A Perfect Collision-Free Pseudonym System

Ji Won Yoon and Hyoungshick Kim

***Abstract*—For anonymous communication, the most popular method is to use pseudonyms instead of original identities. We previously demonstrated the possibility of a *collision-free* pseudonym system by uniquely assigning a permutation element to each network entity [1]. We extend this idea by assigning k permutation elements to each entity for $k \geq 1$. We analyse the randomness of the pseudonyms used in the proposed system to show the effect of k by varying k from 1 to 50. Our experimental results show that our pseudonym system can provide practical anonymity with k.**

***Index Terms*—Collsion-free pseudonym, anonymous communication, collision-free, pseudo-random permutation.**

I. Introduction

A NETWORK entity can anonymously interact with other entities using a pseudonym instead of its original identifier. Under the assumption that pseudonyms are unlinkable to the entity's original identifier, we can provide anonymity for the entity against an adversary that seeks to gain information regarding the entity's original identifier.

The most popular method of providing pseudonyms to network entities is to use the *entity generated pseudonyms* (EP), in which each entity's pseudonyms are generated from the outputs of a one-way hash function such as SHA-2 or MD5. Since each entity can generate its own pseudonyms locally, this approach may be preferable in a distributed system. However, we cannot guarantee that this approach always satisfies the *collision-free* property. In principle, collisions of pseudonyms are inevitable due to the inherent limitation of hash functions, even though a main security requirement for a hash function is its collision resistance. It is well-known that all hash functions suffer from a generic birthday paradox based attack. More precisely, if H is a hash function that outputs h-bit values, then among the hash values of $2^{h/2}$ different messages, there exists a collision with non-negligible probability. Hash functions should therefore have an output of a reasonably long length (e.g. 160 bits) to avoid birthday paradox based attacks. In practice, however, it may be difficult to increase the size of a pseudonym without any restriction due to the legacy of existing network infrastructure. For example, a randomly chosen MAC address may conflict with other addresses since there are only 48 bits in a MAC address [2]. Alternatively, an intuitive solution is to use *centralised pseudonym assignment* (CP) to solve the collision problem. In CP, an administrator collects a set of unique pseudonyms, avoiding repetition of the same pseudonyms, and then distributes these to every network entity. This avoids pseudonym collisions, but is inefficient since the administrator distributes all entity's pseudonyms beforehand, and requires each entity to store all of its pseudonyms. Finally, in hybrid pseudonym assignment (HP), each entity's pseudonyms are locally generated and centrally administered to prevent collisions. We propose a new pseudonym assignment scheme based on permutation array (PA) theory.

Manuscript received January 10, 2011. The associate editor coordinating the review of this letter and approving it for publication was C. Mitchell.

J. Yoon is with Trinity College Dublin, Ireland.

H. Kim is with the University of Cambridge, UK (e-mail: Hyoungshick.Kim@cl.cam.ac.uk).

Digital Object Identifier 10.1109/LCOMM.2011.040711.110052

II. A Pseudonym Generator

Our pseudonym generation function is based on permutation array theory. The network entity uses an element in a particular column of a random permutation array $\mathcal{A}$ as its pseudonym, which can be locally computed by each entity. Each column of the array $\mathcal{A}$ consists of m unique values so we can obtain the *collision-free* property in our pseudonym assignment.

A. Model

We assume that there exists a trustworthy administrator $\mathbf{y}$ to generate system parameters and then distribute them to all network entities. Consider that there are n_e network entities $\mathbf{x}$ where $\mathbf{x} = \{\mathbf{x}_i\}_{i=1}^{n_e}$. For each entity $\mathbf{x}_i$, the administrator $\mathbf{y}$ randomly chooses a set of k identifiers, $\mathbf{S}_i$, where $\mathbf{S}_i = \{\eta_1^i, \cdots, \eta_k^i : \eta_l^i \in [1, m] \text{ for } 1 \leq l \leq k\}$, $\mathbf{S}_i \cap \mathbf{S}_j = \emptyset$ for $i \neq j$ and $n_e << m$ and secretly distributes them to $\mathbf{x}_i$. For delivery of $\mathbf{S}_i$, we assume a secure communication between $\mathbf{y}$ and $\mathbf{x}_i$.

For anonymous communication, each entity has to generate its pseudonyms secretly. An entity's pseudonym should be renewed over time. For simplicity, we assume that time is divided into fixed discrete time intervals $\{t_1, t_2, \cdots\}$ and all entities' pseudonyms are renewed every interval in a synchronized manner. We use t_j to denote the jth time interval. At the time interval t_j, $\mathbf{x}_i$'s pseudonym is generated by first selecting an element from $\mathbf{S}_i$ in a random manner. Without loss of generality, we let $\eta_l^i = q$ be the randomly chosen element from $\mathbf{S}_i$. Next, $\mathbf{x}_i$'s pseudonym is generated by computing the element $\mathcal{A}(q, j)$, which means the element in row q and column j. The main problem is how to compute an element of $\mathcal{A}$. The most simple way is for entities to maintain $\mathcal{A}$ and to access an element of the array $\mathcal{A}$ directly. However, it is infeasible to maintain $\mathcal{A}$ at each network entity individually since the array $\mathcal{A}$ should be sufficiently large to prevent an adversary's brute-force attack (e.g. $m \geq 2^{128}$). We therefore propose an efficient way to generate pseudonyms by calculating the desired value of $\mathcal{A}$ only by accessing values of small arrays. Therefore we obtain any value of the permutation array $\mathcal{A}$, which corresponds to a pseudonym, without knowing or loading the whole array. For distributed systems, the administrator is only required during the 'set-up' process. The administrator $\mathbf{y}$ first generates n_s small arrays $\{A_j\}_{j=1}^{n_s}$, which are used to compute an element of a random permutation array $\mathcal{A}$, and publishes them to all network entities. Since the small arrays $\{A_j\}_{j=1}^{n_s}$ are public information, given a pseudonym $\mathcal{A}(q, j)$ at t_j, an adversary can try to find the secret identifier q by computing an element of the array $\mathcal{A}$ and comparing it with $\mathcal{A}(q, j)$ sequentially.

B. Construction of an $\mathcal{A}$ from n_s small arrays

Suppose that we have a permutation array over $\Re$ of size m_j, so it is an $m_j \times m_j$ array. Let Γ_{m_j} denote the set of all $m_j!$ permutations with m_j distinct elements of some fixed set R. We use (m_j, d) PA to stand for a subset of Γ_{m_j} with the property that the Hamming distance between any two distinct permutations in the array is at least d. The PA is r-balanced if each element of R appears exactly r times in each column. Of importance are 1-balanced PAs for small arrays, since a 1-balanced $(m_1m_2, m_1m_2 - 1)$ PA can be reconstructed by a combination of a 1-balanced (m_1, m_1-1) PA and a 1-balanced $(m_2, m_2 - 1)$ PA [3]. This algorithm can be generalized by sequentially combining n_s small permutation arrays, $\{A_j\}_{j=1}^{n_s}$ to construct a large permutation array $\mathcal{A}$.

C. Efficient access to $\mathcal{A}$

We have shown that we can build a large permutation array $\mathcal{A}$ by combining several small PAs. The next questions relate to the issue of whether we can hide the information and whether we can access only the required values of the permutation array $\mathcal{A}$ without loading and calculating the whole of $\mathcal{A}$. We address this problem by using a division algorithm [1]. The general algorithm described in Algorithm 1 is a recursive function and the number of iterations is the same as the number of small arrays. As a result, our proposed method does not load or calculate the large array $\mathcal{A}$.

Algorithm 1 Access to $\mathcal{A}(x, y)$ with small arrays

```
1: for i=1 to $n_s$ do
2:     If $i = 1$, then $q_{x_1} = x$ and $q_{y_1} = y$.
       Otherwise, $q_{x_i} = \left\lfloor \frac{q_{x_i - 1}}{m_{i-1}} \right\rfloor$ and $q_{y_i} = \left\lfloor \frac{q_{y_i - 1}}{m_{i-1}} \right\rfloor$.
3:     $r_{x_i} = q_{x_i} \bmod m_i$ and $r_{y_i} = q_{y_i} \bmod m_i$
4: end for
5: for $i = n_s$ to 1 decreasing by 1 do
6:     If $i \neq n_s$, then $D_i = A_i(r_{x_i}, r_{y_i}) + m_i \times D_{i+1}$.
       Otherwise, $D_i = A_i(q_{x_i}, q_{y_i})$.
7: end for
8: $\mathcal{A}(x, y) = D_1$
```

III. Analysis

A. Advantages of our approach

Table I summarizes the characteristics among CP, EP, HP and our approach. In this table m is the column length of the array $\mathcal{A}$. In the worst case, the number of total pseudonyms for each entity is m.

TABLE I
COMPLEXITY COMPARISON FOR GENERATING m PSEUDONYMS

	CP	EP	HP	our approach
Collision	free	exist	free	free
Space complexity	$O(m)$	$O(1)$	$O(1)$	$O(\log m)$
Time complexity	$O(m)$	$O(m)$	$O(m)$	$O(m \log m)$
Message complexity	$O(m)$	–	$O(m)$	$O(\log m)$

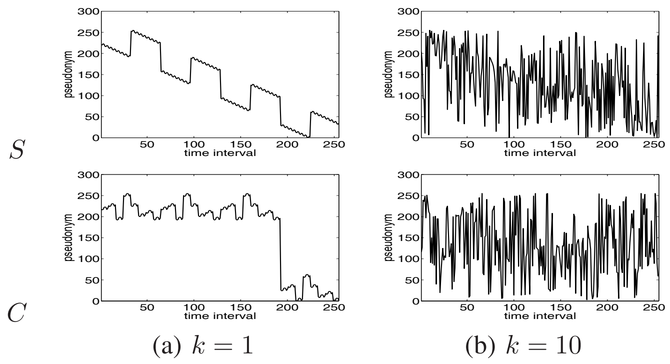


Fig. 1. Sequences of pseudonyms for an entity when $k = 1$ and $k = 10$: we used two different types of small arrays, S and C. We can see that a stepwise pattern is clearly repeated for both of them when $k = 1$.

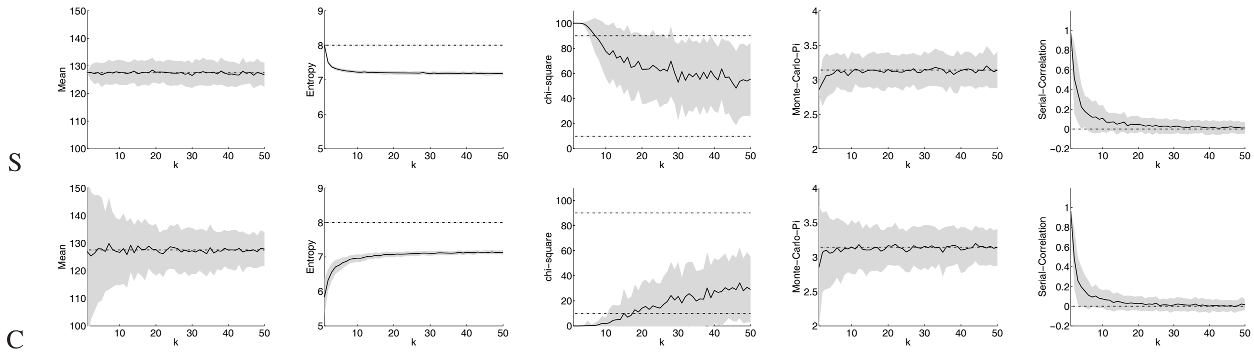
1) Collision-free: In CP and HP, a trustworthy administrator can provide the *collision-free* property by checking collisions of pseudonyms whenever it generates them while in EP collisions can happen. However, our proposed approach maintains the *collision-free* property even without a centralised collision test by an administrator since all numbers are unique in any column of the constructed 1-balanced permutation array.

2) Complexity: For generating m pseudonyms in CP, each entity should receive and maintain m pseudonyms beforehand, while each entity does not need to maintain pseudonyms beforehand in EP or HP. In CP, EP, and HP, pseudonyms can be accessed as needed in time $O(1)$ when we assume that the running time of a cryptographic hash algorithm is $O(1)$. So the total running time for m pseudonyms is $O(m)$ in these approaches. In CP and HP, $O(m)$ messages are additionally required for centralised collision detection.

In the proposed system, each entity maintains the system parameters $\{A_j\}_{j=1}^{n_s}$ where n_s is the number of small arrays. Without loss of generality, we assume that the small arrays $\{A_j\}_{j=1}^{n_s}$ are arranged in monotonically increasing size. Suppose that A_1 is an array with the size $c \times c$. By the property of our permutation array construction, we can see $c^{n_s} \leq m$. From this, we can derive $n_s \leq \log_c m$. Thus the total number of small arrays n_s is $O(\log_c m)$. Since c is a constant, the required space for each entity is $O(\log m)$ and the message complexity is also $O(\log m)$. Each entity can compute an element of a large array $\mathcal{A}$ with n_s smaller arrays by using Algorithm 1. Since the running time of this algorithm is $O(n_s)$, each entity can generate a pseudonym in time $O(\log m)$. To generate m pseudonyms, $O(m \log m)$ is required.

B. Unlinkability

To show *unlinkability* between an entity's pseudonym at time interval t_i and at time interval t_j for $i \neq j$, we have



(a) Mean (b) Entropy (c) χ^2 distribution (d) Monte Carlo π (e) Correlation

Fig. 2. Randomness testing varying k from 1 to 50: five different tests to check the randomness of pseudonyms – the means of estimated values (solid line), the standard deviations of estimated values (gray shadow) and the best values for randomness (dot-dashed line).

used well-known random number testing procedures with a sequence of the pseudonyms used for an entity. We generated pseudonyms ranging from 0 to 255 with $\mathcal{A}$ of size 256×256. In order to analyse the effect of small arrays used for generation of $\mathcal{A}$'s elements, we used two different types of small arrays: simpler (S) and relatively complicated structure (C). While 8 fixed 1-balanced (2, 1) PAs are used, all of which having the same format as $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ in S, 3 relatively larger small arrays are used with randomly assigned two 1-balanced $(8, 7)$ PAs and a 1-balanced $(4, 3)$ PA in order to build $\mathcal{A}$ in C. The upper row of graphs in Fig. 1 show the sequences of pseudonyms of a certain entity for $k = 1$ and $k = 10$ in S, whilst the bottom row represent those in C respectively. In Fig. 1, we have an important observation. When $k = 1$, a stepwise pattern is clearly visible for both graphs. With this pattern, entity's pseudonyms can be highly predictable, regardless of the complexity of small arrays. However, the complexity of the sequences dramatically increases with k. This result implies that the randomness of pseudonyms depends on not the complexity of small arrays but the size of k.

Fig. 2 shows several results of randomness testing. We have 100 iterations for each experiment with five randomness testing methods: mean, entropy, χ^2 distribution, the estimation of π by Monte Carlo and Correlation (see more details in [4]). In Fig. 2-(a), a desired mean for a perfect random sequence is 127.5. Their means are close to the desired mean even if $k = 1$, but the standard deviation of the 100 means of C is wide with small k and becomes narrower as k increases. Given the best value for the entropy, $8 = \log_2 256$, both S and C converge around 7 as k increases. For the χ^2 distribution, if the value is between 10% and 90%, it is assumed to be random according to the χ^2 test. Both S and C obtained the randomness of random sequence with $k > 10$ and $k > 15$ respectively. Also, sequences of pseudonyms with $k \geq 5$ can reach 3.141592 in both experiments by Monte Carlo simulation. The final test is to calculate the correlation between two subsequent bytes (pseudonyms). Both experiments approach the ideal value at 0.0 for the perfect random sequence with $k > 15$.

Our experiments demonstrate two key results: (a) the randomness increases as k increases as shown in Fig. 2 and (b) the proposed approach can generate practical pseudo-random sequences even if n_s is small but fixed arrays with size 2×2 if entities have privately distributed multiple keys. Through the second fact we can reduce the key size further since the administrator $\mathbf{y}$ can distribute only the number of small arrays n_s and the sets of private identifiers to entities at the initial set-up period rather than maintaining and distributing n_s small arrays.

IV. Conclusion and Future Work

We proposed a new method for *collision-free* pseudonyms among entities for anonymous communications. Our proposed approach achieves the perfect *collision-free* property of pseudonyms based on permutation array theory. Any value of the permutation array can be efficiently computed not by loading the whole array itself but by loading the small arrays, which were used to compute any element value in the large permutation array. Specifically, the proposed algorithm runs in $O(m \log m)$ time, $O(\log m)$ space and $O(\log m)$ messages to generate m pseudonym. In addition, we showed that the proposed approach can guarantee reasonable and practical randomness even with a small number of simple arrays by increasing the number of the permutation elements assigned for each entity.

Here we assume that each entity's pseudonym is updated every time interval. In practice, however, this assumption may be unacceptable since the communication session is terminated by force when a time interval occurs in the middle of a communications session. As future work, we plan to implement a more practical solution without this assumption.

References

[1] J. Yoon and H. Kim, "A new collision-free pseudonym scheme in mobile ad hoc networks," *5th Workshop on Resource Allocation, Cooperation and Competition in Wireless Networks*, June 2009.

[2] T. Jiang, H. J. Wang, and Y. Hu, "Preserving location privacy in wireless LANs," in *Proc. 5th international Conference on Mobile Systems, Applications and Services*, pp. 246–257, 2007.

[3] C. Ding, F. Fu, T. Klove, and V. Wei, "Constructions of permutation arrays," *IEEE Trans. Inf. Theory*, vol. 48, pp. 977–980, 2002.

[4] J. Walker, ENT: a pseudorandom number sequence test program, http://www.fourmilab.ch/random.