# A Bloom Filter based Model for Decentralised Authorisation

Simon N. Foley
Dept. of Computer Science
University College Cork
s.foley@cs.ucc.ie

Guillermo Navarro-Arribas
Dept. of Information and Communications Engineering
Universitat Autønoma de Barcelona
gnavarro@deic.uab.cat

**Abstract**

A decentralised authorisation mechanism is proposed that uses Bloom filters to implement authorization delegation. This lightweight mechanism is unlike conventional approaches that typically rely on public key certificates to implement distributed delegation. In taking an approach based on one-way hash functions, the mechanism may be preferable for use in computationally constrained environments where public-key cryptography is not desirable.

## 1 Introduction

Access Control systems such as [2, 8, 24] are intended to provide a decentralized approach to constructing and interpreting authorization relationships between principals (public keys). Unlike a centralized authorization server-based approach, authorization rules are defined and signed locally by issuing principals and these cryptographic delegation credentials can be distributed in any manner to suit the principals involved. In computational terms, and assuming non-threshold delegation, a delegation action requires a public-key based signing operation, while a verifier must carry out a public key based signature verification operation. These, relatively computationally expensive, operations may not be desirable in applications where performance and/or energy consumption are constrained. Such applications include sensor networks, RFID systems, and some smartcards..

One-way cryptographic hash-functions have been used previously to provide alternatives to symmetric key cryptography in the implementation of authentication protocols, for example, [14, 35]. These lightweight implementations are motivated by constraints on computation and power consumption [35], or other concerns, such as the availability of cryptographic operations [14]. This paper extends this idea by developing a distributed authorization delegation mechanism that is constructed entirely using one-way cryptographic hash functions. It is suggested in [10] that hash-chains, used for example in micropayment [32, 31] and password systems [18], can be used to provide a rudimentary form of authorisation delegation. In [10] a hash-chain provides a total ordering of permissions and delegation corresponds to exposing a hash-value to a recipient, while authorization verification corresponds to a test for chain membership. However, a hash-chain is not useful as a general authorization system since it can only support sets of permissions that form *total* orderings. By implementing permissions as Bloom filters [3], the approach described in this paper supports the more practical lattice and partial orderings of permissions.

The paper is structured as follows. Section 2 provides a review of Bloom filters and some particular properties that are used in the development of Bloom permissions, which are described in Section 3. Section 4 considers potential attacks on the system and provides an analysis of the security of the proposed mechanism. Finally, Section 5 discusses the proposal with regard to related work, and Section 6 concludes the paper.

## 2 Bloom filters

The proposal relies on the use of Bloom filters introduced by B.H. Bloom in 1970 [3], as a space and time efficent method for testing membership of elements in a given set.

A Bloom filter is a set of $n$ elements implemented as a bit vector $\mathcal{B} = \mathcal{B}[1], \ldots, \mathcal{B}[m]$ with initial value 0. It uses a family of one-way hash functions $f_i$ $(i : 1..k)$ with an image uniformly distributed over the interval $[1..m]$. To add an element $x$ to the filter (set), one computes the values $x_i = f_i(x)$ for $i : 1..k$ and sets the bits $\mathcal{B}[x_i] = 1$ in $\mathcal{B}$. To check if an element $y$ is in the filter, one verifies that, given the values $y_i = f_i(y)$ for $i : 1..k$, then the bits $\mathcal{B}[y_i]$ are equal to 1.

Bloom filters are mostly used as probabilistic data structures to encode sets of elements. Examples of their application include Google's BigTable distributed storage system [6], which uses them to check in-memory data to reduce disk lookups, and the Squid Web Proxy [29], which implements its cache digest with Bloom filters.

In addition to being an efficient data structure for encoding sets, Bloom filters have some interesting security characteristics. Given a value it is fast and easy to check whether the element is in the filter. However, given a suitably configured filter, it is not feasible to calculate the list of elements contained in the filter.

## 2.1 Encrypted Bloom filters

Bellovin and Cheswick [1], introduced encrypted Bloom filters. They use a Bloom filter but changing the family of hash functions by the Pohlig-Hellman encryption algorithm [28], which has the property that $E_{k_j}(E_{k_i}(x)) = E_{k_i \circ k_j}(x)$, where $E_{k_i}(x)$ denotes the encryption of $x$ with key $k_i$. To introduce a value $x$ in the filter, they compute $E_{k_A}(x)$ and divide the result in parts of size $\lceil log_2 m \rceil$, so the length of the output determines the value of $k$ in the filter (number of hash functions in the original model).

The scenario proposed by this work is that of privacy preserving searches. That is, given a user $B$ with a set of values, another user $A$ can send an encrypted value to $B$. Then $B$ can check if the value is in its set without gaining knowledge about the value sent by $A$. To that end a trusted third party (TTP) is used to avoid key distribution problems. The user $A$ computes $E_{k_A}(x)$ and sends it to the TTP. The TTP has all values $r_{ij} = k_j \circ k_i$ for all $i, j$, and can then compute $E_{k_B}(x) = E_{k_{A \circ B}}(E_{k_A}(x))$. The value $E_{k_B}(x)$ is returned to $A$. Now $A$ can send this value to $B$, who can check if the respective value $x$ is in the filter.

There are similar approaches for encrypted bloom filters such as [17], which uses an encryption functions as simple hash functions, however the approach requires the generation and distribution of keys, which indirectly can lead to the use of a TTP if we consider a generic distributed scenario. In [27], the authors attempt to remove the need for a TTP, however that comes at the expense of using public key cryptography, either with blind signatures or using oblivious pseudorandom functions, which require at least a modular exponentiation.

In the approach proposed in this paper, the use of a TTP or a complex key distribution scheme is avoided. Moreover, although the Pohlig-Hellman algorithm is relatively efficient, it is still expensive when compared with the original Bloom filter model. We therefore opt to use the Bloom filter in its original formulation. We will argue that, assuming a suitable configuration of the Bloom filter, we can obtain a tolerable degree of security without relying on the use of more complex cryptographic operations.

## 2.2 False positives in Bloom filters

Bloom filters may present false positives. This section discusses several definitions of the false positive rate of Bloom filters, which have been considered in the literature

The first estimation of the false positive rate of Bloom filters appeared in [26]. Given a Bloom filter of size $m$, a family of $k$ hash functions and $n$ elements, this first approach (denoted as $fp1$) is given by:

$$fp1(k, n, m) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \tag{1}$$

That is, if $p$ is the probability that a specific bit is still 0 after adding $n$ elements in the filter, then:

$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$
$$fp1(k,n,m) = (1-p)^k \approx (1 - e^{-kn/m})^k$$

Departing from Equation (1), the false positive rate can be minimised in terms of the parameter $k$. Thus, given $m$ and $n$, an optimal configuration is achieved for:

$$k = \frac{m}{n} \ln 2 \qquad (2)$$

Moreover, in this case we have that $fp1(k,n,m) = (1/2)^k \approx (0.6185)^{m/n}$ [5].

Recently, in [4] the authors found Equation (1) to be slightly imprecise. They determined the probability of a false positive in a Bloom filter as:

$$fp2(k,n,m) = \frac{1}{m^{k(n+1)}} \sum_{i=1}^{m} i^k i! \binom{m}{i} \left\{ {kn \atop i} \right\} \qquad (3)$$

where $\left\{ {kn \atop i} \right\}$ is the Stirling number of second kind [15], defined as:

$$\left\{ {kn \atop i} \right\} = \frac{1}{i!} \sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} j^{kn}$$

In the same paper [4] the authors provide an upper and lower bound for $fp2(k,n,m)$.

$$p^k < fp2(k,n,m) \leq p^k \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right) \qquad (4)$$

where $p = 1 - \left(1 - \frac{1}{m}\right)^{kn}$. Note that $p^k = fp1(k,n,m)$ (Equation (1)), that is, the original estimation of the false positive rate is actually a strict lower bound.

In [21] the authors arrive at an equivalent formulation of Equation (3), confirming the validity of this new estimation:

$$fp3(k,n,m) = \frac{m!}{m^{k(n+1)}} \sum_{i=1}^{m} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^{kn} i^k}{(m-i)! j! (i-j)!} \qquad (5)$$

Given the current state of the art, $fp1$ is considered as a first approximation of the false positive ratio of Bloom filters, while $fp2$ and $fp3$ do provide the exact estimation. The error in the original estimation of the false positive given by $fp1$ from Equation (1) is that given the result of the hash functions $y_1, \ldots, y_k$ for a given element, the events $\mathcal{B}(y_i)$ and $\mathcal{B}(y_1) = \mathcal{B}(y_2) = \ldots = \mathcal{B}(y_k) = 1$ were assumed to be independent. This was found to be imprecise, that is,

$$Pr(B_i = 1 \mid B_1 = \ldots = B_{i-1} = 1) > Pr(B_i = 1)$$

Despite this recent estimation, most authors still regard $fp1$ as the false positive ratio, since it is easier to handle, and the error if compared to $fp2$ is relatively minimum [19]. Moreover, the relative error diminishes with large values of $m$, where large is considered to be $m \geq 1024$ bits [21]. For the sake of simplicity in this paper we will consider the events $\mathcal{B}(y_i)$ and $\mathcal{B}(y_1) = \mathcal{B}(y_2) = \ldots = \mathcal{B}(y_k) = 1$ to be independent. We also consider relatively large filters reducing the error that this last assumption may introduce.

## 2.3 Further properties of Bloom filters

### 2.3.1 Filter intersection and union

Set operations such as union and intersection can be carried out on the filters, which can be regarded as sets of elements. Consider two sets of elements $A$ and $B$ encoded in the Bloom filters $\mathcal{B}(A)$, $\mathcal{B}(B)$ respectively, with the same size $m$ and $k$ hash functions. Then:

- The *union* of $\mathcal{B}(A)$ and $\mathcal{B}(B)$ (denoted as $\sqcup$) can be computed by a bitwise OR operation on the bit vectors corresponding to each filter. That is, $\mathcal{B}(A \cup B) = \mathcal{B}(A) \sqcup \mathcal{B}(B)$.
- The *intersection* of $\mathcal{B}(A)$ and $\mathcal{B}(B)$ (denoted as $\sqcap$) can be computed by a bitwise AND operation on the bit vectors corresponding to each filter, $\mathcal{B}(A \cap B) = \mathcal{B}(A) \sqcap \mathcal{B}(B)$, with probability [16]:

  $$Pr(\sqcap) = \left(1 - \frac{1}{m}\right)^{k^2 \cdot |A \backslash A \cap B| \cdot |B \backslash A \cap B|} \tag{6}$$

  Contrary to the union, the intersection of Bloom filters does not always correspond to the intersection of their relative sets.

### 2.3.2 Vector subset ordering

We define an ordering $\sqsubseteq$ on Bloom filters, based on set inclusion, that is, vector subset. Given two sets of elements $A$ and $B$ and their respective Bloom filters, $\mathcal{B}(A)$ and $\mathcal{B}(B)$, then the following holds:

$$\mathcal{B}(A) \sqsubseteq \mathcal{B}(B) \iff A \subseteq B$$

This filter ordering can be expressed in terms of filter intersection as:

$$\mathcal{B}(A) \sqsubseteq \mathcal{B}(B) \Longrightarrow \mathcal{B}(A) \sqcap \mathcal{B}(B) = \mathcal{B}(A) \tag{7}$$

In this case the probability of obtaining the intersection by the bitwise AND operation will always be $Pr(\sqcap) = 1$, since $|A \backslash A \cap B| = 0$ (see Equation (6)). We also note that the inverse is not true for the general case, that is: $\mathcal{B}(A) \sqcap \mathcal{B}(B) = \mathcal{B}(A) \nRightarrow \mathcal{B}(A) \sqsubseteq \mathcal{B}(B)$. However the event of this inverse relation not being true is unlikely. In order for this to happen the filter $\mathcal{B}(A)$ has to have at least an element $x$ not contained in $A \cap B$ which overlaps with the other elements in $A$. That is, a false positive. Thus, the following is considered to hold:

$$\mathcal{B}(A) \sqcap \mathcal{B}(B) = \mathcal{B}(A) \Longrightarrow \mathcal{B}(A) \sqsubseteq \mathcal{B}(B) \tag{8}$$

with probability $fp(k, n, m)$.

It is important to note that the binary relation $\sqsubseteq$ is an order relation since it is:

- *Reflexive*: $\mathcal{B}_i \sqsubseteq \mathcal{B}_i$ for any Bloom filter $\mathcal{B}_i$ as determined by Equation (8).
- *Transitive*: $\mathcal{B}_i \sqsubseteq \mathcal{B}_j, \mathcal{B}_j \sqsubseteq \mathcal{B}_k \implies \mathcal{B}_i \sqsubseteq \mathcal{B}_k$, which holds given the transitivity of the bitwise AND operation and Equation (8).
- *Anti-symmetric*: $\mathcal{B}_i \sqsubseteq \mathcal{B}_j, \mathcal{B}_j \sqsubseteq \mathcal{B}_i \implies \mathcal{B}_i = \mathcal{B}_j$, which can be directly deduced from Equation (8).

These properties are ensured on the basis of the false positive probability noted above. It will be demonstrated in later section that this relation can be used to determine a partial ordering or lattice on a set of Boom filters.

# 3 Bloom Permissions

## 3.1 Hash chains as total orderings of permissions

Bloom Permissions are inspired by previous work on micropayment systems [10, 31, 32], which make use of a hash chain to represent coins, and where the knowledge of a value from a hash chain entitles a user to use the coin. In [10] it is observed that such hash-chains can also be used to provide a rudimentary form of authorisation delegation. In this scheme, permissions are represented by hash values and a principal generating a hash chain $h^n(s)$, defines a total ordering between hash-permission values as $h^n(s) \sqsubseteq h^{n-1}(s) \sqsubseteq \ldots h(s) \sqsubseteq s$, where $s$ is secret and represents the highest permission and $h^n(s)$ is the lowest permission and is known by all. In the absence of knowledge of the secret $s$, a third party delegated permission value $h^i(s)[i \leq n]$, implicitly holds 'lower' permissions $h^j(s)[i \leq j]$ under $\sqsubseteq$. The hash values can be used as secret permission credentials that confer access rights to the holder. However, [10] is not useful as a general authorization system since it only support sets of permissions that form *total* orderings.

## 3.2 Bloom filters as partial orderings of permissions

Let $(P, \leq)$ be a lattice of permissions $P$ with a supremum denoted as $\top$ and infimum denoted as $\bot$. Given $x, y \in P$ then, the ordering $x \leq y$ is interpreted to mean that if a user holds permission $y$ then the user also implicitly holds permission $x$. Note that techniques such as [11, 12] can be used to transform arbitrary partial orders of permissions to a lattice ordering.

The Bloom filter $\mathcal{B}$ forms a lattice under vector subset $\sqsubseteq$ that is isomorphic to the permission lattice $(P, \leq)$ under the mapping $\mathcal{B}(a)$ for values $a \in P$. Thus, we have that (with probability bounded by $fp(k, n, m)$), for $a, b \in P$ then:

$$a \leq b \Leftrightarrow \mathcal{B}(a) \sqsubseteq \mathcal{B}(b)$$

The lattice ordering $(P, \leq)$ is also isomorphic to the powerset lattice with subset ordering $\subseteq$ under the mapping $\lceil a \rceil = \{x \in P \mid a \leq x\}$, that is we have for $a, b \in P$ then [7]:

$$a \leq b \Leftrightarrow \lceil a \rceil \supseteq \lceil b \rceil$$

Therefore, we have (with probability bounded by $fp(k, n, m)$),

$$a \leq b \Leftrightarrow \mathcal{B}(\lceil b \rceil) \sqsubseteq \mathcal{B}(\lceil a \rceil)$$

We denote this isomorphic lattice as $(\mathbf{B}, \sqsubseteq)$, where each element in $\mathbf{B}$ is a Bloom filter that encodes the corresponding set from the powerset of $(P, \leq)$. A Bloom filter belonging to $\mathbf{B}$ is called a *Bloom permission*.

As an example consider the lattice from Figure 1. The corresponding isomorphic powerset lattice is depicted in Figure 2, and the Bloom permission lattice is built by replacing each subset by its corresponding Bloom filter.

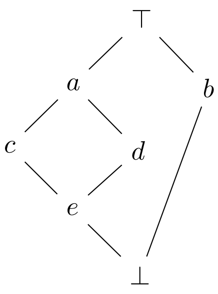

Figure 1: Example of permission lattice.

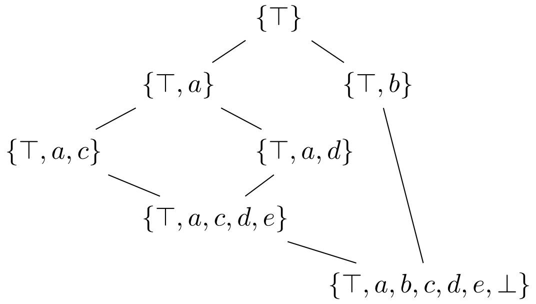

Figure 2: Lattice isomorphic to the example in Figure 1.

## 3.3 Securing Bloom Permissions

The owner of the policy $(P, \leq)$ generates a set of Bloom permissions of the form $\mathcal{B}(\lceil a \rceil)$ for each $a \in P$. These are the permission values that are used as delegation tokens. Following the examples in Figures 1, and 2, Figure 3 depicts the Bloom permission lattice generated by the authority.

A principal that knows the value for (holds) a Bloom permission $\mathcal{B}(\lceil x \rceil)$ is considered to be authorized for an action requiring permission $x \in P$. The delegation mechanism works on the basis that it is not feasible for a principal to compute a Bloom permission $\mathcal{B}(\lceil x \rceil)$ unless the principal holds (has been delegated) the value for some $\mathcal{B}(\lceil y \rceil)$, where $x \leq y$. In the proposed scheme, the supremum permission $\top \in P$ is treated as a seed value and is assumed to be a secret that is known only to the owner/creator of the policy $(P, \leq)$; all other permission values $P \setminus \{\top\}$ are considered to be publicly available/known values. Thus, Bloom permission $\mathcal{B}(\lceil \top \rceil) = \mathcal{B}(\{\top\})$ is also secret as are all other Bloom permissions $\mathcal{B}(\lceil x \rceil)$ unless otherwise revealed to a principal. Section 4 provides a security analysis of this scheme.

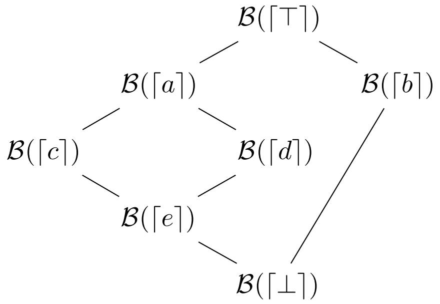
$\mathcal{B}(\lceil \top \rceil)$
$\mathcal{B}(\lceil a \rceil)$
$\mathcal{B}(\lceil b \rceil)$
$\mathcal{B}(\lceil c \rceil)$
$\mathcal{B}(\lceil d \rceil)$
$\mathcal{B}(\lceil e \rceil)$
$\mathcal{B}(\lceil \bot \rceil)$

Figure 3: Example of Bloom permission lattice.

## 3.4 Authorisation Delegation

Given a principal holding Bloom permission $\mathcal{B}(\lceil y \rceil)$, for $y \in P$, then in order to (attempt to) delegate permission $x \in P$, the principal computes $\mathcal{B}(\lceil y \rceil) \sqcup \mathcal{B}(\lceil x \rceil \setminus \{\top\})$. Note that the principal may not know the secret $\top$ and so can only compute $\lceil y \rceil \setminus \{\top\}$. Since the set $\mathbf{B}$ of all Bloom permissions forms a lattice then it follows that

$$x \leq y \implies \mathcal{B}(\lceil x \rceil) = \mathcal{B}(\lceil y \rceil) \sqcup \mathcal{B}(\lceil x \rceil \setminus \{\top\}) \tag{9}$$

Thus, the computed Bloom permission for $x$ will be valid only if the principal carrying out the delegation does know $\mathcal{B}(\lceil y \rceil)$ and $x \leq y$. Otherwise the generated Bloom permission will not correspond to any permission from the policy (the Bloom permission lattice). This is consistent with the interpretation of the permission ordering: given $x, y \in P$ then $x \leq y$ means that if a user holds permission $y$ then the user also holds (and may delegate) the permission $x$.

As an example, consider the permission lattice from Figure 1 and a user who holds permission $a$. This user has the Bloom permission $\mathcal{B}(\lceil a \rceil)$, and wants to delegate permission $e$ to another user. The user constructs the Bloom filter $\mathcal{B}(\{c, d, e\})$ and then easily computes $\mathcal{B}(\lceil e \rceil) = \mathcal{B}(\lceil a \rceil) \sqcup \mathcal{B}(\{c, d, e\})$, or equivalently, in the generic case $\mathcal{B}(\lceil e \rceil) = \mathcal{B}(\lceil a \rceil) \sqcup \mathcal{B}(\lceil e \rceil \setminus \top) = \mathcal{B}(\lceil a \rceil) \sqcup \mathcal{B}(\{a, c, d, e\})$.

### 3.5 Authorisation Verification

A principal presents Bloom permission (bit vector) $X$ to a service owner (verifier) who requires permission $x$ for authorization for some requested action. This request is authorized if and only if the following is verified to be true.

$$\mathcal{B}(\lceil x \rceil) = X \tag{10}$$

One might be tempted to permit a requester to present a Bloom permission $Y$ corresponding to permission $\mathcal{B}(\lceil y \rceil)$, where $x \leq y$ and have the verifier test $\mathcal{B}(\lceil x \rceil) \sqsubseteq Y$. However, this is not secure since any requester can present a Bloom filter corresponding to the empty set $\emptyset$ and we have $\mathcal{B}(\lceil x \rceil) \sqsubseteq \emptyset$ under the Bloom permission ordering.

Alternatively, suppose that the requesting principal holds permission $y \in P$ such that $x \leq y$ and requests an action requiring permission $x$. In this case he/she can easily compute $\mathcal{B}(\lceil x \rceil)$ from $\mathcal{B}(\lceil y \rceil)$, as $\mathcal{B}(\lceil x \rceil) = \mathcal{B}(\lceil y \rceil) \sqcup (\mathcal{B}(\lceil x \rceil) \setminus \mathcal{B}(\{\top\}))$. Note that while the requester could present $\mathcal{B}(\lceil y \rceil)$ and the verifier could compute $\mathcal{B}(\lceil x \rceil)$, there is the risk that the requester implicitly delegates a permission $\mathcal{B}(\lceil y \rceil)$ that the verifier previously did not hold.

### 3.6 Bloom permission authentication

In this paper we do not prescribe how the Bloom permissions might be securely exchanged between principals. In the case of delegation, the delegator must securely transfer value $\mathcal{B}(\lceil x \rceil)$ to a recipient. This could be done over a secure network connection, if available, or via a physical exchange of the value; for example, a transfer of the Bloom permission from a reader to a smart-card. In [23] a wireless sensor network application is outlined where Bloom permissions are installed at sensor deployment or via Handheld sensor-access devices.

In the case of verification, the requester must prove knowledge of the required permission/secret to the verifier. In addition, in order to prevent an implicit delegation, the requester needs to be sure that the verifier also holds the given permission. This can be achieved by a mutual authentication protocol whereby both parties prove that the know the secret (Bloom permission): Bloom permissions can be treated as secrets (keys) that are shared between principals. The reader is referred to [14, 20, 35] for examples of lightweight authentication protocols that are also implemented in terms of one-way cryptographic hash functions.

# 4 Security of Bloom Permissions

The security of the previously described scheme is based on the difficulty of forging a Bloom permission.

**Proposition 1** *It is not feasible for a principal to compute $\mathcal{B}(\lceil x \rceil)$ unless the principal already knows the value of some $\mathcal{B}(\lceil y \rceil)$, where $x \leq y$.*

Assume a principal attempts to compute $\mathcal{B}(\lceil x \rceil)$ without knowledge of any Bloom permission $\mathcal{B}(\lceil y \rceil)$ such that $x \leq y$. There are three possible cases:

1. Compute $\mathcal{B}(\lceil x \rceil)$ from scratch.
2. Compute $\mathcal{B}(\lceil x \rceil)$ knowing the permission set $P \setminus \{\top\}$ but not their corresponding Bloom permissions.
3. Compute $\mathcal{B}(\lceil x \rceil)$ knowing the permission set $P \setminus \{\top\}$ and some Bloom permissions $\mathcal{B}(\lceil z \rceil)$ such that $z \leq x$.

In order for the proposed scheme to be secure, all three cases must be difficult to achieve. This difficulty can be ensured by the properties of a suitably configured Bloom filter, as will be shown in this section. The first case, computing $\mathcal{B}(\lceil x \rceil)$ without knowledge of any other information can be regarded as a brute-force attack, which is clearly unfeasible in the general case. We will discuss the other two cases.

**Proposition 2** *It is not feasible for a principal to compute $\mathcal{B}(\lceil x \rceil)$ when knowing only the permission set $P \setminus \{\top\}$.*

To be able to compute $\mathcal{B}(\lceil x \rceil)$, the principal cannot use $\lceil x \rceil$, since he does not know $\top$. The principal, however, might try to guess $\mathcal{B}(\{\top\})$ in order to compute $\mathcal{B}(\lceil x \rceil) = \mathcal{B}(\{\top\}) \sqcup \mathcal{B}(\lceil x \rceil \setminus \{\top\})$.

Consider an empty Bloom filter. In this case, the pricipal needs to guess the position of the $k$ bits corresponding to the element $\top$ in the filter vector. Assume that the $k$ bit positions are selected randomly. There are $\binom{m+k-1}{k}$ possible combinations (note that a bit can be selected more than once for the same element), and thus the probability for the principal to forge $\mathcal{B}(\{\top\})$ without the knowledge of $\top$ is $P_0 = \binom{m+k-1}{k}^{-1}$. Figure 4 depicts this probability for a filter with $m = 1024$ and $k : 0..50$. This attempt to forge $\mathcal{B}(\{\top\})$ can be seen as a brute force attack on $\mathcal{B}(\{\top\})$, which becomes very difficult even with low values of $k$. Thus, assuming that $\top$ is secret ensures that it is not feasible for a principal to create arbitrary Bloom filters containing valid permissions that have not been delegated to the principal.

Consider the more general case, whereby the principal might not only know $P \setminus \{\top\}$, but also may also know any Bloom permission $\mathcal{B}(\lceil z \rceil)$ such that $z \leq x$.

**Proposition 3** *It is not feasible for a principal to compute $\mathcal{B}(\lceil x \rceil)$ from $P \setminus \{\top\}$ and any $\mathcal{B}(\lceil z \rceil)$ such that $z \leq x$.*

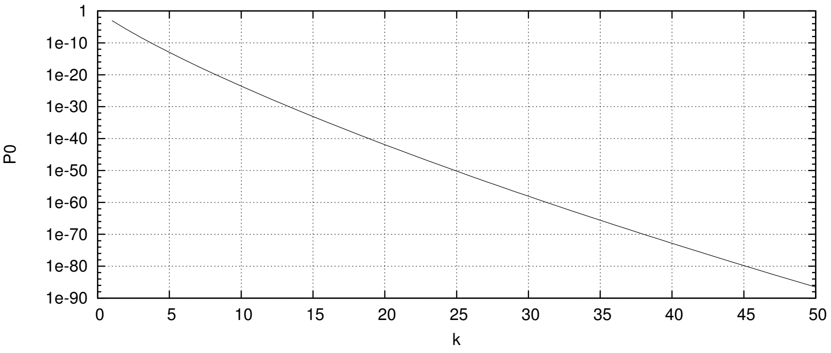

Figure 4: Probability of forging $\mathcal{B}(\{\top\})$.

Note that permission $x$ could also be the top permission $\top$, thus the proposition can be paraphrased as: is not feasible for a principal to compute $\mathcal{B}(\{\top\})$ even with the knowledge of $P \setminus \{\top\}$ and any Bloom permission other than $\mathcal{B}(\{\top\})$. There are several ways in which an attacker might attempt to use Bloom permissions $\mathcal{B}(\lceil z \rceil)$ such that $z \leq x$ in order to obtain $\mathcal{B}(\lceil x \rceil)$.

- By removing elements from an existing filter. In this case the attacker has the filter $\mathcal{B}(\lceil z \rceil)$ and attempts to remove all elements $\lceil z \rceil \setminus \lceil x \rceil$ from the filter by setting some given bits to 0 in the filter. This case is considered in Section 4.1.

- By filter intersection. In this case the attacker attempts to intersect two filters to obtain the required element. The success of the attack will be determined by the probability of intersecting Bloom filters given by Equation (6). Section 4.2 discusses this case.

## 4.1 Removing elements from a Bloom filter

As previously stated, a property that makes Bloom filters particularly suited to security applications, is the fact that removing an element from a filter is difficult. It is interesting to note that this is normally considered a drawback of Bloom filters, and there are versions of Bloom filters that allow the removal of elements [9]. In the proposed scheme the difficulty of element removal is important as it ensures the difficulty of forging Bloom permissions.

By removing one or more elements, a principal who holds a Bloom permission can try to obtain $\mathcal{B}(\{\top\})$ but also can attempt to scale privileges. For instance, from the example in Figure 3, a user who holds permission $d$, has the Bloom permission $\mathcal{B}(\lceil d \rceil) = \mathcal{B}(\{\top, a, d\})$. This user can try to remove the element $d$ from the filter, and end up with a filter equivalent to $\mathcal{B}(\{\top, a\})$, thus gaining the Bloom permission that implements $a$.

To remove a given element $x$ from a Bloom filter one must set some specific bits $\mathcal{B}[x_i] = 0$ while being aware that:

- Setting $\mathcal{B}[x_i] = 0$ may cause the deletion of other elements if they overlap the same bit $\mathcal{B}[x_i]$.
- Given that filter equality is going to be used to test the authorisation (see Section 3.5), one has to remove all bits $\mathcal{B}[x_i]$ corresponding to the element $x$, if and only if they have not been overlapped by other elements. Note that if instead, we only used element inclusion to test the resulting filter, it will be enough to remove one bit belonging to element $x$ that has not been overlapped.

This last point is the primary reason to require filter equality as an authorisation proof as opposed to filter inclusion.

For example, consider the case where we want to remove the element $x$ from the Bloom filter $\mathcal{B}(X)$, and assume we know that $x \in X$. We know the element $x$, and thus $\mathcal{B}(\{x\})$: this means that we know the $k$ values $f_i(x) = x_i$, which are the bits $B[x_i]$ modified by inserting the element $x$ in the filter.

- If there is no overlapping bit, removing the element $x$ can be done simply by setting $B[x_i] = 0$ for $i = 1, \ldots, k$.
- On the other hand, if there are overlapping bits, then not all $B[x_i]$ bits can be set to 0, since by doing so we might be deleting other elements from the filter.

We are interested in configurations of Bloom filters that have a high probability of overlapping bits. That is, a bit set to 1 in the vector representing the Bloom filter corresponds to more than one element.

Let $E1$ be the event that at least one bit is overlapped by more than one element. That is, the same bit $B[i] = 1$ for more than one element. Then, let $E0$ be the event that no element overlaps any bit. So $Pr(E1) = 1 - Pr(E0)$.

Note that $Pr(E0) = 0$ if $kn > m$, and $Pr(E0) = 1$ if $kn \leq 1$,

$$Pr(E0) = \frac{m(m-1)\ldots(m-nk+1)}{m^{nk}} = \frac{m^{\underline{nk}}}{m^{nk}} \tag{11}$$

$$Pr(E1) = 1 - Pr(E0) = 1 - \frac{m^{\underline{nk}}}{m^{nk}} \tag{12}$$

where $x^{\underline{n}}$ denotes the falling factorial $x^{\underline{n}} = x \cdot (x-1) \cdot \ldots \cdot (x-(n-1))$. Figure 5 shows $Pr(E1)$ for a filter with $m = 1024$, and $n = 50$. As it can be seen with $k = 2$ the probability becomes significantly close to 1 (0.993).

## 4.2 Aggregation attacks

The lowest upper bound operator in the permission lattice can be used to define permission aggregation: a principal holding permissions $a, b \in P$ may be considered to hold their aggregation $a \oplus b \in P$. In general, if $a \oplus b$ is the lowest upper bound of permissions $a$ and $b$ of lattice $P$ then it follows that

$$\lceil a \oplus b \rceil = \lceil a \rceil \cap \lceil b \rceil$$

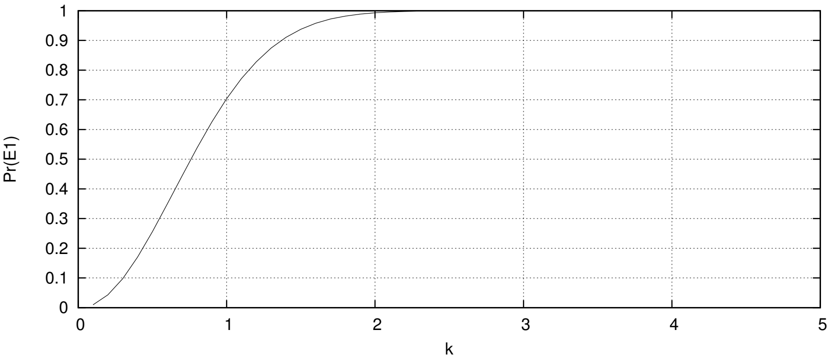

Figure 5: Probability of bit overlapping for a Bloom filter with $m = 1024$, and $n = 50$.

$$\begin{tikzcd}
 & \mathcal{B}(\{\top\}) \arrow[dl] \arrow[dr] & \\
\mathcal{B}(\{\top, a\}) & & \mathcal{B}(\{\top, b\})
\end{tikzcd}$$

Figure 6: Worst case scenario for the aggregation attack.

Therefore, Bloom permission aggregation (lowest upper bound) can be implemented in terms of set intersection and this preserves permission ordering under the isomorphism.

**Proposition 4** *It is not generally safe to distinguish between knowledge of individual permissions and their aggregation. That is, given $a, b \in P$ where $a \oplus b \in P$ is the lowest upper bound of $a$ and $b$, then it is not safe to assume that it is not feasible for a principal, who knows Bloom permissions $\mathcal{B}(\lceil a \rceil)$ and $\mathcal{B}(\lceil b \rceil)$, to compute their lowest upper bound $\mathcal{B}(\lceil a \oplus b \rceil)$.*

A simplistic worst case scenario of a successful attack can be demonstrated using the ordering in Figure 6. In this case, a principal who knows $\mathcal{B}(\lceil a \rceil)$ and $\mathcal{B}(\lceil b \rceil)$ can obtain $\mathcal{B}(\{\top\})$ by attempting to compute the intersection as $\mathcal{B}(\{\top\}) = \mathcal{B}(\lceil a \rceil) \sqcap \mathcal{B}(\lceil b \rceil)$. That is, given the sets $A = \{\top, a\}$, and $B = \{\top, b\}$, compute $A \cap B = \{\top\}$, on the filters. Recall that the user does not know the elements contained in the filters. The probability of success, given by the probability of Bloom filter intersection from Equation (6), will be determined firstly by the parameters $m$ and $k$ of the filters. If we consider $m = 1024$, the probability of the success on the intersection in terms of $k$ is shown in Figure 7.

Although it might seem a relatively high probability from a security point of view, we have to consider the other parameter that affects this probability. As shown in Equation (6), the success in the attack also depends on the value

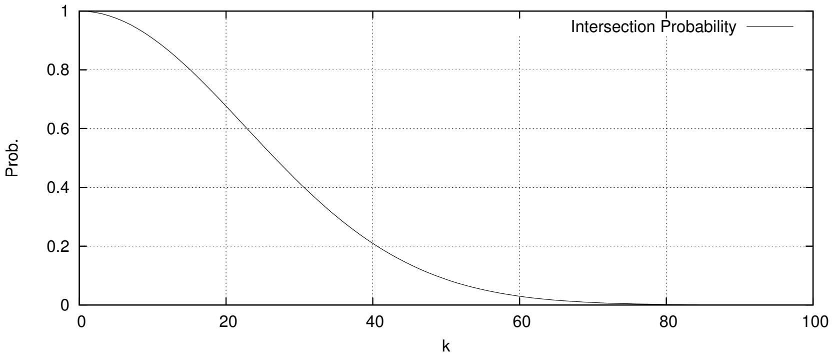

Figure 7: Probability of filter intersection with $m = 1024$ for $k : 0..10$.

of $|A \setminus A \cap B| * |B \setminus A \cap B|$. Let $\rho$ denote the number of elements in $A$ but not in $A \cap B$, that is $\rho = |A \setminus A \cap B|$, and assume that it is equal to $|B \setminus A \cap B|$. As $\rho$ increases, the probability of intersection is reduced as depicted in Figure 8. In this case, if we take $n = 50$ the value of $k$, which minimises the false positive probability following Equation (2) is $k = m/n \ln 2 \simeq 14.2$, so with $k = 14$ we have a false positive of approximately $fp1(k, n, m) = 5.36 \times 10^{-5}$, and with $\rho = 5$ a probability of intersection $Pr(\sqcap) = 8.3 \times 10^{-3}$, which we consider to be quite acceptable. Moreover one can sacrifice the false positive rate by increasing, to a small degree, the value of $k$ which provides a lower $Pr(\sqcap)$ if a lower $\rho$ is expected. For example, Figure 9 depicts the intersection probability in filters with $m = 1024$, $k = 14$ ($n = 50$), for $\rho : 0..10$

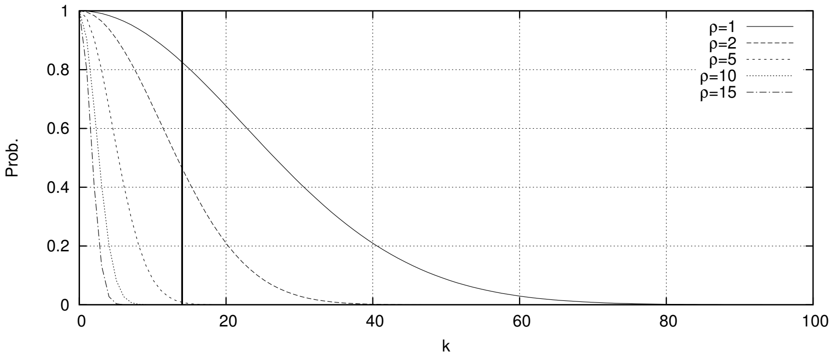

Figure 8: Probability of intersection for $\rho \in \{1, 2, 5, 10, 15\}$. If $n = 50$ then optimal configuration can be achieved with $k = 14$.

It is necessary to select a trade-off between the probabilities of false positives and intersection. Increasing $k$ we can reduce the probability of success for this attack, but it comes with the cost of increasing the probability of false positives. Table 1 shows the false positive ($fp1$), and intersection probabilities $Pr(\sqcap)$ for

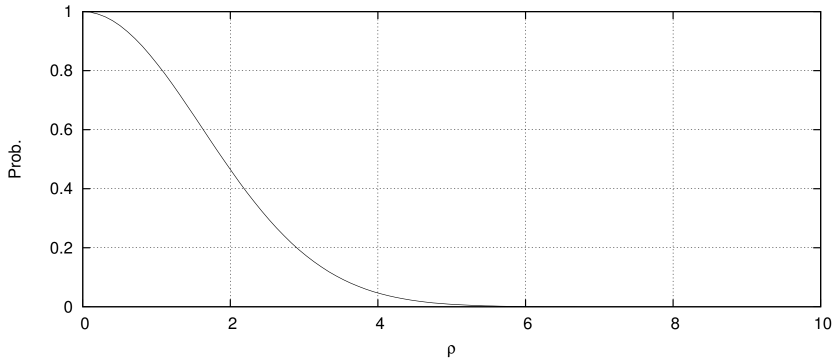

Figure 9: Probability of intersection for filters with $m = 1024$, $k = 14$, for $\rho : 0..10$.

some different values of $k$ and $\rho$.

| $k$ | $\rho$ | $fp1$ | $Pr(\sqcap)$ |
|---|---|---|---|
| 14 | 5 | $5.36 \times 10^{-5}$ | $8.3 \times 10^{-3}$ |
| 20 | 3 | $7.90 \times 10^{-5}$ | $1.96 \times 10^{-2}$ |
| 25 | 3 | $1.61 \times 10^{-4}$ | $4.10 \times 10^{-3}$ |
| 30 | 3 | $3.79 \times 10^{-4}$ | $3.66 \times 10^{-4}$ |
| 30 | 2 | $3.79 \times 10^{-4}$ | $2.97 \times 10^{-2}$ |
| 35 | 2 | $9.26 \times 10^{-4}$ | $8.33 \times 10^{-3}$ |
| 40 | 2 | $2.22 \times 10^{-3}$ | $1.92 \times 10^{-3}$ |

Table 1: False positive and intersection probabilities for different values of $k$ and $\rho$ in filters with $m = 1024$, and $n = 50$.

Many lattice-based permission models rely on an assumption that if a principal holds a number of permissions then the principal also holds the lowest upper bound on those permissions. For example, holding a read permission and a write permission implies possession of a read-write permission. The proposed Bloom permission model effectively requires this assumption since a principal has the potential to determine $\mathcal{B}(\lceil a \oplus b \rceil)$ from knowing $\mathcal{B}(\lceil a \rceil)$ and $\mathcal{B}(\lceil b \rceil)$. In practice, this means that the potential for permission aggregation to escalate authority must be considered during the design of the permission ordering. For example, returning to the lattice in Figure 1, a principal that holds permissions $c$ and $b$, implicitly holds $\top = c \oplus b$, and therefore also holds permissions $a$ and $d$. If this escalation is considered undesirable, then one could modify the original lattice to include a distinct permission to reflect the aggregation of $c$ and $b$. In general, a powerset lattice of individual permissions could be used to provide distinct aggregate permissions.

# 5 Discussion and related work

Bloom permissions are a form of software capability whereby possession of the capability entitles the principal to engage in specified actions. Many existing applications, ranging from web authorization cookies [13] to embedded systems [35] implement forms of software capabilities based on a keyed one-way hash of the permissions. For example, on receipt of capability $h_k(y)$ for a request requiring permission $x$, the verifier checks $x \leq y$ and validates the capability using a secret $k$ known only to the verifier. The holder of a capability $h_k(y)$ can delegate it to a third party by revealing the value $h_k(y)$. However, without knowing the secret $k$, the holder cannot delegate a lesser capability $h_k(x)$ where $x < y$. This issue could be solved if in being delegated authority for $y$, the recipient was given a copy of every capability $h_k(x)$, where $x \leq y$. However, this is undesirably complex and is not practical for low-power/computation systems. The recipient of a Bloom permission $\mathcal{B}(\lceil y \rceil)$ can compute, and therefore further delegate, any permission $\mathcal{B}(\lceil x \rceil)$, where $x \leq y$; this computation does not require knowledge of the underlying secret $\top$. Thus, Bloom permission delegation and verification does not require coordination with a central authority (the owner of $\top$) and can be decentralized across the network.

Trust Management systems [2, 8, 24] also provide a decentralized approach to delegation of permissions between public keys. Authorization is defined in terms of whether trust management capabilities (credentials) provide a delegation chain, for the given permission, from the verifier to the requester. This is unlike Bloom permissions, whereby possession of a permission is sufficient to determine authorization. Additionally, it is not possible to delegate a Bloom permission prior to holding the permission. Typical Trust Management systems allow a principal to issue a delegation certificate independent of the permissions it may hold at that time, with an assumption that other necessary credentials to make up the delegate chain will be obtained prior to verification.

Lattice-based permission orderings are found in a variety of both Mandatory and Discretionary Access Control models. In addition to providing sensitivity orderings for Multilevel security and Chinese Walls [12], role hierarchies in Role Based Access Control [34], they are used in application-level security, for example, the `implies` ordering across Java permissions. Bloom permissions can be used in the implementation of these models. In [23] a prototype of a Wireless Sensor Network is outlined whereby, among other security requirements, Bloom permissions are used to determine access to sensors by readers. The permission lattice is defined as a cartesian product of sensor categories and actions. For example, a medic's reader has permission $(\{\mathsf{fire}, \mathsf{EMT}\}, \{\mathsf{rd}\})$ reflecting authority to read fire-service and EMT-service sensors.

It has been demonstrated [22] that public key operations can be carried out by devices with limited computational power. Thus, in principle, the access control model described in this paper could be implemented by a conventional Trust Management system. However, this is not considered, as the objective of this paper is the design of an authorization model that does not rely on relatively expensive public key operations.

Verification of authorization requires evidence that the requester knows the (secret) Bloom permission. This corresponds to a mutual authentication between requester and verifier. While Section 3.6 suggests that existing authentication mechanisms could be used, investigating mechanisms that support Bloom permission authorization and delegation for specific applications such as [23] is a topic of ongoing research.

Rivest [33] notes that conventional revocation mechanisms such as Certificate Revocation Lists and the Online Certificate Revocation Protocol can put a burden on the authorization verifier. He argues that short-lived credentials move this (computational and network) burden from the verifier to the requester. This view is particularly true for the kinds of applications that are intended for Bloom permissions. A date value, defining the validity period, could be incorporated into each Bloom permission. However, this date value would have to be immutable. That is, the holder of a Bloom permission $y$ valid until date $d$ can only delegate a permission $x$ ($x \leq y$) that is also valid until the same date $d$. With the existing mechanism it would not be possible to delegate $x$ with an earlier expiry date $d'$ as it would presume the existence of a date ordering $d' \leq d$ implemented in the Bloom filter. Investigating whether this could be effectively addressed using a date-(hash) chaining mechanism such as [25] is a topic for future research.

# 6 Conclusions

A model for decentralized authorization implemented in terms of Bloom filters is proposed. Bloom filters provide a permission structure that is isomorphic to the required permission ordering with the property that it is not feasible for a principal to compute the Bloom permission for some permission $x$ unless the principal already knows the value of a Bloom permission for some $y$, where $x \leq y$ in the original permission ordering. This provides a basis for a lightweight authorization mechanism whereby knowledge of the Bloom permission value indicates authorization, and delegation and verification is implemented in terms of one-way hash function calculations.

# Acknowledgements

Partial support by the Spanish projects TIN2011-27076-C03-03 (CO-PRIVACY), TIN2010-15764 (N-KHRONOUS), ARES – CONSOLIDER INGENIO 2010 CSD2007-00004, is acknowledged. The research leading to these results has received funding from Science Foundation Ireland grant 08/SRC/11403 and the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 262608.

# References

[1] S.M. Bellovin, W.R. Cheswick. Privacy-Enhanced Searches Using Encrypted Bloom Filters. *Cryptology ePrint Archive*, 2004/022.

[2] M. Blaze, J. Feigenbaum, J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–173.

[3] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, vol. 13, no. 7, 1970.

[4] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008

[5] A. Broder, M. Mitzenmacher. Network Applications of Bloom Filters: A Survey, *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2003.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Volume 7 OSDI 2006.

[7] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 2nd edition, 2002.

[8] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. *SPKI Certificate Theory.* Request For Comments 2693 (Experimental), IETF, September 1999.

[9] L. Fan, P. Cao, J. Almeida, A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Trans. on Networking*, vol.8 no.3, pp.281-293, 2000.

[10] S.N. Foley. Using trust management to support transferable hash-based micropayments. In *Financial Cryptography*, pp. 1–14, 2003.

[11] S.N. Foley, A taxonomy for information flow policies and models, In *Proceedings of Research in Security and Privacy*, IEEE Press, 1991.

[12] S.N. Foley. The specification and implementation of "commercial" security requirements including dynamic segregation of duties. In *Proceedings of the 4th ACM conference on Computer and communications security.* ACM Press, 1997.

[13] K. Fu, E. Sit, K. Smith, N. Feamster, Dos and Don'ts of Client Authentication on the Web, In *Proceedings of Usenix Security*, pp251–268, 2001.

[14] L. Gong, Using one-way hash functions for authentication, *ACM Computer Communications Review*, Jan 1990.

[15] R. Graham, D. Knuth, and O. Patashnik *Concrete Mathematics. A foundation for Computer Science*. Addison-Wesley, 1989.

[16] D. Guo, J. Wu, H. Chen, Y. Yuan, X. Luo. The Dynamic Bloom Filters. *IEEE Trans. on Knowl. and Data Eng.* vol. 22, no. 1 (January 2010), pp. 120–133.

[17] E. Goh. Secure indexes, *Cryptology ePrint Archive*, Report 2003/216.

[18] N. Haller et. al. *A One-Time Password System*, Request for Comments 2289, IETF 1998.

[19] M.C. Jeffrey, J.G. Steffan Understanding Bloom Filter Intersection for Lazy Address-Set Disambiguation. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA 2011). pp. 345-354.

[20] A. Juels, Minimalist Cryptography for Low-Cost RFID Tags (Extended Abstract), In *Security in Communication Networks*, Springer LNCS 3352, 2005.

[21] K. Christensen, A. Roginsky, M. Jimeno. A New Analysis of the False-Positive Rate of a Bloom Filter, *Information Processing Letters*, vol. 110, no. 21, pp. 944–949, 2010.

[22] J. Lopez Unleashing public-key cryptography in wireless sensor networks, *Journal of Computer Security*, pp. 469-482, vol. 14, n. 5, 2006.

[23] E. Mercadal, G. Navarro-Arribas, S.N. Foley and J. Borrell. Towards efficient access control in a mobile agent based wireless sensor network, In *Proceedings of the International Conference on Risks and Security of Internet and Systems*, IEEE Press, 2012.

[24] N. Li and J. C. Mitchell. RT: A role-based trust-management framework. In *The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 201–212, Washington, D.C., April 2003. IEEE Computer Society Press, Los Alamitos, California.

[25] S. Micali. Efficient certificate revocation. In *Proceedings of the 1997 RSA Data Security Conference*, 1997.

[26] J. Mullin. A second look at Bloom filters, *Communications of the ACM*, vol. 26 no. 8, pp. 570–571, 1983.

[27] R. Nojima, Y. Kadobayashi. Cryptographically Secure Bloom-Filters, *Transactions on Data Privacy*, vol. 2 no. 2, pp. 131–139, 2009.

[28] S.C. Pohlig, M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE Transactions on Information Theory*, vol. IT-24, pp. 106–110, 1978.

[29] Squid-cache.org Squid FAQ, Cache Digests. In *squid-cache wiki.* `http://wiki.squid-cache.org/SquidFaq/CacheDigests`. Accessed Dec, 2012.

[30] T. Page. *The application of hash chains and hash structures to cryptography.* Technical Report RHUL-MA-2009-18, Department of Mathematics, Royal Holloway, University of London, 2009.

[31] T. P. Pedersen. Electronic payments of small amounts. In *Proc. 4th International Security Protocols Conference*, pages 59–68, 1996.

[32] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Proc. 4th International Security Protocols Conference*, pages 69–87, 1996.

[33] R.L. Rivest Can we eliminate revocation lists?, In *Proceedings of Financial Cryptography 1998.* Springer Lecture Notes in Computer Science No. 1465.

[34] R. Sandhu, et. al. Role-Based Access Control Models, In *IEEE Computer*, 29(2):38-47, Feb. 1996.

[35] B. Song and C. Mitchell, *RFID Authentication Protocol for Low-cost Tags*, in Proceedings of the first ACM conference on Wireless network security, 2008, pages 140–147.