Access Control with Delegation for Smart Home Applications

Tam Le
Michigan State University
East Lansing, Michigan
letam@msu.edu

Matt W. Mutka
Michigan State University
East Lansing, Michigan
mutka@msu.edu

ABSTRACT

With the emergence of smart home applications, it is important to have flexible access control so that users can create/transfer their permissions in a convenient way. We propose a lightweight authorization protocol with support of a delegation chain in which a user can easily transfer (part of) his/her access rights to smart appliances in the form of a Bloom filter. The security of our protocol is based on the false positive rate of a Bloom filter. A prototype has been built for evaluation.

CCS CONCEPTS

• **Security and privacy → Access control**;

KEYWORDS

acccess control, delegation, Bloom filter, IoT

ACM Reference Format:
Tam Le and Matt W. Mutka. 2019. Access Control with Delegation for Smart Home Applications. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '19), April 15–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3302505.3310076

1 INTRODUCTION

Internet of Things (IoT) technologies have emerged with various smart appliances such as smart door locks, smart light bulbs, sensors, HVAC, surveillance camera, etc. A person may possess several to hundreds of smart devices and vice versa, a single device can be shared between multiple users. However, smart home users have been facing difficulties in sharing their devices. Due to the diversity of users in home environments, including various types of visitors, it is difficult to define an access control policy that can satisfy all of user's ad-hoc behaviors and demands [10].

In Fig. 1, we present a common scenario: Bob listed his house on Airbnb. He can create a digital key that provides full control to all devices in the house, such as HVAC, lighting system, etc. but limited access to the door lock. A security company is also authorized to receive access logs from his door lock but not permitted to enter the house. Bob gives the key to his renter, Dave, whose family should also be able to use the system. It would be more convenient for Dave to grant permissions to his family members instead of Bob. As

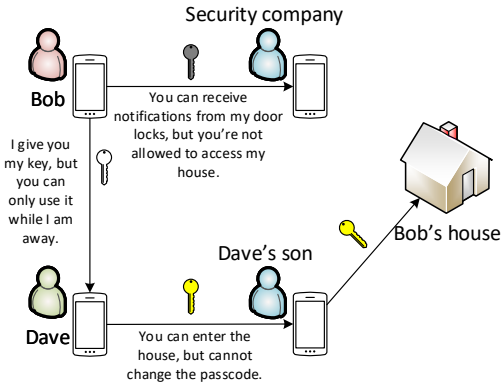

Figure 1: A smart home scenario

Dave is not allowed to change the passcode of the lock, he can only delegate the same or lower permissions. When the lease expires, Bob revokes the key, including sub-keys created by Dave.

The above scenario addresses the importance of having a scalable and flexible access control mechanism so that a user can create various permissions, depending on his needs and capability. Moreover, most IoT devices have limited resources, such as low computational power or battery capacity. Therefore they cannot support complicated security mechanisms.

We propose a lightweight delegation mechanism using Bloom filters [2], which is inspired by Foley et al. [5]. Leveraging the property that items can only be inserted but are difficult to remove from the Bloom filter, they present a decentralized authorization model where permissions are represented in the forms of Bloom filters. Although the idea has potential, the proposed model was not secure enough to be adapted to real-world applications. In this work, we extend [5] and build an access control mechanism that allows users to transfer their permissions to their smart home devices to other users. We create a trusted chain within a small packet that the devices can verify without complicated cryptographic algorithms. Although the Bloom filter has a false positive rate, a sufficient security level can be achieved by setting appropriate parameters.

2 RELATED WORK

Many commercial products have been studied and found to have insufficient security mechanisms. Examining several smart appliances, Notra et al. [11] highlight the lack of encryption, appropriate authentication, message integrity checks and privacy implications. Unauthorized and over-privileged access have also been found in several popular smart home systems [4, 8].

In terms of end-user experiences, a case study on several commercial smart home devices [14] points out that although each

IoTDI '19, April 15–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6283-2/19/04…$15.00
https://doi.org/10.1145/3302505.3310076

studied device employs a different access control mechanism, none provides a convenient method to share access with other users. In a more recent study [15], the authors also express their concerns about the imbalance of power between multiple users who are supposed to have the same role, where some users may have (intentionally or unintentionally) more privileges than others.

Roman et al. [12] addresses the importance of an access control model for distributed IoT that should support granular policies, delegation and inexpensive computational overhead. In this sense, capability-based access control (CBAC) is found to be more suitable for IoT environments than other traditional access control models [9]. Others [7, 13] propose a CBAC framework that allows users to manage and share their own access control by issuing capability tokens, which will be verified by the IoT devices. Although their approach works well with 1 level of delegation from 1 owner to multiple users, it will create more computational overhead for the device when the delegation chain expands to more than 1 hop. Hussein et al. [9] introduce a Community CBAC framework, in which an IoT community is formed by IoT devices that shares the same mission, then a more capable device in the community can make decisions on behalf of those with limited resources. How to build such community with trust, however, was not addressed.

SmartTokens [3] introduced a delegable access control system for NFC-enabled smart phones without a central authority. Although SmartTokens uses symmetric cryptography, which is suitable for constrained devices, users still need to present all delegated tokens through the delegation chain in order to be verified. More recently, WAVE [1] introduced a blockchain-based decentralized authorization system for IoT. WAVE supports non-interactive delegation with fine-grained access control policies using smart contracts. However, since running as blockchain nodes is not feasible for constrained devices, some trusted gateways are still required for the devices to interact with the blockchain network.

3 BLOOM FILTERS AS PERMISSIONS

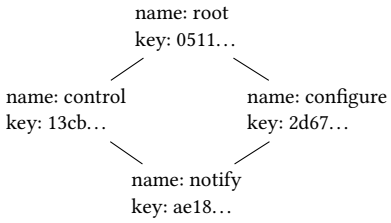

Figure 2: Example of a permission lattice of a smart lock

A Bloom filter (BF) is a probabilistic data structure that can check for membership of a given item. An empty filter is a bit array of length m that is initialized to all zeros. An item is added to the filter by setting k bits, whose positions are generated by k hash functions. A BF can tell that a given item is definitely not in the set if any of its k bits is 0. Otherwise, if all of these bits are 1, the item may or may not belong to the set with a false positive rate.

Foley et al. [5] introduces an approach to implement permissions as BFs. Given 2 permission x, y, x is considered lower than y (denoted by $x \leq y$) if holding y also implies holding x. A BF format

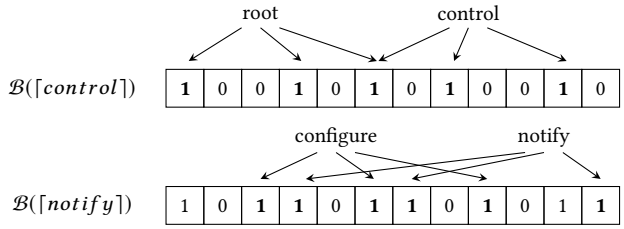

Figure 3: Example of 'notify' and 'control' permission

of x is a filter consisting of all permissions equal to or higher than x. By keeping the root permission as a secret, it is easy to create subsequent permissions but difficult to obtain a higher permission. Consider a lock system [1] where users can control, configure the lock and receive notifications. A permission lattice $P = \{root, control, configure, notify\}$ can be created for this lock as in Fig. 2 [2], where each permission is represented by a key. The highest permission *root* is kept as a secret between the owner and the lock, while all other values can be public. Let $\lceil p \rceil = \{x \in P | x \geq p\}$ and $\mathcal{B}(X)$ be the BF to which items from X are added. Suppose Carol is allowed to control the door lock. Her BF permission will be $\mathcal{B}(\lceil control \rceil) = \mathcal{B}(\{root, control\})$. To grant her father John a notification-only permission, Carol adds $configure, notify$ to her permission to create $\mathcal{B}(\lceil notify \rceil) = \mathcal{B}(\{root, control, configure, notify\})$ without knowing *root*. To verify $\mathcal{B}(\lceil notify \rceil)$, the lock checks if all values $root, control, configure, notify$ are presented in the filter. It should be noted that the presence of $root$, $control$, $configure$ does not mean John has those permissions; instead it implies that his $notify$ permission was granted by a legitimate user.

The above takes advantage of the non-reversibility property of BFs: items can be added but cannot be removed from the filter as they may share the same bits. Fig. 3 describes Carol's $\mathcal{B}(\lceil control \rceil)$ and John's $\mathcal{B}(\lceil notify \rceil)$ using a 12-bit BF. John can try to remove $notify$ from his filter to obtain $\mathcal{B}(\lceil control \rceil)$. However, since *root* and $notify$ share the same bit at position 4, removing $notify$ by setting all of its bits back to 0 will also invalidate *root*. Since *root* is unknown, John cannot know which bits should be retained, thus fails to get the permission. Similarly, Carol is not able to remove *control* as well since it overlaps with *root* at bit 6.

The probability that there is at least 1 overlapping bit in a BF is

$$Pr\{overlap\} = 1 - Pr\{nonoverlap\} = 1 - \frac{m!}{(m - nk)!m^{nk}} \quad (1)$$

where m is the filter length, n is the number of items in the filter and k is the number of bits used to index an item to the filter.

Although Foley et al. [5] claim that a high probability of overlap can be achieved with appropriate m, n, k, in practice it is not enough to secure the filter. Figure 4 shows the probability of overlap when there are 2 items ($n = 2$) with $m = \{512, 1024\}$ and $k = \{1, 2, \ldots, 50\}$. With a 1024-bit filter, k should be at least 45 to have a sufficient probability of overlap. However, with 45 bits, there are only at most 2^{45} combinations of overlapping bits. It should be emphasized that in practice, the probability to find the correct bits

[1] http://support.getvera.com

[2] Here we assume control and configure are independent permissions. The policies to create a permission lattice may depend on manufacturers.

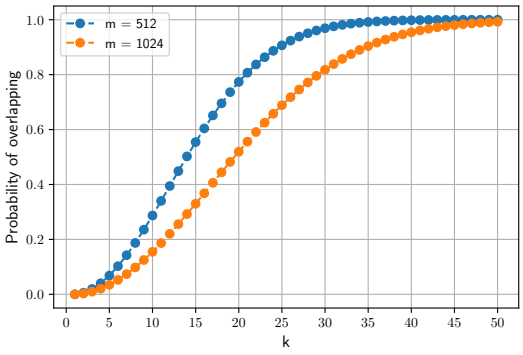

Figure 4: Probability of overlapping with 512 and 1024-bit Bloom filters containing 2 items

is much less than 2^{-45}, as it is very unlikely that all of the bits will overlap. To achieve sufficient security, k should be more than 128 bits. As a result, the filter length m must also be increased to reduce false positive rate as well as prevent the filter to be filled up quickly when more items are added, which is not a scalable solution.

To solve the problem, we present a new solution by generating all of the items' values dynamically for each delegation so that the same item will be inserted using 2 different sets of bits in 2 different BF permissions, depending on the permission information such as user ID, permission name, etc. As a result, any attempt to remove an item will not be successful since a valid permission would use a different set of bits than the one resulted from the removal. We provide more details in the following section.

4 DELEGABLE PERMISSIONS

4.1 Protocol overview

We consider 4 entities in our domain:

- Owner: has full control to his/her devices.
- Smart device: has a built-in permission lattice where each node is defined by a pair $\{name, key\}$ where key is kept secret between the owner and the device. This is different from [5] where only the highest permission is secret.
- Normal user: obtains permissions to a device from an owner or authorized user. If the permission is given by an authorized user, it needs to be activated at the device before usage.
- Authorized user: is authorized to issue lower permissions than his/her own permissions to other users. However, permission cannot be issued directly but in forms of certificates that will be activated at the device.

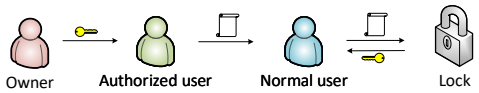

Figure 5: Delegation procedure

The protocol involves 5 procedures:

- $\mathcal{B}_p \leftarrow \texttt{PermGen}(p, id, t)$: to issue to user id a BF permission $\mathcal{B}_p$ that expires at time t.
- $\mathcal{B}_p^{del} \leftarrow \texttt{DelPermGen}(p, id, t)$: to issue to user id a delegation permission $\mathcal{B}_p^{del}$ that expires at time t. Users can use $\mathcal{B}_p^{del}$ to issue authorization certificate $\mathcal{B}_q^{auth}$ where $q < p$.
- $\{0 \text{ or } \mathcal{B}_q\} \leftarrow \texttt{Activate}(B_q^{auth})$: to check an authorization certificate $\mathcal{B}_q^{auth}$ and return BF permission $\mathcal{B}_q$ if the certificate is valid, otherwise return 0.
- $\{0 \text{ or } 1\} \leftarrow \texttt{Verify}(\mathcal{B}_p)$: to check a BF permission $\mathcal{B}_p$ and return 1 if the permission is valid and grant access accordingly, otherwise return 0.
- $K \leftarrow \texttt{KeyDerive}(B, salt)$: to derive a key K from a BF permission $\mathcal{B}$. Here the BF permission is used as master key or key material to generate a one-time session key K for secure message exchange between the user and device.
- $\texttt{Enc}(K|m)$ and $\texttt{Dec}(K|c)$: symmetric encryption and decryption algorithms using key K.

Table 1: Notations and definitions

$(P, \leq)$	a permission lattice
$\lceil p \rceil$	set of all permissions higher than or equal to p i.e. $\lceil p \rceil = \{x \in P \mid x \geq p\}$
$\lfloor p \rfloor$	set of p's successor permissions, i.e. permissions lower than p, i.e. $\lfloor p \rfloor = \{x \in P \mid x < p\}$
$\mathcal{B}(X)$	Bloom filter to which items from set X are added
$\mathcal{B}_p$	BF permission that allows any access lower than or equal to p, i.e. $\mathcal{B}_p = BF(\lceil p \rceil)$
$\mathcal{B}_p^{del}$	BF delegation permission that is used to create permissions lower than p
$\mathcal{B}_p^{auth}$	BF authorization certificate for permission p
K_p^{auth}	authorization key derived from $\mathcal{B}_p^{auth}$

The reason not to use BF permissions directly as access tokens but for encryption keys is three-fold. First, a BF permission should not be presented to the device in plain text to avoid replay attacks. Second, an attacker may create a filter with many 1s to increase the probability of matching. However, by using symmetric keys, the device regenerates the BF permissions by themselves, which means there will be no redundant 1s, hence that kind of attack can be avoided. Lastly, an access request is often associated with certain data. For example, in a request to lock/unlock a door or change the AC temperature, the lock action and the temperature are sensitive data that may expose user privacy. Therefore, using BF permissions as keys both helps to avoid revealing the permission as well as protect user data. Table 1 provides notations used in our scheme.

4.2 Bloom filter index

To distinguish BF permissions of different users, we define a public value $PID = \{perm\_name, userid, exp\_time\}$ that serves as a unique identifier of a permission. For example, $\{control, Alice, 191231\}$ identifies Alice's *control* permission, which expires on Dec 31, 2019.

To insert a permission to a BF, we apply the secure index mechanism [6]. Given a pseudo-random function f, the insertion of permission p to BF permission $\mathcal{B}$ with id PID works as follows:

Algorithm 1 Bloom permission generation

```
1: function PermGen(p, id, t)
2:     PID ← p.name||id||t
3:     B_p ← ∅
4:     for each x in ⌈p⌉ do
5:         y ← f(PID, f(x.name, x.key))
6:         Insert x to B_p using y
       return B_p
7: function DelPermGen(p, id, t)
8:     PID ← p.name||id||t
9:     m ← f(PID, p.key)
10:    B_p^del ← ∅
11:    for each x in ⌈p⌉ do
12:        x' ← x
13:        x'.key ← f(m, x.key)
14:        y' ← f(PID, f(x'.name, x'.key))
15:        Insert x' to B_p^del using y'
16:    ⌊p'⌋ ← ∅
17:    for each x in ⌊p⌋ do
18:        x' ← x
19:        x'.key ← f(m, x.key)
20:        ⌊p'⌋ ← ⌊p'⌋ ∪ {x'}
       return B_p^del, ⌊p'⌋
```

(1) Compute $x = f(p.name, p.key)$
(2) Compute $y = f(PID, x)$
(3) Divide y to k chunks $\{y_1, \ldots, y_k\}$, each of which serves as an index to the Bloom filter.

The permission generation is described by procedure `PermGen` in Algorithm 1.

4.3 Protocol Details

4.3.1 Permission delegation. Fig. 6 depicts our delegation protocol. The owner gives Alice permission a in the form of $\mathcal{B}_a$. If Alice is an authorized user, she is given another BF called *delegation permission* $\mathcal{B}_a^{del}$, by which she can delegate parts of her rights by inserting lower permissions than a to it. The generation of delegation permission, described by procedure `DelPermGen` in Algorithm 1, is identical to the normal permission, except that it works on a new set of permission values, which are derived from the original ones by hashing them with the permission ID. The set of successor permissions of a but with the new derived values (denoted by $\lfloor a' \rfloor$) is then sent to Alice. Given $\mathcal{B}_a^{del}$ and $\lfloor a' \rfloor$, Alice can create a sub permission $b < a$, by adding corresponding items to $\mathcal{B}_a^{del}$. Since this new BF is created by Alice - an authorized user, we call it an *authorization permission* and denote it as $\mathcal{B}_b^{auth}$ to distinguish it with the delegation permission $\mathcal{B}_a^{del}$ that is created by the owner.

It is noteworthy that any delegation with the same permission b from Alice will result in the same $\mathcal{B}_b^{auth}$, thus to make each delegation unique, Alice chooses a random secret x_b and computes an *authorization key* K_b^{auth} that is derived from x_b and $\mathcal{B}_b^{auth}$, i.e. $K_b^{auth} = \texttt{KeyDerive}(\mathcal{B}_b^{auth}, x_b)$. To issue b to Bob with ID id_B, Alice generates an authorization certificate $cert_B^A$ by encrypting x_b and Bob's new permission ID $\{b, id_B, t_B\}$ and sends the certificate together with K_b^{auth} to Bob. The use of both certificate and authorization key is two-fold. The encryption of Bob's permission ID by Alice's key prevents Bob from altering the information. On the other hand, sending the certificate alone is vulnerable to a replay attack, when any malicious party who is able to overhear the message can just resend it to obtain Bob's permission. Bob's authorization permission serves as a temporary session key that is only known to Bob, thus can prevent the attack.

4.3.2 Activation protocol. Fig. 7a depicts the activation protocol. To activate permission b given by Alice, Bob first sends his authorization certificate $cert_B^A$ and Alice's corresponding permission information PID_A. The device then can compute Alice's permission $\mathcal{B}_a$, decrypt the certificate to get all information needed to compute the authorization key K_b^{auth} and Bob's permission $\mathcal{B}_b$, which is then sent back in encrypted form by K_b^{auth}.

4.3.3 Verification protocol. As the activation protocol, verification also involves secure message exchange using the BF permission as an encryption key. To request a service under permission b, Bob encrypts the request data and sends it with his public permission information, based on which the device computes the corresponding key $\mathcal{B}_b$ to decrypt the message. If the permission is valid, the device executes the request and sends back the encrypted response to Bob. Both parties can also use challenges for mutual authentication as in the activation protocol. The protocol is depicted in Fig. 7b.

5 SECURITY ANALYSIS

Our analysis is under an assumption that users' smart phones and PCs are capable of a secure permission exchange. Since BF permissions are used as a symmetric encryption key, the security of our scheme depends on the difficulty of forging such keys. As discussed in section 3, there are 2 possible attacks that are analyzed in [5] to infer a higher permission from 1 or more permissions: (1) removal attack where the attacker tries to remove items from the BF and (2) aggregation attack by computing the intersection between 2 or more filters. The removal attack does not work in our scheme since each item is hashed with the BF permission ID that contains the lowest permission. For example, given a BF permission $\mathcal{B}_b$, an attacker wants to remove b to obtain $\mathcal{B}_a$ where $a > b$. However in $\mathcal{B}_b$, permission a is binded to $b.name$ while in a valid $\mathcal{B}_a$ it should be binded to the name of itself. Therefore, any attempt of removal will result in a meaningless BF as it no longer matches with the attacker's desired condition. Likewise, aggregation or collusion between users would not work either. As a result, the only possibility left is to do an exhaustive search to find the filter. The difficulty of the search depends on the filter size m, the number of permission items n and the codeword length k. Since the bits are generated by pseudo random functions, we assume they are under a uniform distribution. After inserting n items using k bits per item, the probability that a bit $b_i, i = 1 \ldots m$ is still 0 is

$$P(b_i = 0) = \left(1 - \frac{1}{m}\right)^{kn} \tag{2}$$

Therefore, the false positive rate of Bloom filter is

$$fpr = \left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]^k \approx (1 - e^{-kn/m})^k \tag{3}$$

Owner		Alice		Bob
To delegate a, expiring at t_A:				
$\mathcal{B}_a \leftarrow \texttt{PermGen}(a, id_A, t_A)$	$\xrightarrow{\mathcal{B}_a}$	Save $\mathcal{B}_a$		
If is_authorized(id_A):				
$\{\mathcal{B}_a^{del}, \lfloor a' \rfloor\} \leftarrow \texttt{DelPermGen}(a, id_A, t_A)$	$\xrightarrow{\mathcal{B}_a^{del}, \lfloor a' \rfloor}$	Save $\mathcal{B}_a^{del}, \lfloor a' \rfloor$ To delegate b, expiring at t_B: $\mathcal{B}_b^{auth} \leftarrow \mathcal{B}_a^{del} \cup \mathcal{B}(\lceil b' \rceil - \lceil a' \rceil)$ Choose $x_b \in \{0,1\}^*$ $K_b^{auth} \leftarrow \texttt{KeyDerive}(\mathcal{B}_b^{auth}, x_b)$ Choose $x_a \in \{0,1\}^*$ $K_a \leftarrow \texttt{KeyDerive}(\mathcal{B}_a, x_a)$		
		$cert_B^A \leftarrow \texttt{Enc}(K_a \mid \{b, id_B, t_B, x\})$	$\xrightarrow{K_b^{auth}, cert, x_a}$	Save $K_b^{auth}, cert_B^A, x_a$

Figure 6: Permission delegation

Bob $id_B, K_b^{auth}, cert_B^A, x_a$		Device
$PID_A \leftarrow a \mid\mid id_A \mid\mid t_A$	$\xrightarrow{PID_A, cert}$	$\{a, id_A, t_A\} \leftarrow \texttt{parse}(PID_A)$ $\mathcal{B}_a \leftarrow \texttt{PermGen}(a, id_A, t_A)$ $K_a \leftarrow \texttt{KeyDerive}(\mathcal{B}_a, x_a)$ $\{b, id_B, t_B, x_b\} \leftarrow \texttt{Dec}(K_a \mid cert_B^A)$ $\mathcal{B}_b^{auth} \leftarrow \texttt{DelPermGen}(b, id_A, t_A)$ $K_b^{auth} \leftarrow \texttt{KeyDerive}(\mathcal{B}_b^{auth}, x_b)$ $\mathcal{B}_b \leftarrow \texttt{PermGen}(b, id_B, t_B)$
$\{\mathcal{B}_b\} \leftarrow \texttt{Dec}(K_b^{auth} \mid m_1)$	$\xleftarrow{m_1}$	$m_1 \leftarrow \texttt{Enc}(K_b^{auth} \mid \mathcal{B}_b)$

(a) Permission activation

Bob $id_B, \mathcal{B}_b$		Device
Choose $x \in \{0,1\}^*$ $K_b \leftarrow \texttt{KeyDerive}(\mathcal{B}_b, x)$ $m_1 \leftarrow \texttt{Enc}(K_b \mid reqData)$		
$PID_B \leftarrow b \mid\mid id_B \mid\mid t_B$	$\xrightarrow{PID_B, m_1, x}$	$\{b, id_B, t_B\} \leftarrow \texttt{parse}(PID_B)$ $\mathcal{B}_b \leftarrow \texttt{PermGen}(b, id_B, t_B)$ $K_b \leftarrow \texttt{KeyDerive}(\mathcal{B}_b, x)$ $reqData \leftarrow \texttt{Dec}(K_b \mid m_1)$ If $id_B \notin$ REVLIST and $t_B \geq$ time() $res\_data \leftarrow \texttt{Exec}(reqData)$
$res\_data \leftarrow \texttt{Dec}(K_b \mid m_2)$	$\xleftarrow{m_2}$	$m_2 \leftarrow \texttt{Enc}(K_b \mid resData)$

(b) Permission verification

Figure 7: Activation and verification protocol

The average number of bits that is set to 1 after n insertions is

$$s = m \times P(b_i = 1) = m\left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right] \quad (4)$$

Hence, the search space to find a Bloom permission is $S = \binom{m}{s}$.

Eq. (3) shows that as the delegation chain expands, the fpr will increase since more items are added to the filter. Therefore, given a fixed m, we want to find a value of k that can provide large enough search space when n is small while still keep a reasonable false positive rate when n is large. We choose 2 as the lower bound of n since *root* is always present in any BF permission[3]. For the upper bound that basically is the total number of nodes in the permission lattice, we choose 20 by intuition as we believe an IoT device would not have more than 20 different types of permissions. Table 2 shows fpr and search space S and with 3 sizes of the BF and different values of k. While fpr is monotonically increasing, S will start to decrease when s exceeds $\lceil \frac{m}{2} \rceil$. If we take the difficulty of finding a 128-bit key (whose search space is 3.4×10^{38}) as the minimum security threshold, then according to the table, except for $m = 256$ when S drops to 3.15×10^{30} with $n = 20$, k in the range of $[16, 32]$ can provide both good false positive rate and search space.

The fpr is the probability that an item is recognized despite not being in the set. Thus, it implies the probability that a user can successfully claim his given permission to be another one that he is not granted. It should be noted that the calculated false positive rate by eq. (4) is only for 1 item, while a valid BF permission requires all items to be recognized without any redundant bit. To claim a false BF permission, not only the lowest but all n items should be false positives since every item is associated with the lowest one when the permission is generated. Thus, for a BF permission containing n items to be falsely accepted, the actual probability is fpr^n, which exponentially increases to be negligible as n increases.

6 IMPLEMENTATION AND DISCUSSION

We use an Arduino MKR1000 board with a 32-bit low power ARM microcontroller as a smart device and a Java client that runs on a Dell laptop. The key size and BF size are both 256 bits. ArduinoLibs [4] library is used for cryptographic operations. The pseudo random function f and the key derivation function use HMAC-SHA256 and

[3] Since only the owner holds secret keys, it is unnecessary for them to use BF permission consisting of only *root* to control their devices, thus we omit the case when n = 1.

[4] https://rweather.github.io/arduinolibs/crypto.html

Table 2: Search space (S) and false positive rate (fpr)

m	k	fpr (n=2)	fpr (n=20)	S (n=2)	S (n=20)
256	8	1.8156e-10	0.0021760	1.0079e+25	3.0697e+75
	16	1.3206e-15	0.0045108	1.1362e+39	1.5845e+65
	32	1.0791e-21	0.0645177	5.3677e+57	3.1516e+30
512	8	8.0289e-13	2.6919e-05	8.4114e+29	1.5410e+128
	16	3.2966e-20	4.7352e-06	4.6757e+49	1.3348e+152
	32	1.7441e-30	2.0348e-05	1.1827e+79	8.0194e+131
1024	8	3.3377e-15	1.9172e-07	6.2091e+34	1.8376e+182
	16	6.4463e-25	7.2463e-10	4.9759e+60	1.5588e+257
	32	1.0867e-39	2.2422e-11	2.0996e+100	inf

the encryption algorithm is AES-256. Despite not being specifically designed to be lightweight, these primitives can work well with Arduino devices. Each item is inserted using 32 bits of indices. The protocol runs over TCP.

In our first test, the Arduino simulates a door lock with the permission lattice shown in Fig. 2. Table 3a shows the processing time of *control* and *notify* permissions under the two types of requests. To test for scalability, we also implement a simple lattice with 20 permission nodes and measure the processing time with different number of nodes added to the filter. The average processing time is given in Table 3b. Though the activation process has to perform more operations than the access request, it still takes less than 200 ms even with a large number of items present in the filter.

Compared to [13] which is CBAC and relies on Elliptic Curve Cryptography (ECC), our scheme is faster. In [13], a capability token includes 2 signatures. Using micro-ecc [5] library, we found that it would take 356 ms to verify 1 token on our device, in which a single signature verification takes 158 ms. As a result, if the delegation expands, it would take the device more than a second to process a request. On the other hand, our scheme can take less than 100 ms.

Table 3: Processing time of request

(a) With the simulated smart lock

	control	notify
Access request	32 ms	38 ms
Activation request	N/A [a]	68 ms

[a]Since $control$ can only be given by the owner, there is no activation needed.

(b) With different number of items

n	10	12	14	16	18	20
Access	57 ms	62 ms	69 ms	73 ms	78 ms	83 ms
Activate	121 ms	135 ms	148 ms	161 ms	175 ms	188 ms

In a certificate-based approach, a parent certificate must be piggybacked in every delegation, hence a user needs to send a number of certificates to prove its authorization. However, in our approach, the size of a BF permission is always a constant despite the expansion of the delegation chain. In addition, the permissions are verified using BFs and symmetric-key algorithms, which is less computationally expensive than digital signatures. The scalability also supports flexible and fine-grained access control, since users can add various operations to the filter, as long as they do not violate the permission hierarchy. In addition, since we use very lightweight techniques, the solution can be applied to a wide-range of devices, including those that may be less powerful than the Arduino model. To save space when the lattice size is large, all of the key values can be generated from a single seed.

[5]https://github.com/kmackay/micro-ecc

7 CONCLUSION

We propose a lightweight distributed authorization protocol with support of delegation for home environments. Access right to a smart device is transferred in form of a Bloom filter with secured hashing to prevent the permission from being forged. Thanks to the inability to remove items in the Bloom filter, a user cannot recreate a permission higher that what he/she is holding, but still is able to transfer lower permissions. Our protocol can be implemented with little encryption, thus is suitable for resource-constrained devices.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation. Any opinions, findings, conclusions or recommendations reflects the views of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Michael P Andersen, John Kolb, Kaifei Chen, Gabriel Fierro, David E Culler, and Raluca Ada Popa. 2017. WAVE: A Decentralized Authorization System for IoT via Blockchain Smart Contracts. (2017).
[2] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422–426.
[3] Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Sandeep Tamrakar, and Christian Wachsmann. 2012. *SmartTokens: Delegable access control with NFC-enabled smartphones.* Springer.
[4] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2016. Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy.*
[5] Simon N Foley and Guillermo Navarro-Arribas. 2013. A Bloom Filter Based Model for Decentralized Authorization. *International Journal of Intelligent Systems* (2013).
[6] Eu-Jin Goh et al. 2003. Secure Indexes. *IACR Cryptology ePrint Archive* (2003).
[7] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. 2013. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling* (2013).
[8] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart Locks: Lessons for Securing Commodity Internet of Things Devices. In *ASIA CCS '16.* ACM.
[9] D. Hussein, E. Bertin, and V. Frey. 2017. A Community-Driven Access Control Approach in Distributed IoT Environments. *IEEE Communications Magazine* (2017).
[10] Tiffany Hyun-Jin Kim, Lujo Bauer, James Newsome, Adrian Perrig, and Jesse Walker. 2011. Access right assignment mechanisms for secure home networks. *Journal of Communications and Networks* (2011).
[11] S. Notra, M. Siddiqi, H. Habibi Gharakheili, V. Sivaraman, and R. Boreli. 2014. An experimental study of security and privacy risks with emerging household appliances. In *2014 IEEE Conference on Communications and Network Security.*
[12] Rodrigo Roman, Jianying Zhou, and Javier Lopez. 2013. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks* (2013).
[13] Antonio F Skarmeta, José L Hernández-Ramos, and M Victoria Moreno. 2014. A decentralized approach for security and privacy challenges in the internet of things. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on.* IEEE.
[14] Blase Ur, Jaeyeon Jung, and Stuart Schechter. 2013. The current state of access control for smart devices in homes. In *Workshop on Home Usable Privacy and Security (HUPS).*
[15] Eric Zeng, Shrirang Mare, and Franziska Roesner. 2017. End user security & privacy concerns with smart homes. In *Symposium on Usable Privacy and Security (SOUPS).*