. Figure 3 shows that while not identical, the chain of derivations is analogous to the chain of transactions in a blockchain and is thus verifiable in a similar way.

For every device, there is a *root token* (similar to the genesis in a blockchain) that authorizes every operation and is held by the device's owner. Only the device can generate its root token using a secret that it alone possesses. When the device receives a token over the network, it verifies the signature is intact and checks if all derivations are valid. Then, it checks that all constraints are met. If any of the checks fail, the token is discarded. If all checks pass, the entire token is considered well-formed and any requests in the token will be served. Finally, CAPLETS defines an efficient key exchange protocol for in-transit encryption.

For the camera tenancy application, we identify the following requirements:

1) A tenant must lose any access once their tenancy period ends,
2) Each secure operation needed by the application must finish under a second,
3) A tenant is authenticated by their public key,
4) Once tenancy is transferred, the owner cannot access the device until the tenancy ends,
5) A tenancy can be canceled early only by the tenant.

We fulfill the first requirement through the use of a timeout constraint on all tenant tokens. A timeout constraint carries a UTC time after which the server will reject any token carrying it. The second one we demonstrate in the evaluation (Section V). We meet the third one by introducing a public key constraint to the tenant token. The public key constraint requires the client to sign tokens they send with the corresponding private key of the public key carried in the constraint.

While the first three requirements are supported by CAPLETS without smart contracts, the last two need the introduction of a new ability. The core design of CAPLETS

implies that every client is essentially acting on behalf of the owner. In other words, any operation that any client can perform, the owner can also perform (since the owner holds the root token for the device). This design goes against requirements 4 and 5. Our solution is to add an ownership transfer protocol to CAPLETS. This is implemented by the following service interface:

```
service caplets_host {
    transfer_ownership(
        until: Time,
        key: array<u8>
    ) -> Token;
    early_cancel() -> bool;
    get_root_token() -> Token;
}
```

The `transfer_ownership` function is called by the owner to temporarily relinquish ownership rights carried in the original root token. This call immediately changes the server's internal signing key. The function returns a new root token signed with the new key which, for this application, must be the public key of the tenant. Thus the owner gets back a temporary root capability (that invalidate the previous root) that only the tenant can use. To implement this rights transfer, the device places a public key constraint on the returned token with the tenant's public key (which the tenant supplied when occupying the rental), meeting requirement 4. It also places a timeout constraint to meet requirement 1. The owner then passes this token to the tenant. Since the tenant can sign tokens, only the tenant has access to the device until the timeout expires. Further, because the tenant holds the temporary root, it can call `early_cancel` to revert its ownership and end its tenancy early.

As described in the next section, performing public key operations can incur between 2 to 3 orders of magnitude overhead to processor and power use. Since the root token held by the tenant has a public key constraint, this cost is applied to every single request made by the tenant. It is possible to amortize this overhead, however. Once the tenant has their constrained token, they can perform a request to `get_root_token` (essentially a session key for the duration of the rental) to obtain a root token that is only timeout constrained. After that point, the tenant can use the more efficient root token for the duration of the rental.

With these primitives in mind, we describe a CAPLETS-based smart camera scenario. We begin with the provisioning of the smart camera. When the camera is first enabled, it generates a secret, S_1. It uses this secret to tag the root token and transmit it to the owner. We refer to the contents of the root token as R, and the tagged token as $[R]_{S_1}$. The $[X]_Y$ notation is used to denote X is tagged with the secret Y. This step is shown in Figure 4.

Once the owner has the root capability, they can now execute any of the functions served by the camera. We now describe the transfer ownership scenario. Here, the tenant makes a move-in request $\mathcal{MI} = (K_T)$ where K_T is the public key of the tenant. The owner in turn makes a transfer ownership request $\mathcal{TO} = [R]_{S_1} \rhd transferOwnership(K_T, E)$ to the

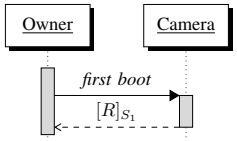

Fig. 4. Device provisioning in the CAPLETS approach.

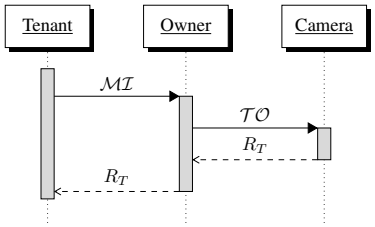

Fig. 5. Ownership transfer in the CAPLETS approach.

camera. Here, the $X \rhd Y$ notation means that Y is derived from X. E stands for the agreed-upon move-out time. The camera in turn generates a new secret S_2 and switches to it and responds with the public key constrained tenant root token, $R_T = ExpiresAt_E!MustSign_{K_T}![R]_{S_2}$. Here, the $X!Y$ denotes that Y is constrained on X. At this point, the owner loses access to the camera. This operation is shown in Figure 5.

Now, the tenant can exercise their root token since they have the private key for K_T. For instance, they can make a set stream encryption key request $\mathcal{SK} = \{R_T \rhd setStreamEncryptionKey(key)\}_{K_T}$. Here $\{X\}_Y$ means that X is digitally signed with the corresponding private key of Y. This is shown in Figure 6. However, using R_T directly like this incurs considerable overhead if a request is expected to be made frequently. In such cases, the tenant makes a request $\mathcal{GR}$ to the `get_root_token` function to acquire a cheaper to exercise token and uses it in the future. The cheaper root token, $R_C = ExpiresAt_E![R]_{S_2}$, is identical to R_T except that it does not have the public key constraint. The cheap set key request $\mathcal{SK}'$ is identical to $\mathcal{SK}$ except that it is derived from R_C instead of R_T. This scenario is shown in Figure 7. While the use of this operation is optional, we have observed performance improvements up to a factor of 500 after introducing and using it.

This approach does not suffer from any of the blockchain-related caveats mentioned in Section III. Specifically, it has no monetary cost to execute, has no computationally expensive operations that increase latency, and does not require additional operations at the end of a tenancy.

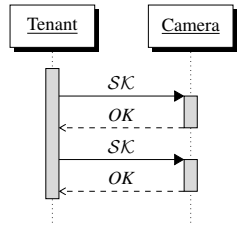

Fig. 6. Encryption key setting in the CAPLETS approach.

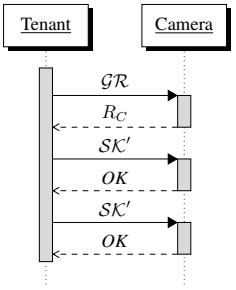

Fig. 7. Encryption key setting in the optimized CAPLETS approach.

V. EVALUATION

In this section, we empirically evaluate the performance of two smart camera systems – one Ethereum-based as discussed in Section III-A, and another one based on our proposed method as discussed in Section IV. For the Ethereum-based system, we deploy our smart contract in Ropsten[3] [32], which is one of Ethereum's public test networks (testnets) that uses the same algorithm as the main network (mainnet). Ropsten thus closely replicates the mainnet environment.

We invoke the smart contract functions using a Truffle [33] client from a Eucalyptus [34] private cloud instance containing two 2GHz CPU and 4GB memory and measure the time between invocation and return. We perform capability experiments on both the same instance as we use for the blockchain experiments and an STM32L475 microcontroller with an 80MHz ARM Cortex M4 processor with 64KB of RAM. We take 100 readings for each of the two functions (*transferTenancy* and *pollTenancy*) and present the average and standard deviation.

A. Performance of a Blockchain-Based System

As *transferTenancy* has to update the state, a corresponding transaction must be mined in the blockchain for it to execute. On the other hand, *pollTenancy* is a view function and does not require a transaction. Table II shows the average time taken to transfer tenancy and to poll tenancy.

As expected, *transferTenancy* is slower than *pollTenancy*. The average execution time of the former is 20.267 seconds whereas that of the latter is 0.992 seconds. That is, *transferTenancy* is more than $20x$ slower than *pollTenancy*. The high standard deviation of 10.796 seconds in *transferTenancy* is expected, as the function gets executed only when the transaction containing the function invocation gets mined, which can be the immediate next block that is mined or an arbitrary number of blocks after that. The observed minimum and maximum execution times for this function are 5.685 seconds and 60.302 seconds respectively.

B. Ether (ETH) Expenditure

In Ethereum, *gas* is a measure of the amount of computational effort required to execute an operation [35]. At the time of smart contract deployment, the deployer has to specify how much ETH he/she is willing to spend per unit of gas, i.e., gas price. If the gas price is higher, miners have a greater incentive to mine, resulting in a transaction getting mined faster.

In our experiment, we used the default gas price in Ropsten, which was 0.00000002 ETH at the time of deployment. In general, the cost of a transaction is the amount of gas used times the gas price. Table I shows the cost of contract creation and function execution according to the market value at the time of the deployment of the contract (20 April 2021). As we can see, apart from incurring an initial cost of USD 27.21, we also require USD 1.90 every time we call the *transferTenancy* function. As no mining effort is required for the execution of a view function, the *pollTenancy* function can be executed free of cost.

TABLE I
EXECUTION COST WITH A GAS PRICE P=0.00000002 ETH, 1 ETH=USD 2328.54.

action/function	**gas (G)**	**Ether (GxP)**	**USD**
contract creation	584216	0.01168432	27.21
transferTenancy	40731	0.00081462	1.90
pollTenancy	-	-	-

C. Performance of a CAPLETS-based System

Unlike the blockchain version, the capability-based implementation has no external dependencies, so read-only vs update request latency does not change in a significant way. However, whether the request token includes a public key constraint or not affects the latency considerably, so we report two sets of results. The public key constrained version is called only once per tenant.

We implement the public key constraint using ECDSA (Elliptic Curve Digital Signature Algorithm) on the secp256r1 curve. The constraint consists of the public key that has to sign the token.

Table II shows the average time it takes to execute a request on a particular host. Note that both implementations have the same security guarantees. Our experiments show that the capability-based approach is 5 and 6 orders of magnitude faster on *transferTenancy* and *pollTenancy* operations respectively compared to the blockchain. Even the

TABLE II
LATENCY RESULTS OF *transferTenancy* AND *pollTenancy* (OR *setKey*) OPERATIONS FOR BOTH BLOCKCHAIN-BASED AND CAPABILITY-BASED IMPLEMENTATION.

	Mean latency (stddev) in microseconds on virtual machine	Mean latency (stddev) in microseconds on microcontroller
Blockchain transfer	20,267,000 (10,796,000)	N/A
Blockchain poll_tenancy	992,000 (50,000)	N/A
Caplets transfer with pubkey	652 (32)	156,230 (167)
Caplets transfer without pubkey	7 (2)	922 (26)
Caplets set_key with pubkey	584 (34)	150,931 (527)
Caplets set_key without pubkey	3 (1)	457 (25)

[3]relevant transactions can be explored at https://bit.ly/3sQT61z

microcontroller version of capabilities performs 2 to 4 orders of magnitude faster than the blockchain implementation on a fully provisioned, resource-rich server.

Due to the very expensive elliptic curve operation, the tokens with the public key constraint take more than 2 orders of magnitude more time to use. However, since it doesn't incur any overhead on tokens that do not use it, and we can drop the constraint after the first request, we believe it is a good trade-off for the benefits public key cryptography brings in this application.

VI. Conclusion

Smart contracts have received revitalized attention due to the emergence of blockchains. Smart home systems, and IoT applications in general, can now embed business logic in smart contracts while providing the security and privacy commonly associated with blockchains. However, blockchain is inherently a resource-intensive technology and hence its application in systems with resource-constrained IoT devices is challenging. Moreover, applications performing time-sensitive operations are complex to implement using blockchains. Hence, we propose a new blockchain-independent approach to smart contracts that is resource-efficient and suitable for IoT applications. Our results show that the proposed method can outperform existing blockchain-based solutions while providing security and privacy. In the future, we plan to explore the use of our proposed method in different IoT applications beyond those of smart home systems.

References

[1] H. Lin and N. W. Bergmann, "Iot privacy and security challenges for smart home environments," *Information*, vol. 7, no. 3, p. 44, 2016.

[2] D. Geneiatakis, I. Kounelis, R. Neisse, I. Nai-Fovino, G. Steri, and G. Baldini, "Security and privacy issues for an iot based smart home," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2017, pp. 1292–1297.

[3] W. Ali, G. Dustgeer, M. Awais, and M. A. Shah, "Iot based smart home: Security challenges, security requirements and solutions," in *2017 23rd International Conference on Automation and Computing (ICAC)*. IEEE, 2017, pp. 1–6.

[4] M. N. Islam and S. Kundu, "Preserving iot privacy in sharing economy via smart contract," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2018, pp. 296–297.

[5] A. Qashlan, P. Nanda, and X. He, "Automated ethereum smart contract for block chain based smart home security," in *Smart Systems and IoT: Innovations in Computing*. Springer, 2020, pp. 313–326.

[6] Q. Xu, Z. He, Z. Li, and M. Xiao, "Building an ethereum-based decentralized smart home system," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2018, pp. 1004–1009.

[7] T. L. N. Dang and M. S. Nguyen, "An approach to data privacy in smart home using blockchain technology," in *2018 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE, 2018, pp. 58–64.

[8] P. K. Singh, R. Singh, S. K. Nandi, and S. Nandi, "Managing smart home appliances with proof of authority and blockchain," in *International Conference on Innovations for Community Services*. Springer, 2019, pp. 221–232.

[9] M. AbuNaser and A. A. Alkhatib, "Advanced survey of blockchain for the internet of things smart home," in *2019 IEEE Jordan international joint conference on electrical engineering and information technology (JEEIT)*. IEEE, 2019, pp. 58–62.

[10] M. Moniruzzaman, S. Khezr, A. Yassine, and R. Benlamri, "Blockchain for smart homes: Review of current trends and research challenges," *Computers & Electrical Engineering*, vol. 83, p. 106585, 2020.

[11] Y. N. Aung and T. Tantidham, "Review of ethereum: Smart home case study," in *2017 2nd International Conference on Information Technology (INCIT)*. IEEE, 2017, pp. 1–4.

[12] "Home — ethereum.org," https://ethereum.org/en/ [Online; accessed 22-Apr-2021].

[13] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian, "A survey on the adoption of blockchain in iot: Challenges and solutions," *Blockchain: Research and Applications*, p. 100006, 2021.

[14] "Ethereum Average Block Time Chart — Etherscan," https://etherscan.io/chart/blocktime [Online; accessed 22-Apr-2021].

[15] "Bitcoin Block Time Chart," https://bitinfocharts.com/comparison/bitcoin-confirmationtime.html [Online; accessed 23-Apr-2021].

[16] J. B. Dennis and E. C. Van Horn, "Programming semantics for multiprogrammed computations," *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, 1966.

[17] "Enterprise on Ethereum mainnet — ethereum.org," https://ethereum.org/en/enterprise/#private-vs-public [Online; accessed 30-Apr-2021].

[18] R. Yang, R. Wakefield, S. Lyu, S. Jayasuriya, F. Han, X. Yi, X. Yang, G. Amarasinghe, and S. Chen, "Public and private blockchain in construction business process and information integration," *Automation in Construction*, vol. 118, p. 103276, 2020.

[19] A. Pouraghily, M. N. Islam, S. Kundu, and T. Wolf, "Privacy in blockchain-enabled iot devices," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2018, pp. 292–293.

[20] N. Szabo, "Smart contracts: building blocks for digital markets," *EXTROPY: The Journal of Transhumanist Thought,(16)*, vol. 18, no. 2, 1996.

[21] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1027–1034.

[22] "Transactions — ethereum.org," https://ethereum.org/en/developers/docs/transactions/ [Online; accessed 26-Apr-2021].

[23] "Anatomy of smart contracts — ethereum.org," https://ethereum.org/en/developers/docs/smart-contracts/anatomy/ [Online; accessed 26-Apr-2021].

[24] F. Bakir, C. Krintz, and R. Wolski, "Capability-based access control for iot," University of California, Santa Barbara, Tech. Rep. 2021-02, 2021, https://cs.ucsb.edu/research/tech-reports/2021-02/.

[25] R. N. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for unix." in *USENIX Security Symposium*, vol. 46, 2010, p. 2.

[26] "Json web token (jwt)." [Online]. Available: https://tools.ietf.org/html/rfc7519

[27] S. Mullender, G. van Rossum, A. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba – A distributed Operating System for the 1990's," *IEEE Computer*, vol. 23, no. 5, May 1990.

[28] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Network and Distributed System Security Symposium*, 2014.

[29] D. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-based access control*. Artech House, 2003.

[30] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[31] H. Krawczyk, R. Canetti, and M. Bellare, "Hmac: Keyed-hashing for message authentication," 1997, [Online; accessed 26-Apr-2019] https://tools.ietf.org/html/rfc2104.

[32] "TESTNET Ropsten (ETH) Blockchain Explorer," https://ropsten.etherscan.io/ [Online; accessed 26-Apr-2021].

[33] "Sweet Tools for Smart Contracts — Truffle Suite," https://www.trufflesuite.com/ [Online; accessed 26-Apr-2021].

[34] "Eucalyptus Documentation," https://docs.eucalyptus.com/eucalyptus/4.3.0/ [Online; accessed 12-Sep-2017].

[35] "Gas and Fees — ethereum.org," https://ethereum.org/en/developers/docs/gas/ [Online; accessed 25-Apr-2021].